Optimal Test Trajectories for Calibrating Inertial Systems

by

Louis J. Lintereur

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author [illegible]
Department of Aeronautics and Astronautics
May 10, 1996

Certified by
Professor Eric Feron
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by [illegible]
Tom P. Thorvaldsen
Principle Member Technical Staff, Draper Laboratory
Thesis Supervisor

Accepted by [illegible]
Professor Harold Y. Wachman
Chairman, Department Graduate Committee

Optimal Test Trajectories for Calibrating Inertial Systems

by

Louis J. Lintereur

Submitted to the Department of Aeronautics and Astronautics
on May 10, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Optimal trajectories for a 2-axis gimbaled test table are generated to calibrate errors in inertial systems caused by angular motion. The covariance matrix from a Kalman filter is used to determine calibration uncertainty produced by a particular test table trajectory. A conjugate gradient optimization method is employed to compute trajectories which minimize the calibration uncertainty. Models are derived for both a ring laser gyro IMU and a micro-mechanical IMU. These models are incorporated in a Mathematica script program which is used to develop the Kalman filter and optimization equations.

The optimal trajectories that are generated do yield more accurate calibrations than traditional test table trajectories. They achieve higher accuracy by making several calibration terms simultaneously observable instead of isolating each term for a short period. Improvements in calibration accuracy of over 50% for the ring laser gyro IMU and between 15% and 60% for the micro-mechanical IMU are achieved by the optimal trajectories. However, numerical problems, especially in the ring laser gyro Kalman filter equations, are caused by high observability of certain states. They impose limitations on the test table gimbal angular rates and accelerations which make direct comparison of the calibration trajectories unclear.

Thesis Supervisor: Professor Eric Feron
Title: Assistant Professor of Aeronautics and Astronautics

Thesis Supervisor: Tom P. Thorvaldsen
Title: Principle Member Technical Staff, Draper Laboratory

Acknowledgments

I would like to express thanks to all those who helped make this research effort such an enjoyable and fulfilling learning experience. I extend my most sincere gratitude to my technical supervisor, Tom Thorvaldsen, whose knowledge, patience and endless stream of ideas were crucial to keeping me focused and on schedule. Special thanks also go to Professor Eric Feron. I was constantly inspired by his enthusiasm towards my project and his ability to uncover the most important and useful aspects of any problem. Throughout my engineering career, I will always strive to emulate the qualities of these two people.

I owe much appreciation to the Charles Stark Draper Laboratory for giving me the opportunity to pursue graduate study at MIT. The technical staff of the Control, Guidance and Navigation Division was always willing to put their world class expertise at my disposal whenever I needed help.

I would also like to thank all my good friends here, especially Rick Niles, Gail Ackermann, Ernie Spevak and Christy Esau whose companionship made these past two years truly wonderful. They will always be dear to me.

Finally, none of this would have been possible without the love and support of my family. They have always been my true source of inspiration and perspective. Thank you Mom and Dad; Sam, Beau, Barry, Max and Ross.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

The emergence of micro-mechanical inertial instruments and the increasing popularity of GPS are enabling the design of very small, low cost guidance systems for a wide range of new applications. Draper Laboratory is currently developing a guidance and control system for a 5 inch artillery shell for the Extended Range Guided Munitions (ERGM) project. This system must be able to fit inside the shell's fuse well, survive an 8,000 G acceleration and navigate accurately in hostile jamming environments. To meet these specifications a GPS with a strapdown, micro-mechanical INS is proposed for the guidance system.

During flight, the artillery shell spin rate will be slowed to approximately 1 Hz. Since the INS is in a strapdown configuration this angular rate will be imposed on the inertial instruments. The errors in the INS caused by angular motion must be calibrated for the system to compute an accurate navigation solution.

The purpose of this thesis is to calibrate errors in a strapdown INS caused by angular motion of the system. Optimal trajectories for a 2-axis gimbal table will be determined that maximize the observability of angular motion errors, producing a high accuracy calibration.

1.2 Guidance and Navigation

In recent years, advances in sensor and computer technologies have made automatic guidance of aerospace vehicles more practical in terms of cost, size and fidelity. A principle task of all guidance systems is navigation. Navigation involves the determination of a physical body's position and velocity relative to some reference frame [3]. There are a wide array of sensor systems available that may be used to obtain a navigation solution.

One category of sensor systems depends on navigation references external to the vehicle to obtain the navigation solution. Examples of these systems include star trackers, magnetic compasses, global positioning system (GPS), Doppler radars and ground based radio beacons. These types of systems are of great value but suffer limitations such as limited range and the possibility of signal loss or jamming.

The other category, known as inertial systems, does not rely on any external references. An inertial navigation system (INS) is a self contained system that utilizes the inertial properties of its sensors to obtain the information needed to navigate. All of the disadvantages of the first category of systems may be eliminated or minimized by a well designed INS [16]. In addition, an INS readily provides information on the vehicle orientation or attitude which is often difficult to obtain using external referencing systems. Inertial systems, however, rely on a numerical integration of force measurements to compute the velocity and a second integration to compute the position of the vehicle. Depending on the quality of the INS, the navigation solution may drift far from true position and velocity.

A complete guidance system may utilize one or a combination of several of the sensor systems described above. Combinations of sensor systems often reduce the disadvantages of using any one system alone. The GPS/inertial sensor combination has received much attention especially for military applications. This combination affords a more accurate navigation solution and computes vehicle orientation using smaller, less expensive inertial instruments than would be required of a guidance system with INS only.

1.3 Inertial Systems

For an inertial navigation system to compute a navigation solution it must perform four functions: instrument a reference frame, measure a specific force, have knowledge of the gravitational field and time integrate the specific force data to compute position and velocity [3]. These tasks are performed using an inertial measurement unit (IMU) consisting of gyroscopes and accelerometers and performing numerical integrations and perhaps coordinate transformations using a digital computer. IMUs are used in two types of inertial navigation systems: gimbaled and strapdown.

In a gimbaled system, the inertial instruments are mounted inside a set of at least three mechanical gimbals that are controlled using feedback from the gyroscopes to create an inertially non-rotating platform for the accelerometers. The accelerometer information is then time integrated by the computer to obtain the vehicle position and velocity.

In a strapdown system, all inertial sensors are fixed to the vehicle body and the navigation computer executes a complex algorithm which uses measurements from the gyroscopes and accelerometers to compute the navigation solution.

Gimbaled systems in general yield a more accurate navigation solution than strapdown systems since there is no angular motion imposed on the inertial sensors. However, because of the necessity for mechanical gimbals, they are larger, more expensive and have higher power consumption than strapdown systems.

1.4 Inertial System Calibration

Before inertial instruments may be used in a navigation system they must be properly calibrated to determine their general input-output behavior and account for errors in the instrument measurements. The number of terms calibrated in an IMU varies with the accuracy required of the navigation system and the fidelity of the gyroscopes and accelerometers.

If inertial systems were perfect they would be able to exactly measure acceleration and angular rate. Real systems however are prone to measurement errors that must be compensated for before a meaningful navigation solution can be computed. An accurate compensation of measurement errors relies on knowledge of the inertial system errors that are obtained through calibration. IMU errors may be classified as static or dynamic. The difference between the two classifications is in the gimbal table motion typically used to calibrate the terms in each class.

1.4.1 Static Errors

Static errors may be calibrated by holding the IMU package steady at particular orientation for a set amount of time before moving to the next orientation. Only the data gathered while the gimbal table is at rest are used in the calibration while data collected during the test table maneuvering are neglected.

Common static error terms calibrated in gyroscopes are bias and g-sensitive terms. A bias manifests itself as a constant gyroscope measurement when the instrument is at rest in inertial space. Once the gyroscope bias is known, it must be subtracted from the gyroscope measurement to obtain the true angular rate sensed by the instrument. Accelerations on a gyroscope may have significant effects on its measurement of angular velocity. These effects are called g-sensitive errors. Gyroscope g-sensitivity is normally calibrated by holding the instrument at different orientations and determining the effect of gravity on its measurement of the earth's rotation.

The major static errors in accelerometers are bias, scale factor and misalignments. Bias is a constant accelerometer output when there is no acceleration imposed on the instrument. Scale factor is the proportionality constant between true acceleration and accelerometer measurement. This term is calculated by positioning the accelerometer to sense gravity and dividing the known gravitational acceleration by the accelerometer measurement. The accelerometers in an IMU are arranged approximately in an orthogonal triad with each sensitive axis aligned with an IMU body axis. Misalignment errors occur when the accelerometer triad is not exactly orthogonal or their axes are not perfectly aligned with the IMU body. When an IMU has accelerometer misalignments, small components of an acceleration along one IMU body axis will be sensed by accelerometers that are approximately orthogonal to that axis.

1.4.2 Dynamic Errors

Dynamic errors are observable when the IMU undergoes angular motion. They are determined by rotating the test table gimbals through a known trajectory and correlating the measurements from the gyros and accelerometers with the test table motion

The most prevalent gyroscope dynamic errors are scale factor and misalignments. Gyroscope scale factor is the proportionality constant between a known angular velocity about the instrument sensitive axis and the gyroscope measurement. Depending on the quality of the gyroscope, there may be scale factor nonlinearities that require the calibration of ω^2 and ω^3 terms to more accurately capture the gyroscope input-output behavior. Misalignment errors are caused when the input axes of the gyroscope triad are not in line with the corresponding IMU body axes. When the IMU is rotated about one of its body axes, components of the angular velocity will be observed by the gyroscopes supposedly orthogonal to that axis.

The major dynamic errors present in accelerometers are lever arms. These errors are classified as a size effect since the major contributing factors are the physical size of the accelerometers and their spatial locations in the IMU. Ideally, when an IMU undergoes pure rotation, no measurement should be yielded by the accelerometers. Lever arm errors are created when the accelerometers sense IMU accelerations at points that are not at the center of the IMU body frame. These errors cause the instruments to sense centripetal accelerations when the IMU is rotated.

1.5 Previous Work

The calibration of inertial systems, because of its difficulty and need for high precision, has been the focus of intense research. Accurate error models for gyroscopes and accelerometers have been derived in [2] and the application of optimal estimation techniques such as weighted least squares and Kalman filtering to estimate the calibration terms has been covered in [4] and [11]. The calibration techniques that have been developed however use test table trajectories that are not optimal.

Optimum test table positions for calibrating strapdown inertial systems have been determined in [19], but these trajectories are only good for static error terms. An attempt at determining optimal test table trajectories for calibrating dynamic errors in accelerometers was made in [15], but the trajectories found, while better than a previously existing trajectory, were not optimal.

The goal of this thesis is to determine optimal test table trajectories for calibrating dynamic errors in inertial systems. Using these trajectories will provide high accuracy calibrations of inertial system errors caused by angular motion of the IMU. This will allow improved navigation of vehicles that spin or encounter high angular rates while maneuvering.

Chapter 2

Model Formulation

2.1 Mathematical Notation

2.1.1 Vectors and Matrices

Vectors are rows or columns of numbers and will generally be denoted as lower case letters. When the vector components are written out, they will be enclosed by floor brackets, $\lfloor \cdot \rfloor$, if a row vector, or braces, $\{\cdot\}$, if a column vector. If a vector quantity is expressed in a particular frame of reference, the reference frame will appear as a superscript. Matrices will be denoted as upper case letters and their components enclosed by square brackets, $[\cdot]$.

Angular Velocity

Angular velocity is a special vector that relates the angular rate occurring between two reference frames. The lower case Greek letter ω is reserved for angular velocity vectors. The complete notation for the angular velocity of frame b with respect to frame i expressed in frame e coordinates is given by ω_{ib}^{e}. Some basic rules governing angular velocity vectors are

$$\omega_{ib}^{e} = -\omega_{bi}^{e} \tag{2.1}$$

and

$$\omega_{ib}^{e} = \omega_{iq}^{e} + \omega_{qb}^{e}. \tag{2.2}$$

Coordinate Transformations

A common operation on vector quantities is changing the frame of reference in which the vector is expressed. The transformation of a vector from frame b to frame e may be accomplished by premultiplying the vector by a direction cosine matrix, denoted C_b^e. Performing this transformation on the vector r^b is given by the equation

$$r^e = C_b^e \, r^b. \tag{2.3}$$

Transformation matrices are orthogonal. A matrix Q is orthogonal if $Q\,Q^T = I$. This useful property permits the reference frames of a direction cosine matrix to be reversed using the formula

$$C_e^b = \left(C_b^e\right)^T. \tag{2.4}$$

Derivatives

Applying Newton's Laws of motion requires time derivatives of vectors expressed in an inertial frame. One method of obtaining an inertial time derivative is to apply the theorem of Coriolis. Let r^b be a vector expressed in the b-frame. The time rate of change of this vector relative to the inertial frame i is

$$\dot{r}^i = C_b^i \left(\dot{r}^b + \omega_{ib}^b \times r^b\right). \tag{2.5}$$

The same result may be obtained using the properties of direction cosine matrices. In this case, the transformation of r^b to the i-frame is given by

$$r^i = C_b^i \, r^b. \tag{2.6}$$

Taking time derivatives of both sides yields

$$\dot{r}^i = \dot{C}_b^i r^b + C_b^i \dot{r}^b. \tag{2.7}$$

The time derivative of the direction cosine matrix C_b^i is given by the formula

$$\dot{C}_b^i = C_b^i \Omega_{ib}^b \tag{2.8}$$

where Ω_{ib}^b is the skew symmetric cross product matrix of the angular velocity vector $\omega_{ib}^b = \lfloor \omega_x \; \omega_y \; \omega_z \rfloor^T$ and is given by

$$\Omega_{ib}^b = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{2.9}$$

The result is

$$\dot{r}^i = C_b^i \left(\dot{r}^b + \Omega_{ib}^b r^b \right) \tag{2.10}$$

which is completely analogous to equation (2.5) with the substitution of the cross product matrix Ω_{ib}^b for $\omega_{ib}^b \times$.

Another type of derivative is the gradient denoted by the ∇ operator. Given a scalar valued function $f(x)$, where x is the vector $\lfloor x_1 \; x_2 \; x_3 \rfloor^T$, the gradient is given by

$$\nabla f(x) = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{Bmatrix}. \tag{2.11}$$

If f is a function of more than one vector, say x and y, the operation $\nabla_y f(x, y)$ is the gradient of f taken with respect to the components of vector y.

The final type of derivative that will be employed is denoted by the operator D.

Given a vector valued function

$$f(x) = \left\{ \begin{array}{c} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{array} \right\} \tag{2.12}$$

where x is an m-vector, the derivative Df is given by

$$Df = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}. \tag{2.13}$$

As with the gradient, subscripts on D may be used if f is a function of more than one vector.

2.1.2 Reference Frames

A number of reference frames are needed to effectively model an IMU. An inertial frame is defined to be fixed in space, although in this thesis, a non-rotating earth centered frame will be considered inertial. The next frame is earth centered earth fixed (ecef) which rotates in the inertial frame at the earth's rotation rate. The local level or navigation frame consists of up, east and north components relative to the IMU location on the earth. Figure 2-1 shows the definitions and relationships between the local level, ecef and inertial frames. The final basic reference frame is the body frame whose components are fixed in the IMU. In addition to the basic reference frames, intermediate frames are used in the model derivation. These frames are defined as they are needed. A summary of the reference frames described above, their components and abbreviations is given in Table 2.1.

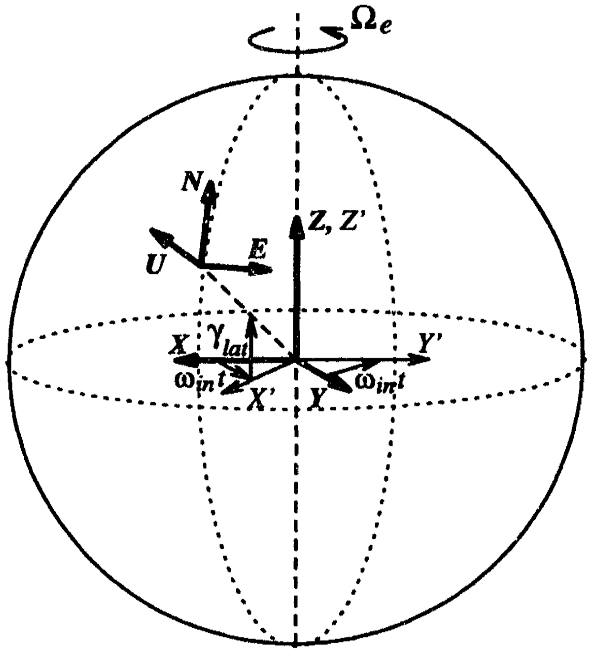


Figure 2-1: Definitions of the local level ($U\,E\,N$) and inertial ($X\,Y\,Z$) reference frames.

Frame	*Components*	*Abbreviation*
body	x, y, z	b
local level	up, east, north ($U\,E\,N$)	n
inertial	X, Y, Z	i
ecef	X', Y', Z'	e
intermediate	x', y', z'	b', q

Table 2.1: Standard reference frames

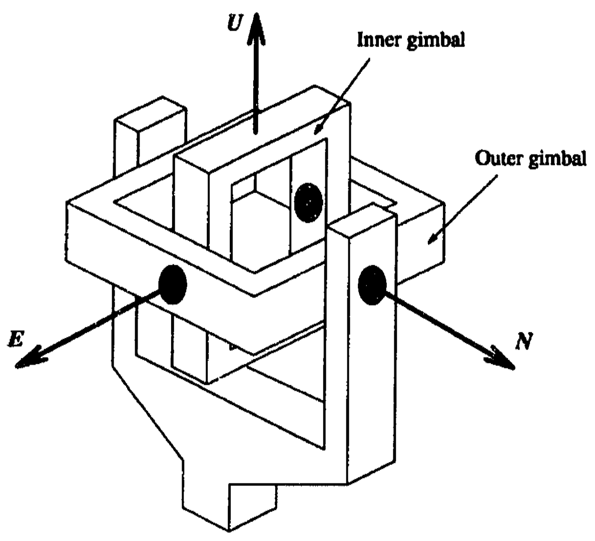


Figure 2-2: Diagram of the two axis test table used in the calibration.

2.2 Gimbal Table Characteristics

A gimbal table or test table is a device frequently used to calibrate inertial instruments. It generally consists of two or three concentric rings, each able to rotate about a different axis at any angular velocity or acceleration under the mechanical constraints of the table. In this thesis, a two axis gimbal table is assumed for the calibration trajectory optimization. A schematic of this test table showing the relationship of the gimbal axes and their orientations in the local level frame is shown in Figure 2-2. When the test table is in the neutral position its outer gimbal has positive rotation about local level north and the inner gimbal positive rotation is about local level east. This sequence of rotations is shown in Figure 2-3. The complete transformation from the local level frame to the body frame may be achieved by an outer gimbal rotation through an angle θ to an intermediate $(x'\,y'\,z')$ frame and then rotating the inner gimbal an angle ϕ about the intermediate y' axis to the body frame. The two direction cosine matrices corresponding to these rotations are

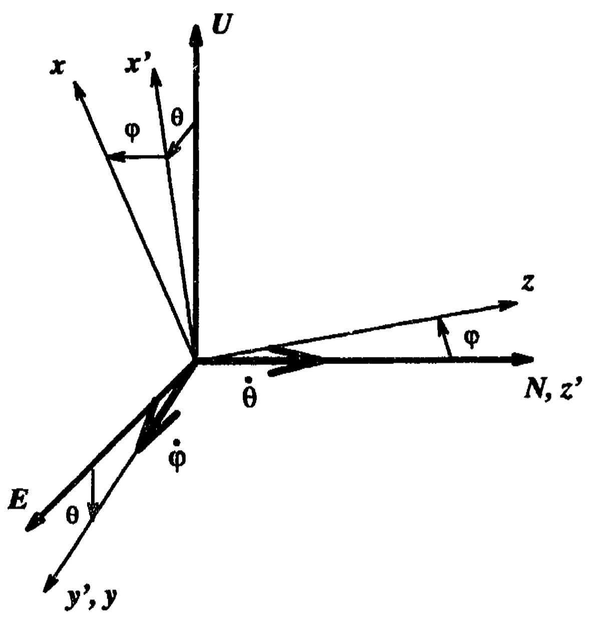


Figure 2-3: Euler angle rotation sequence of two axis gimbal table.

$$C_n^{b'} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

$$C_{b'}^{b} = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix}. \tag{2.15}$$

The complete transformation from local level to body is given by $C_n^b = C_{b'}^b C_n^{b'}$, which results in

$$C_n^b = \begin{bmatrix} \cos\theta\cos\phi & \sin\theta\cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta & 0 \\ \cos\theta\sin\phi & \sin\theta\sin\phi & \cos\phi \end{bmatrix}. \tag{2.16}$$

The angular velocity of the body with respect to local level is simply the sum of the gimbal angle rates. These angular rates, $\dot{\theta}$ and $\dot{\phi}$, are shown in Figure 2-3 to act along the intermediate axes z' and y' respectively. Therefore the expression for angular velocity in body frame coordinates, ω_{nb}^{b}, is determined to be

$$\omega_{nb}^{b} = C_{b'}^{b} \begin{Bmatrix} 0 \\ \dot{\phi} \\ \dot{\theta} \end{Bmatrix}^{b'} \tag{2.17}$$

$$= \left\lfloor -\dot{\theta}\sin\phi,\ \dot{\phi},\ \dot{\theta}\cos\phi \right\rfloor^{T}. \tag{2.18}$$

2.3 Total Angular Velocity

The body frame angular velocity in equation (2.18) accounts for angular motion of the body frame relative to a local level frame. To calculate the complete expression for angular velocity of the body frame with respect to the inertial frame, the angular velocity of local level in inertial space, ω_{in}, must be added to ω_{nb}. Since the test table used in the calibration is mounted on the ground, the local level frame is earth fixed. Therefore ω_{in} is simply earth rate or Ω_e. The angular velocity of the body frame in inertial space expressed in the body frame is

$$\omega_{ib}^{b} = \omega_{in}^{b} + \omega_{nb}^{b} \tag{2.19}$$

$$= C_n^b C_e^n \begin{Bmatrix} 0 \\ 0 \\ \Omega_e \end{Bmatrix}^{e} + \begin{Bmatrix} -\dot{\theta}\sin\phi \\ \dot{\phi} \\ \dot{\theta}\cos\phi \end{Bmatrix}^{b} \tag{2.20}$$

where the direction cosine matrix from ecef to local level depends only on the latitude angle γ_{lat} and is given by

$$C_e^n = \begin{bmatrix} \cos\gamma_{lat} & 0 & \sin\gamma_{lat} \\ 0 & 1 & 0 \\ -\sin\gamma_{lat} & 0 & \cos\gamma_{lat} \end{bmatrix}. \tag{2.21}$$

For completeness, earth rate has been included in the total angular velocity vector. However, when calibrating dynamic errors in inertial systems, the earth's rotation rate of 15.041 $\frac{\text{deg}}{\text{hr}}$ is insignificant compared to the test table gimbal rates. It will therefore be neglected in the actual angular velocity model.

2.4 Inertial Instrument Error Models

A typical IMU consists of three gyroscopes and three accelerometers, each set mounted in an orthogonal triad. These instruments are used to measure the angular velocity and linear acceleration of the IMU platform relative to inertial space. The measurements are then used to compute the attitude, position and velocity of the system. Errors in the measurement introduced by the gyros and accelerometers may be compensated for if the instrument errors are accurately calibrated. To perform the calibration, navigation error models must be constructed which depict the position, velocity and attitude errors produced by the instruments under known translational and rotational motion.

2.4.1 Gyroscope Errors

The navigation attitude error model used for the gyroscopes assumes a ψ-angle formulation [3]. In this formulation an error angle vector, ψ, is defined which consists of the transformation error angles in the direction cosine matrix C_n^b [3]. To derive the ψ-angle differential equations, assume that there are small angle errors in C_n^b. To obtain the correct transformation, an additional direction cosine matrix from the actual body frame b to an erroneous body frame q is needed such that $C_n^q = C_b^q C_n^b$. This matrix is defined to be the small angle direction cosine matrix

$$C_b^q = \begin{bmatrix} 1 & -\psi_z & \psi_y \\ \psi_z & 1 & -\psi_x \\ -\psi_y & \psi_x & 1 \end{bmatrix}. \tag{2.22}$$

Taking the time derivative of C_b^q using the formula in equation (2.8) results in the following matrix differential equation:

$$\begin{bmatrix} 0 & -\dot{\psi}_z & \dot{\psi}_y \\ \dot{\psi}_z & 0 & -\dot{\psi}_x \\ -\dot{\psi}_y & \dot{\psi}_x & 0 \end{bmatrix} = \begin{bmatrix} -\psi_z\omega_z - \psi_y\omega_y & -\omega_z + \psi_y\omega_x & \omega_y + \psi_z\omega_x \\ \omega_z + \psi_x\omega_y & -\psi_z\omega_z - \psi_x\omega_x & -\omega_x + \psi_z\omega_y \\ -\omega_y + \psi_x\omega_z & \omega_x + \psi_y\omega_z & -\psi_y\omega_y - \psi_x\omega_x \end{bmatrix}. \tag{2.23}$$

By enforcing the small angle assumptions that $\psi_x, \psi_y, \psi_z \approx 0$, equation (2.23) may be reduced to

$$\begin{Bmatrix} \dot{\psi}_x \\ \dot{\psi}_y \\ \dot{\psi}_z \end{Bmatrix} = \begin{Bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{Bmatrix} \quad \text{or equivalently} \quad \dot{\psi}^b = \omega_{qb}^b. \tag{2.24}$$

In the case of inertial instrument calibration, the angular velocity vector ω_{qb} corresponds to the difference between the true angular velocity in inertial space and the gyroscope measurement of angular velocity, that is

$$\omega_{qb} = \omega_{ib} - \omega_{iq} \tag{2.25}$$

The angular velocity vector ω_{qb} is referred to as the gyro output error vector and is denoted ϵ_g. Since the inertial instruments are fixed in the body frame, the gyro output error vector is generally expressed in body coordinates.

A final term in the attitude error dynamic model is a white noise process, η_a, which represents a random walk in angle that commonly occurs in gyroscope measurements. The complete ψ-angle dynamic model is then given by the differential equation

$$\dot{\psi}^i = C_b^i \epsilon_g^b + \eta_a. \tag{2.26}$$

For the purposes of navigation, the ψ-angle vector is usually expressed in local level coordinates. To obtain the differential equation governing ψ^n, the following transfor-

	Gyroscope axes		
Body axis	*Gyro 1*	*Gyro 2*	*Gyro 3*
x	input	output	output
y	output	input	spin
z	spin	spin	input

Table 2.2: Gyro axis definition in body frame

mation to the inertial frame is needed:

$$\psi^i = C_n^i \psi^n. \tag{2.27}$$

Differentiating this result with respect to time yields

$$\begin{aligned} \dot{\psi}^i &= C_n^i \dot{\psi}^n + \dot{C}_n^i \psi^n \qquad (2.28) \\ &= C_n^i (\dot{\psi}^n + \Omega_{in}^n \psi^n) \qquad (2.29) \end{aligned}$$

Substituting for $\dot{\psi}^i$ from equation (2.26), and rearranging and transforming the expression from the inertial frame to the local level frame gives

$$\dot{\psi}^n = C_b^n \epsilon_g^b - \Omega_{in}^n \psi^n + \eta_a \tag{2.30}$$

or equivalently

$$\dot{\psi}^n = C_b^n \epsilon_g^b - \omega_{in}^n \times \psi^n + \eta_a. \tag{2.31}$$

Each gyro has orthogonal input, output and spin axes which are used to define directions of sensitivity to error terms. In practice, the gyro triad is bolted inside the inner gimbal in an orientation that facilitates the mount. To maintain simplicity of the model however, it is assumed here that each gyro input, output and spin axis, is aligned with one of the body axes. This axis definition is given in Table 2.2.

The complete gyroscope output error vector including both static and dynamic errors is given by the linear model

$$\epsilon_g^b = B_g + M_g\,\omega_{ib}^b + A_g\,g^b \tag{2.32}$$

where

$$B_g = \lfloor b_{g1}\ b_{g2}\ b_{g3} \rfloor^T \equiv \text{gyro bias coefficients,}$$

$$M_g = \begin{bmatrix} sf_{g1} & mo_{g1} & ms_{g1} \\ mo_{g2} & sf_{g2} & ms_{g2} \\ mo_{g3} & ms_{g3} & sf_{g3} \end{bmatrix} \equiv \text{scale factor and misalignment matrix,}$$

$$A = \begin{bmatrix} adi_{g1} & ado_{g1} & ads_{g1} \\ ado_{g2} & adi_{g2} & ads_{g2} \\ ado_{g3} & ads_{g3} & adi_{g3} \end{bmatrix} \equiv g\text{-sensitive coefficient matrix.}$$

In all, there are 21 gyroscope calibration coefficients considered in the attitude error model.

2.4.2 Accelerometer Errors

The navigation error model for the accelerometers may be built up from the differential equation relating the accelerometer output error vector, ϵ_a, to the position error, δr:

$$\delta \ddot{r}^i = C_b^i \epsilon_a^b + \eta_v \tag{2.33}$$

where ϵ_a is the difference between true acceleration of the inertial instrument package including gravity and the accelerometer measurements. For the purposes of calibration, position and velocity errors are generally expressed in the local level frame so that

$$\delta r^i = C_n^i\,\delta r^n. \tag{2.34}$$

Differentiating with respect to time yields

$$\delta \dot{r}^i = C_n^i \delta \dot{r}^n + \dot{C}_n^i \delta r^n \tag{2.35}$$

$$= C_n^i (\delta \dot{r}^n + \Omega_{in}^n \delta r^n). \tag{2.36}$$

The equation above may be differentiated again and simplified to yield the full expression for acceleration

$$\delta \ddot{r}^i = C_n^i (\delta \ddot{r}^n + 2\Omega_{in}^n \delta \dot{r}^n + \dot{\Omega}_{in}^n \delta r^n + \Omega_{in}^n \Omega_{in}^n \delta r^n). \tag{2.37}$$

In this equation, the angular velocity of local level in the inertial frame is constant, making term $\dot{\Omega}_{in}^n$ equal to zero. In addition, since the calibration is performed by merely spinning the inertial instruments in a gimbal table at a fixed location on the earth, there is no actual translation of the instruments in the local level frame. Therefore the Coriolis term $2\Omega_{in}^n \delta \dot{r}^n$ drops out of the acceleration equation and $\Omega_{in}^n \Omega_{in}^n \delta r^n$ may be absorbed into the local gravity vector. The only term remaining on the right hand side of equation (2.37) is the acceleration error in the local level frame caused by instrument errors. By substituting in the accelerometer output error equation (2.33), the navigation equation may be expressed in the following form:

$$\delta \dot{v}^n = C_b^n \epsilon_a^b + \eta_v \tag{2.38}$$

$$\delta \dot{r}^n = \delta v^n. \tag{2.39}$$

Like the gyroscopes, the accelerometer triad is generally fastened in the inner gimbal in the most convenient orientation for mounting. Each accelerometer has a reference frame consisting of orthogonal input, output and pendulous axes. The definitions for the accelerometer axis directions assumed in this thesis are given in Table 2.3.

The complete accelerometer error model accounts for the basic static and dynamic errors including bias, scale factor, misalignment and lever arm errors. The lever arm of an accelerometer is the position of the accelerometer relative to the center of test

	Accelerometer axes		
Body axis	*Accel 1*	*Accel 2*	*Accel 3*
x	input	output	output
y	output	input	pendulous
z	pendulous	pendulous	input

Table 2.3: Accelerometer axis definition in body frame

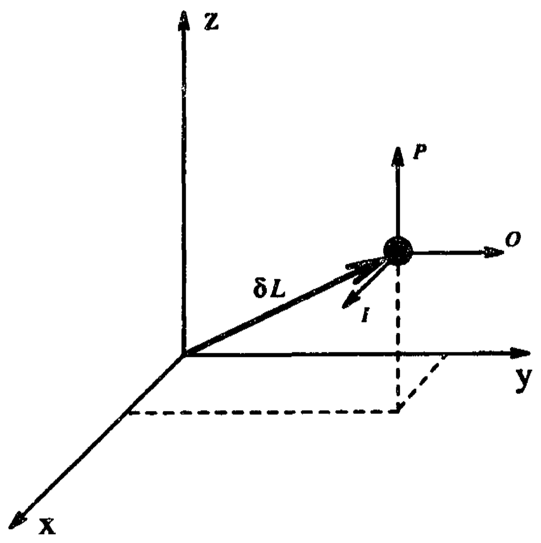


Figure 2-4: Diagram of general accelerometer position in the body frame.

table rotation. A non-zero lever arm then means the accelerometer will sense angular and centripetal accelerations during test table maneuvering. A diagram of a general lever arm vector is given in Figure 2-4. The equation to compute the acceleration of this instrument caused by test table rotation with angular velocity ω_{ib} and acceleration $\dot{\omega}_{ib}$ is

$$a^b = \omega^b_{ib} \times \omega^b_{ib} \times \delta L + \dot{\omega}^b_{ib} \times \delta L. \tag{2.40}$$

An accelerometer only senses accelerations along its input axis. In Figure 2-4 the accelerometer input axis is aligned with the first body frame axis, so only the first component of a^b is observable.

The net accelerometer output error vector in body frame coordinates caused by all error terms is given by the model

$$\epsilon_a^b = B_a + M_a\, g^b + L_a \tag{2.41}$$

where

$$\begin{aligned}
B_a &= \lfloor b_{a1}\ b_{a2}\ b_{a3} \rfloor^T \equiv \text{accelerometer bias coefficients,} \\
M_a &= \begin{bmatrix} sf_{a1} & mo_{a1} & mp_{a1} \\ mo_{a2} & sf_{a2} & mp_{a2} \\ mo_{a3} & mp_{a3} & sf_{a3} \end{bmatrix} \equiv \text{scale factor and misalignment matrix,} \\
L_a &= \left\{ \begin{array}{l} (\omega_{ib}^b \times \omega_{ib}^b \times \delta L_{a1} + \dot{\omega}_{ib}^b \times \delta L_{a1})_x \\ (\omega_{ib}^b \times \omega_{ib}^b \times \delta L_{a2} + \dot{\omega}_{ib}^b \times \delta L_{a2})_y \\ (\omega_{ib}^b \times \omega_{ib}^b \times \delta L_{a3} + \dot{\omega}_{ib}^b \times \delta L_{a3})_z \end{array} \right\} \equiv \text{lever arm error contribution.}
\end{aligned} \tag{2.42}$$

The $(\cdot)_x$ notation in the lever arm contribution vector L denotes the x component of the vector enclosed in parenthesis. There are 21 individual accelerometer parameters to be calibrated. When the accelerometers and gyroscopes are considered together, there are a total of 42 calibration coefficients in the complete system.

2.5 Model Assembly

Now that the attitude and acceleration error models have been derived the complete system model may be assembled. The complete system may be modeled by the following set of linear, time varying differential equations driven by white noise:

$$\dot{x}(t) = F(t)x(t) + w(t) \tag{2.43}$$

$$z(t) = H(t)x(t) + v(t) \tag{2.44}$$

where

$$\begin{aligned}
F(t) &= \text{system dynamics matrix,} \\
H(t) &= \text{measurement geometry matrix,} \\
x(t) &= \text{system state vector,} \\
z(t) &= \text{measurement vector,} \\
w(t) &= \text{process noise vector} \sim N(0, Q(t)), \\
v(t) &= \text{measurement noise vector} \sim N(0, R(t)).
\end{aligned}$$

In this case, the system state is comprised of both static and dynamic states. The dynamic states are the navigation position, velocity and attitude errors while the static states are the gyro and accelerometer error coefficients. The state vector takes the form:

$$x(t) = \lfloor \delta r(t)\ \delta v(t)\ \psi(t)\ c \rfloor^T \tag{2.45}$$

where

$$\begin{aligned}
\delta r(t) &= 3 \times 1 \text{ vector of position errors,} \\
\delta v(t) &= 3 \times 1 \text{ vector of velocity errors,} \\
\psi(t) &= 3 \times 1 \text{ vector of attitude errors,} \\
c &= 42 \times 1 \text{ vector of gyro and accelerometer calibration coefficients.}
\end{aligned}$$

Given this state, the system dynamics matrix has the block form:

$$F(t) = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & 0 & D_c\, \delta\dot{v}^n \\ 0 & 0 & D_\psi\, \dot{\psi}^n & D_c\, \dot{\psi}^n \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.46}$$

After performing the necessary derivatives on $\delta\dot{v}^n$ and $\dot{\psi}^n$ the resulting system matrix, $F(t)$, is found to be linear in the state but dependent on the gimbal angles, angular rates and angular accelerations.

The measurements taken during calibration are the position errors and attitude errors corrupted by white noise. The measurement geometry matrix, $H(t)$, in equation (2.44) is then simply the block form constant matrix:

$$H = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & 0 & I & 0 \end{bmatrix}. \tag{2.47}$$

2.6 Kalman Filter Formulation

When calibrating inertial instruments, a Kalman filter is typically employed to obtain an optimal estimate of the system states, which include the calibration parameters. For the system model given in equations (2.43) and (2.44) the continuous time Kalman filter equations for the state estimate and error covariance propagation are [9];

$$\begin{aligned} \dot{\hat{x}}(t) &= F(t)\hat{x}(t) + P(t)H^T(t)R^{-1}(t)\left[z(t) - H(t)\hat{x}(t)\right], & (2.48) \\ \dot{P}(t) &= F(t)P(t) + P(t)F^T(t) + Q(t) - P(t)H^T(t)R^{-1}(t)H(t)P(t). & (2.49) \end{aligned}$$

In the covariance propagation equation, $Q(t)$ and $R(t)$ are the respective process and measurement noise covariance matrices.

The goal of this thesis is to determine the optimum test table trajectories that maximize the observability of the inertial instrument errors. When these observabilities are maximized, the estimation error variances at the final time, t_f, will be minimized yielding the most accurate estimate of the calibration parameters. The state estimation error variances are the diagonal elements of the covariance matrix $P(t)$. Therefore a reasonable optimization objective is to minimize the cost

$$\text{Cost} = tr\left\{P(t_f)\right\}. \tag{2.50}$$

This minimization may be accomplished by applying an appropriate control to the system which affects the dynamics of the Kalman filter error covariance propagation equation.

In the derivation of the system dynamics matrix, $F(t)$ in equation (2.46) is found to be a function of the gimbal table angles, angular rates and angular accelerations.

By selecting the inner gimbal acceleration, α_ϕ, and the outer gimbal acceleration, α_θ, as the controls, the gimbal angles and angular velocities may be determined by the system of differential equations

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\omega}_\phi \\ \dot{\omega}_\theta \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \phi \\ \theta \\ \omega_\phi \\ \omega_\theta \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} \alpha_\phi \\ \alpha_\theta \end{Bmatrix} \tag{2.51}$$

subject to the angular acceleration and angular velocity constraints

$$|\alpha_\phi(t)| \leq \ddot{\phi}_{max} \tag{2.52}$$

$$|\alpha_\theta(t)| \leq \ddot{\theta}_{max} \tag{2.53}$$

$$|\omega_\phi(t)| \leq \dot{\phi}_{max} \tag{2.54}$$

$$|\omega_\theta(t)| \leq \dot{\theta}_{max} \tag{2.55}$$

where $\ddot{\phi}_{max}$, $\ddot{\theta}_{max}$, $\dot{\phi}_{max}$ and $\dot{\theta}_{max}$ are the maximum accelerations and velocities obtainable by the inner and outer gimbals of the two-axis test table.

The test table dynamics in equation (2.51) may be appended to the error covariance propagation equation (2.49) yielding the complete system of differential equations that describes the behavior of the state estimate error covariances driven by the controls α_ϕ and α_θ.

Chapter 3

Trajectory Optimization

3.1 Optimization by Gradient Methods

In unconstrained parameter optimization problems, the goal is to select a vector of parameters, u, to optimize a criterion $J(u)$. An effective way to solve problems of this type numerically is by invoking a class of algorithms called gradient algorithms. Gradient algorithms start with an initial guess of the parameters in u and then use derivatives of $J(u)$ to produce a new set of parameters which bring $J(u)$ closer to the optimum value. First order gradient algorithms compute a new vector u_{i+1} from u_i using only the gradient of $J(u)$. The first order method known as steepest descent is given by the formula

$$u_{i+1} = u_i - \epsilon_i \nabla J(u_i) \tag{3.1}$$

where the step size ϵ_i is selected according to constraints on the amount of change allowed in the parameter vector at each step. Second order gradient algorithms employ second order derivatives in the optimization. The Newton-Raphson method, a well known second order algorithm, is given by the formula

$$u_{i+1} = u_i - \left[\frac{\partial^2}{\partial u^2} J(u_i)\right]^{-1} \nabla J(u_i). \tag{3.2}$$

Constraints are satisfied by introducing penalty functions into the cost function that add large additional cost if the constraints are not met.

Each method however has advantages and disadvantages. Steepest descent is inherently simple since only the gradient ∇J is computed, but has the disadvantage of slow convergence when the solution nears the optimum. Second order algorithms are generally quicker to converge near the optimum but have the major drawback of requiring the computation of $\left[\frac{\partial^2}{\partial u^2} J(u_i)\right]^{-1}$ [14] [6]. A conjugate gradient method, first proposed by Hestenes and Stiefel in 1952 and refined by Fletcher and Reeves in 1964 [17], improves the convergence method of steepest descent by incorporating information contained in a second order gradient without actually having to compute it as in Newton-Raphson. The result is a powerful and efficient algorithm for solving unconstrained parameter optimization.

The conjugate gradient algorithm starts by establishing an initial guess u_0 and setting $s_0 = -\nabla J(u_0)$. Then for $i = 0, 1, 2, \ldots,$

$$u_{i+1} = u_i + \alpha_i s_i, \tag{3.3}$$

$$\alpha_i = \arg \min_{\alpha \geq 0} J(u_i + \alpha s_i), \tag{3.4}$$

$$s_{i+1} = -\nabla J(u_{i+1}) + \beta_i s_i, \tag{3.5}$$

$$\beta_i = \frac{\nabla J(u_{i+1}) \cdot \nabla J(u_{i+1})}{\nabla J(u_i) \cdot \nabla J(u_i)} \tag{3.6}$$

Although this method requires only function and gradient evaluations, it has been shown in [14] to converge much faster than the method of steepest descent with convergence stability properties superior to second-order Newton's methods.

3.2 Conjugate Gradient Algorithm for Trajectory Optimization

The general formulation of an optimal control problem is to minimize a cost function of the form

$$J = c(x(t_f)) + \int_0^{t_f} d(x(t), u(t))\, dt \tag{3.7}$$

subject to the system dynamics

$$\dot{x}(t) = f(x(t), u(t)) \tag{3.8}$$

with the given initial state $x(0)$. In these equations, x is an n vector, u is an m vector and t_f is the fixed final time. The algorithm may be simplified by defining a scalar valued Hamiltonian function:

$$H(x, u, \lambda) = d(x, u) + \sum_{i=1}^{n} \lambda_i f_i\,. \tag{3.9}$$

Let $\lambda(t)$ be the solution to the adjoint equation

$$\dot{\lambda}(t) = -\nabla_x H(x, u, \lambda) \tag{3.10}$$

with the boundary condition

$$\lambda(t_f) = \nabla c(x(t_f)). \tag{3.11}$$

The gradient is then

$$g(u) = \nabla_u H(x, u, \lambda). \tag{3.12}$$

Define $u_i(t)$ to be the i^{th} approximation of the optimal control. The algorithm is started by selecting an arbitrary initial trajectory u_0, integrating the state equations (3.8) and storing the state trajectories. Next, the adjoint equations (3.10) are integrated backwards in time and the adjoint trajectories are stored. The gradient trajectory g_0 is then computed and the initial direction vector is defined to be $s_0 = -g_0$. Once initialized, the following algorithm is repeated until the solution $u(t)$ converges to an optimum value:

$$\alpha_i = \arg\min_{\alpha \geq 0} J(u_i + \alpha s_i) \tag{3.13}$$

$$u_{i+1} = u_i + \alpha_i s_i \tag{3.14}$$

$$g_{i+1} = g(u_{i+1}) \tag{3.15}$$

$$\beta_i = \frac{\langle g_{i+1}, g_{i+1} \rangle}{\langle g_i, g_i \rangle} \tag{3.16}$$

$$s_{i+1} = -g_{i+1} + \beta_i s_i \tag{3.17}$$

where

$$\langle g_i, g_j \rangle = \int_0^{t_f} g_i(t)\, g_j(t)\, dt. \tag{3.18}$$

The major difficulty with this method is the minimization in equation (3.13). In practice an exact analytical minimization is generally not possible, and the evaluation of $J(u_i + \alpha s_i)$ is typically computationally prohibitive [14] [17]. Instead the parameter α_i is normally chosen to either simply reduce the cost at each stage or be the minimum in a polynomial interpolation of the cost function. Using this step adjustment logic, however, may lead to poor conjugate directions and sharply decrease the rate of convergence [10].

Another limitation of the conjugate gradient method is that, like all gradient methods, it is only able to solve unconstrained optimization problems. However, gradient algorithms may be used to obtain approximations to optimal solutions by the addition of integral penalty functions to the cost function [6]. A good discussion of commonly used penalty functions may be found in [8].

Finally, gradient methods are only able to determine local maxima and minima in a cost function. This problem may be alleviated by executing the algorithm using several initial control trajectories. Although there are no criteria to determine whether the global minimum or maximum has been located, several of the local extrema will be found and one of these trajectories is likely to be better than the rest.

Chapter 4

Implementation

4.1 Dynamic Error Models

Two types of inertial systems are considered. One is a ring laser gyro IMU package, the other is a micro-mechanical IMU. Aside from a large difference in the physical size of the IMU packages, each system has a slightly different error model and a different random walk on the navigation solution. These differences will result in different optimal trajectories for calibrating each system.

In Chapter 2, *Model Formulation*, a general navigator system dynamics model was derived to be

$$\begin{Bmatrix} \delta\dot{r}(t) \\ \delta\dot{v}(t) \\ \dot{\psi}(t) \\ \dot{c} \end{Bmatrix} = \begin{bmatrix} 0 & I & 0 & 0 \\ 0 & 0 & 0 & D_c\,\delta\dot{v}^n \\ 0 & 0 & D_\psi\,\dot{\psi}^n & D_c\,\dot{\psi}^n \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \delta r(t) \\ \delta v(t) \\ \psi(t) \\ c \end{Bmatrix} + \begin{Bmatrix} 0 \\ \eta_v \\ \eta_a \\ 0 \end{Bmatrix}, \tag{4.1}$$

where

$$\dot{\psi}^n = C_b^n \epsilon_g^b - \omega_{in}^n \times \psi^n \tag{4.2}$$

$$\delta\dot{v}^n = C_b^n \epsilon_a^b. \tag{4.3}$$

The terms ϵ_g^b and ϵ_a^b are the gyroscope and accelerometer output error vectors. Originally, these error vectors were derived to include both static and dynamic instrument errors. This thesis however is only concerned with calibrating dynamic errors, so only these errors will be included in the navigator model.

The state vector in the navigator system model is comprised of four separate vectors. The first three, δr, δv and ψ represent the navigator position, velocity and attitude errors respectively and each has three components. The vector c contains the dynamic error terms in the system model. The number of components in c varies with the inertial system that is modeled and its desired accuracy.

4.1.1 Ring Laser Gyro IMU

A ring laser gyro IMU consists of three accelerometers and three ring laser gyroscopes. Ring laser gyros are widely used in strapdown navigation systems since they have excellent scale factor stability over a wide dynamic range of angular rates [18]. The error models for these instruments then typically approximate gyroscope scale factor as linear with angular velocity about the input axis. Since only dynamic errors are considered, the accelerometer and gyroscope output error models are

$$\epsilon_a^b = \left\{ \begin{array}{c} (\omega_{ib}^b \times \omega_{ib}^b \times \delta L_{a1} + \dot{\omega}_{ib}^b \times \delta L_{a1})_x \\ (\omega_{ib}^b \times \omega_{ib}^b \times \delta L_{a2} + \dot{\omega}_{ib}^b \times \delta L_{a2})_y \\ (\omega_{ib}^b \times \omega_{ib}^b \times \delta L_{a3} + \dot{\omega}_{ib}^b \times \delta L_{a3})_z \end{array} \right\} \tag{4.4}$$

and

$$\epsilon_g^b = \begin{bmatrix} sf_{g1} & mo_{g1} & ms_{g1} \\ mo_{g2} & sf_{g2} & ms_{g2} \\ mo_{g3} & ms_{g3} & sf_{g3} \end{bmatrix} \left\{ \begin{array}{c} \omega_1 \\ \omega_2 \\ \omega_3 \end{array} \right\} \tag{4.5}$$

where ω_1, ω_2 and ω_3 are components of the angular velocity vector ω_{ib}^b. Included in these models are 3 lever arm vectors each with 3 components and a 9 component misalignment matrix for a total of 18 dynamic error terms. Therefore the complete navigator model for the ring laser gyro IMU navigator has 27 states.

4.1.2 Micro-Mechanical IMU

New manufacturing technologies has enabled the development of micro-mechanical inertial instruments that offer a significant reduction in size from the ring laser gyro systems. However, poor measurements given by micro-mechanical instruments make them inappropriate for most navigation system applications. The accelerometer dynamic error model for the micro-mechanical IMU is the same as for the ring laser gyro IMU, but the poor gyroscope scale factor stability requires the addition of the nonlinear scale factor terms, ω^2 and ω^3, in the gyroscope error model. The micro-mechanical gyroscope dynamic error model is:

$$\epsilon_g^b = \begin{bmatrix} sf_{g1} & mo_{g1} & ms_{g1} \\ mo_{g2} & sf_{g2} & ms_{g2} \\ mo_{g3} & ms_{g3} & sf_{g3} \end{bmatrix} \begin{Bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{Bmatrix} + \begin{bmatrix} sf2_{g1} & 0 & 0 \\ 0 & sf2_{g2} & 0 \\ 0 & 0 & sf2_{g3} \end{bmatrix} \begin{Bmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \end{Bmatrix} \tag{4.6}$$

$$+ \begin{bmatrix} sf3_{g1} & 0 & 0 \\ 0 & sf3_{g2} & 0 \\ 0 & 0 & sf3_{g3} \end{bmatrix} \begin{Bmatrix} {\omega_1}^3 \\ {\omega_2}^3 \\ {\omega_3}^3 \end{Bmatrix} \tag{4.7}$$

where ω_1, ω_2 and ω_3 are components of the angular velocity vector ω_{ib}^b. The gyroscope error model for this case contains an additional 6 error terms to account for the scale factor nonlinearity. As a result, the micro-mechanical IMU navigator model has a total of 33 states.

4.2 Computer Software

The actual problem in this thesis is to determine a test table trajectory that minimizes the estimation uncertainty of the final navigator state. The navigator differential equations in (4.1) are linear in the state and driven by Gaussian white noise. A Kalman filter will then provide the optimal estimate of the navigator state and the estimation error covariance matrix. Therefore the differential equations that govern the trajectory optimization are the Kalman filter covariance propagation equation (2.49) and the test table dynamics (2.51). These differential equations are repeated below for clarity:

$$\dot{P}(t) = F(t)P(t) + P(t)F^T(t) + Q(t) - P(t)H^T(t)R^{-1}(t)H(t)P(t)$$

$$\begin{Bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\omega}_\phi \\ \dot{\omega}_\theta \end{Bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} \phi \\ \theta \\ \omega_\phi \\ \omega_\theta \end{Bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} \alpha_\phi \\ \alpha_\theta \end{Bmatrix}.$$

The covariance matrix $P(t)$ is a symmetric, $n \times n$ matrix where n is the number of states in the navigator model [9]. In the case of the micro-mechanical IMU with a 33 state navigator model, $P(t)$ is comprised of 1089 first order differential equations. By exploiting the symmetry of a covariance matrix, only the upper triangular components of the Kalman filter covariance propagation equation must be integrated. This reduces the number of differential equations in an $n \times n$ system to $\frac{n(n+1)}{2}$.

The derivation of the equations required for the trajectory optimization problem and the numerical optimization calculations are extremely complex and tedious tasks. Software such as Mathematica and the C programming language are used extensively so that a high speed digital computer could be employed to perform these tasks.

4.2.1 Model Derivation Software

The large number of navigator states and the complexity of the error models made it unreasonable to obtain the complete expression of the covariance propagation differential equation by hand. The program Mathematica is designed to perform symbolic mathematical calculations and is well suited to derive all the equations needed in the optimization problem starting from basic principles. Mathematica script files are developed to derive the Kalman filter covariance propagation differential equations with the appended gimbal table dynamics and compute analytical expressions for the gradient and adjoint equations required for the trajectory optimization. The script file written for the micro-mechanical system is included in Appendix A. The Mathematica script for the ring laser gyro IMU system is nearly identical except for a simpler gyroscope error model.

4.2.2 Optimization Software

A general unconstrained trajectory optimization software package is developed to execute the conjugate gradient algorithm presented in Chapter 3. The software is written entirely in the C programming language and it is designed to directly implement the system model equation output files from the Mathematica model derivation script. A complete user's manual for the software is included in Appendix B.

Validation

The trajectory optimization software is validated against a simple, linear quadratic, optimal regulator problem. A problem of this type has a well known exact solution [13]. The general problem statement is to minimize the cost

$$J = x^T(t_f)\, S\, x(t_f) + \int_0^{t_f} x^T(t)\, Q(t)\, x(t) + u^T(t)\, R(t)\, u(t)\; dt, \tag{4.8}$$

subject to the system dynamics

$$\dot{x}(t) = A(t)\, x(t) + B(t)\, u(t) \tag{4.9}$$

with the initial condition

$$x(0) = x_0. \tag{4.10}$$

The optimal control, u(t), that minimizes the cost J is then given by

$$u(t) = -F(t)\, x(t) \tag{4.11}$$

where

$$F(t) = R^{-1}(t)\, B^T(t)\, P(t) \tag{4.12}$$

and P(t) is the symmetric, positive semidefinite matrix which satisfies the matrix Riccati equation

$$-\dot{P}(t) = P(t)\,A(t) + A^T(t)\,P(t) - P(t)\,B(t)\,R^{-1}(t)\,B^T(t)\,P(t) + Q(t), \qquad (4.13)$$

with the terminal condition

$$P(t_f) = S. \qquad (4.14)$$

The example system used in the validation is second order, single input and time invariant with the following system, initial condition and cost matrices:

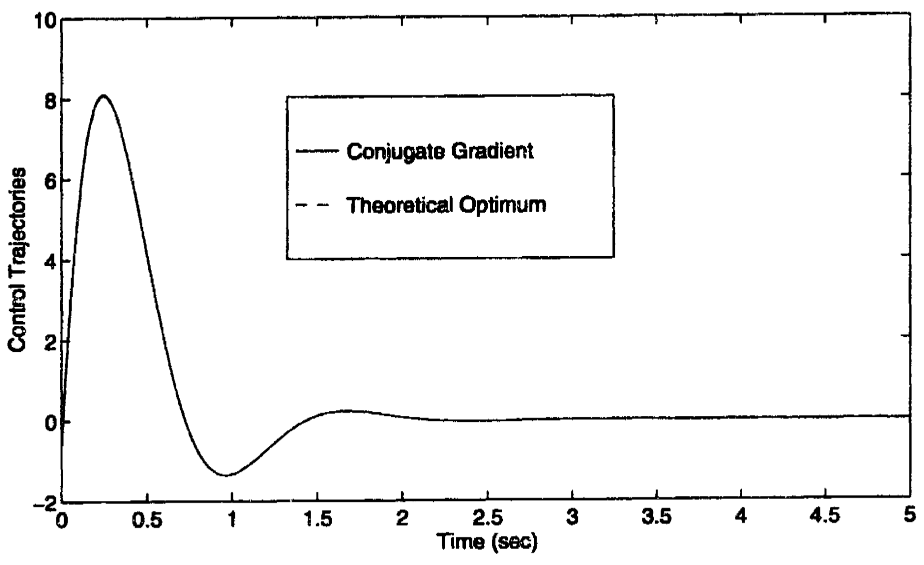
$$A = \begin{bmatrix} 0 & 1 \\ -25 & -2 \end{bmatrix}, \quad B = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \quad x_0 = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix} \qquad (4.15)$$

$$Q = \begin{bmatrix} 3 & \frac{1}{2} \\ \frac{1}{2} & 2 \end{bmatrix}, \quad R = \frac{1}{10}, \quad S = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad (4.16)$$

The exact solution for the optimal control is computed and compared to the optimal control obtained by the conjugate gradient trajectory optimization. In both cases, a forward Euler numerical integration with a ΔT of 0.001 seconds is used. Figure 4-1 shows the plots of the control trajectories computed using both methods and the difference between them.

4.3 Numerical Issues

At this point, the trajectory optimization problem is defined and the mathematical models of the system are derived. There are several issues remaining that involve the Kalman filter initialization, incorporation of the test table constraints and the accuracy and storage limitations involved with the optimal trajectory computation. These issues must be considered before a solution may be computed.



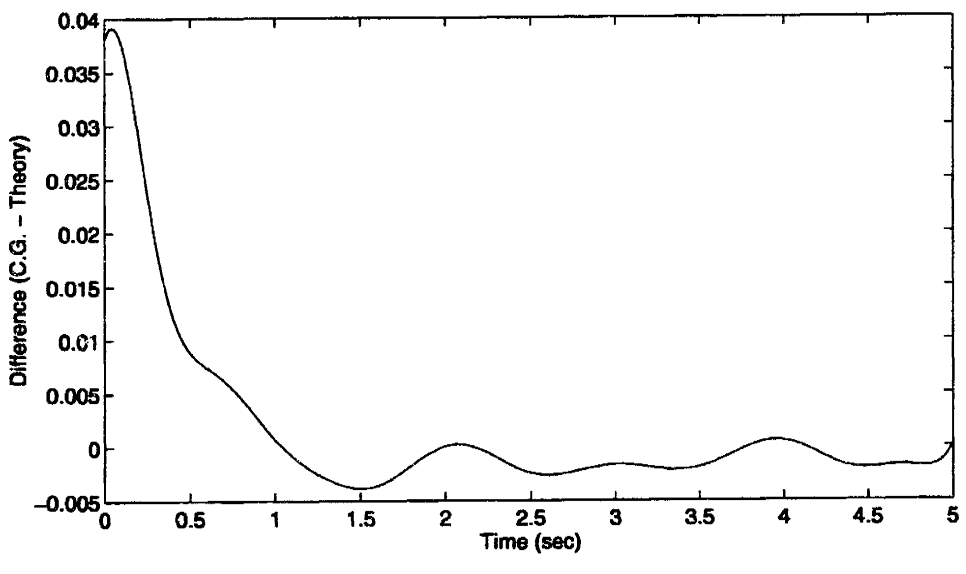


Figure 4-1: Validation of optimal control achieved by conjugate gradient trajectory optimization.

State	*Variance*	*Units*
Position error	40	mm^2
Velocity error	300	mm^2/s^2
Attitude error	2400	μrad^2
Lever Arm	2000	mm^2
Scale Factor	1×10^6	μrad^2
Misalignment	324×10^6	μrad^2

Table 4.1: Initial covariance matrix diagonal elements

4.3.1 Kalman Filter

Propagating the Kalman filter covariance matrix using equation (2.49) requires the system model matrices $F(t)$ and $H(t)$, measurement and process noise covariance matrices $Q(t)$ and $R(t)$, and an initial state estimate error covariance matrix $P(0)$.

The initial covariance matrix is assumed to be a diagonal matrix with the initial state uncertainties as the diagonal elements. The initial state estimation error variances are listed in Table 4.1. This set of numbers is used to initialize the covariance matrix for both the ring laser gyro and the micro-mechanical IMUs.

The general navigator system model, derived in Chapter 2, assumed the form

$$\begin{aligned} \dot{x}(t) &= F(t)x(t) + w(t) && (4.17) \\ z(t) &= H(t)x(t) + v(t). && (4.18) \end{aligned}$$

In this system, $w(t)$ and $v(t)$ are stationary, zero mean, uncorrelated, Gaussian white noise random processes. They represent the process and measurement noise vectors respectively with covariance matrices defined as

$$\begin{aligned} Q(t) &= E\left\{w(t)\, w^T(t)\right\} && (4.19) \\ R(t) &= E\left\{v(t)\, v^T(t)\right\}. && (4.20) \end{aligned}$$

The only navigator states subject to process noise are the velocity errors, δv, and attitude errors, ψ. Process noise must be added to these states to properly model the

Random walk	*Laser IMU*	*Micro IMU*	*Units*
Accelerometer	0.006	0.065	$\text{ft/s}/\sqrt{\text{hr}}$
Gyroscope	0.006	1.4	$\text{deg}/\sqrt{\text{hr}}$

Table 4.2: Random walk behavior of ring laser gyro and micro-mechanical IMUs.

Measurement	*Noise 1-σ*	*Units*
Position error	0.01	ft
Attitude error	1.0	arc sec

Table 4.3: Measurement noise standard deviations.

random walk in the accelerometer and gyroscope readings respectively. The position errors, δr, are the time integrations of the velocity errors, and the calibration error term states are constants, so there is no process noise on these states. The process noise variances are determined by squaring the random walk standard deviation of the inertial instruments. Typical random walk characteristics of ring laser gyro and micro-mechanical IMUs are shown in Table 4.2.

All of the measurements are affected by measurement noise. The measurement noise is caused by the test table itself and its associated data collection devices so the measurement noise covariance matrix, $R(t)$ is the same for both IMU types. Typical measurement noise standard deviations for discrete measurements are given in Table 4.3.

4.3.2 Test Table Constraints

The conjugate gradient method solves unconstrained optimization problems. The calibration problem, however, has inequality constraints imposed by the test table's mechanical limitations. These constraints are dealt with indirectly by adding cost when the states or controls describing the test table motion near a constraint boundary. The gimbals of the two axis test table described in this thesis are subject to the following maximum angular velocities and accelerations:

$$\dot{\theta}_{max} = \dot{\phi}_{max} \quad = \quad 180 \text{ deg/s} \tag{4.21}$$

$$\ddot{\theta}_{max} = \ddot{\phi}_{max} \quad = \quad 270 \text{ deg/s}^2. \tag{4.22}$$

The overall objective of the trajectory optimization is to minimize the trace of the final state estimate error covariance matrix. To account for the test table constraints, an integral penalty function term is added to the cost function. With this additional term, the cost function takes the form

$$\text{Cost} = tr\left\{P(t_f)\right\} + \int_0^{t_f} f(\omega_\phi, \omega_\theta, \alpha_\phi, \alpha_\theta)\ dt. \tag{4.23}$$

Several types of integral penalty functions are valid for inequality constraints. Two of the most common types are those with polynomial or logarithmic integrands [8]. For an integrand $f(x)$ and the inequality constraints

$$|x_i| \leq \text{max}_i\ , \quad i = 1, \ldots, n \tag{4.24}$$

the general forms for these penalty functions are:

$$\text{polynomial}: \quad f(x) \quad = \quad \sum_{i=1}^{n} c_i \left(\frac{x_i}{\text{max}_i}\right)^{m_i} \tag{4.25}$$

$$\text{logarithmic}: \quad f(x) \quad = \quad \sum_{i=1}^{n} c_i \left[-\log(\text{max}_i + x_i) - \log(\text{max}_i - x_i)\right] \tag{4.26}$$

where c_i are positive scaling constants and m_i are positive, even integers. Both types of penalty functions may be tuned to have similar behavior. To compare their behavior, a 20^{th} order polynomial penalty function and a logarithmic penalty function scaled by 0.1 are applied to a hypothetical state constrained to be less than 100. The shapes of these functions are plotted in Figure 4-2. For the purpose of this thesis, a 20^{th} order polynomial integral penalty function is deemed adequate to satisfy the test table inequality constraints. When this penalty function is incorporated, the

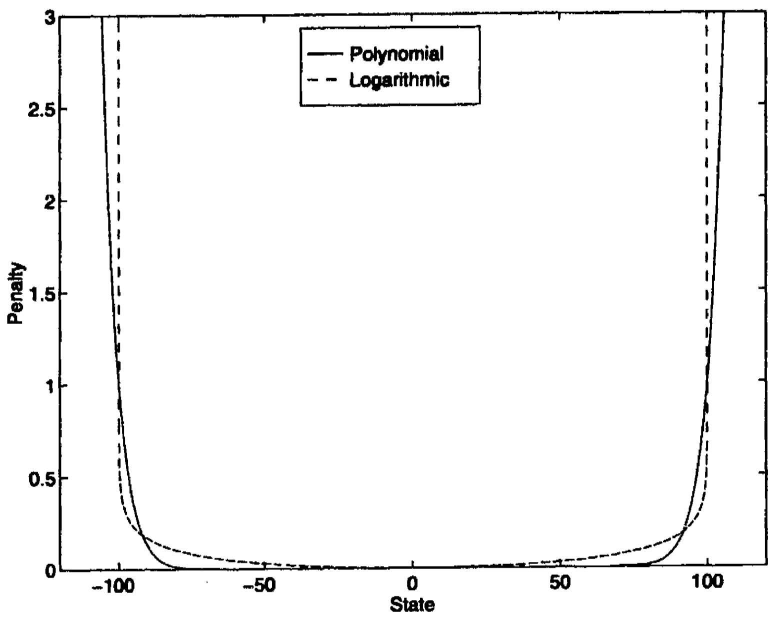


Figure 4-2: Comparison of polynomial and logarithmic penalty functions.

complete cost function becomes

$$\text{Cost} = tr\left\{P(t_f)\right\} + \int_0^{t_f} \left[\left(\frac{\omega_\theta}{\dot{\theta}_{max}}\right)^{20} + \left(\frac{\omega_\phi}{\dot{\phi}_{max}}\right)^{20} + \left(\frac{\alpha_\theta}{\ddot{\theta}_{max}}\right)^{20} + \left(\frac{\alpha_\phi}{\ddot{\phi}_{max}}\right)^{20} \right] dt. \tag{4.27}$$

This cost function is used for both the ring laser gyro and micro-mechanical IMUs.

4.3.3 Computational Limitations

The complexity of this optimization problem and the available computer power hampers the computation of the optimal trajectories. Various computational limitations encountered when computing a solution lead to trade-offs between the trajectory length, level of accuracy and computation time. The two dominant factors that contribute to these computational difficulties are the numerical integrators and the memory requirements.

Numerical Integration Methods

The trajectory optimization software permits the choice of two numerical integration methods: forward Euler and 4th order Runge-Kutta. These methods solve the basic system of first order differential equations:

$$\begin{aligned}
\dot{x}_1(t) &= f_1(x_1(t), \ldots, x_m(t), t) && (4.28)\\
\dot{x}_2(t) &= f_2(x_1(t), \ldots, x_m(t), t) && (4.29)\\
\vdots \quad & \vdots \qquad \vdots && (4.30)\\
\dot{x}_m(t) &= f_m(x_1(t), \ldots, x_m(t), t). && (4.31)
\end{aligned}$$

The forward Euler method solves the system of differential equations by applying the formula:

$$x_i^{n+1} = x_i^n + \Delta T \cdot f_i(x_1^n, \ldots, x_m^n, t^n) \qquad \begin{cases} n = 0, 1, 2, \ldots \\ i = 1, \ldots, m \end{cases} \tag{4.32}$$

where ΔT is the integration time step. This is the simplest and fastest way to solve the first order differential equations in the system but is subject to inaccuracy and instability if ΔT is chosen to be too large or when integrating stiff systems [7].

The 4th order Runge-Kutta method attempts to improve the accuracy and stability of the forward Euler method without the need to compute higher order derivatives. It accomplishes this goal by evaluating $f(x, t)$ at intermediate points on each subinterval. Differential equations are solved using this method by applying the formula:

$$x_i^{n+1} = x_i^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad \begin{cases} n = 0, 1, 2, \ldots \\ i = 1, \ldots, m \end{cases} \tag{4.33}$$

where

$$\begin{aligned}
k_1 &= \Delta T \cdot f_i\left(x_i^n, \ldots, x_m^n,\ t^n\right) && (4.34)\\
k_2 &= \Delta T \cdot f_i\left(x_i^n + \frac{1}{2}k_1, \ldots, x_m^n + \frac{1}{2}k_1,\ t^n + \frac{\Delta T}{2}\right) && (4.35)
\end{aligned}$$

$$k_3 = \Delta T \cdot f_i\left(x_i^n + \frac{1}{2}k_2, \ldots, x_m^n + \frac{1}{2}k_2,\ t^n + \frac{\Delta T}{2}\right) \tag{4.36}$$

$$k_4 = \Delta T \cdot f_i\left(x_i^n + k_3, \ldots, x_m^n + k_3,\ t^n + \Delta T\right) \tag{4.37}$$

In the case of trajectory optimization, the function $f(x,t)$ may not be an explicit function of time but is always implicitly dependent on time through the time varying control trajectory that is being optimized. The control trajectory data is available at each whole time step but not at the half time steps required by the Runge-Kutta method. An approximation of the half time step control data is obtained by performing a linear interpolation on the surrounding data. The interpolation reduces the accuracy of the method but in practice it has been observed to still be both more stable and accurate than forward Euler.

The major advantage the forward Euler integration method has over the 4th order Runge-Kutta is in computation time. Computation of a 2 minute optimal trajectory using forward Euler integration with a ΔT of 0.005 seconds takes approximately 10 hours on a Sun Sparc 20 workstation. Since the 4th order Runge-Kutta requires 4 times as many function evaluations to integrate a differential equation, a solution using this method takes approximately 30 to 40 hours using the same integration time step. However, the increase in solution accuracy and stability afforded by Runge-Kutta integration made it the choice for the trajectory optimizations, while forward Euler was used only to gain preliminary results.

Memory Requirements

Performing a trajectory optimization using the conjugate gradient method generally requires the storage of the entire state, adjoint, gradient, control and conjugate direction trajectories. Depending on the optimization problem, it may not be necessary to save the trajectories for all of the states, adjoints and gradients. For the problem in this thesis, however, storage of all trajectories is required.

The states in the calibration problem are defined to be the upper triangular components of the Kalman filter covariance matrix plus the 4 test table states. This amounts to 382 states for the ring laser gyro IMU and 565 states for the micro-mechanical IMU. The conjugate gradient method also requires as many adjoints as states, and, since a 2 axis test table is being used, there are 2 control, gradient and conjugate direction trajectories. In all, a total of 770 trajectories must be stored for the ring laser gyro IMU case and 1136 for the micro-mechanical IMU case. The tremendous memory demands that these storage requirements impose on the computer limit the trajectory lengths to 120 seconds when a 0.005 second integration time step is used.

Chapter 5

Results

Optimal test table trajectories for the ring laser gyro IMU and the micro-mechanical IMU are generated. The calibration term estimation errors obtained when using the optimal trajectories are then compared to a benchmark calibration which uses a traditional test table trajectory. All of the test table trajectories evaluated are 120 seconds long.

The benchmark calibration trajectory for each IMU is comprised of two types of maneuvers. The first type is a scale factor maneuver in which the test table spins about each IMU body axis. This maneuver maximizes the observability of the gyroscope scale factors, misalignments and 3 of the 9 accelerometer lever arm components. The second type is a lever arm maneuver. In this maneuver, the test table gimbals undergo arbitrarily chosen constant angular accelerations in an attempt to afford observability to the remaining lever arm components rendered unobservable by the scale factor maneuvers.

Numerical problems are encountered during integration of the Kalman filter covariance matrix differential equation when large gimbal accelerations are commanded or some of the error terms become too observable. These factors lead to an ill-conditioned covariance matrix, a problem common in Kalman filtering particularly if the measurements are very accurate [1]. When performing the benchmark calibrations, covariance matrix condition problems are overcome by restricting the gimbal accelerations to be small at all times and not allowing high gimbal velocities during

Benchmark Trajectory: State Estimate 1-σ				
State	*Axis 1*	*Axis 2*	*Axis 3*	*Units*
Scale Factor	1.437	1.061	1.447	ppm
Misalignment 1	1.196	1.586	1.610	μrad
Misalignment 2	1.636	1.607	1.198	μrad
Lever Arm 1	0.2781	0.5830	1.791	mm
Lever Arm 2	5.190	0.3052	7.903	mm
Lever Arm 3	2.562	0.5159	0.2829	mm

Table 5.1: Laser IMU benchmark trajectory calibration accuracy.

the scale factor maneuvers. This is accomplished by processing each maneuver in the benchmark trajectory separately and initializing each new maneuver with the final covariance matrix from the previous maneuver.

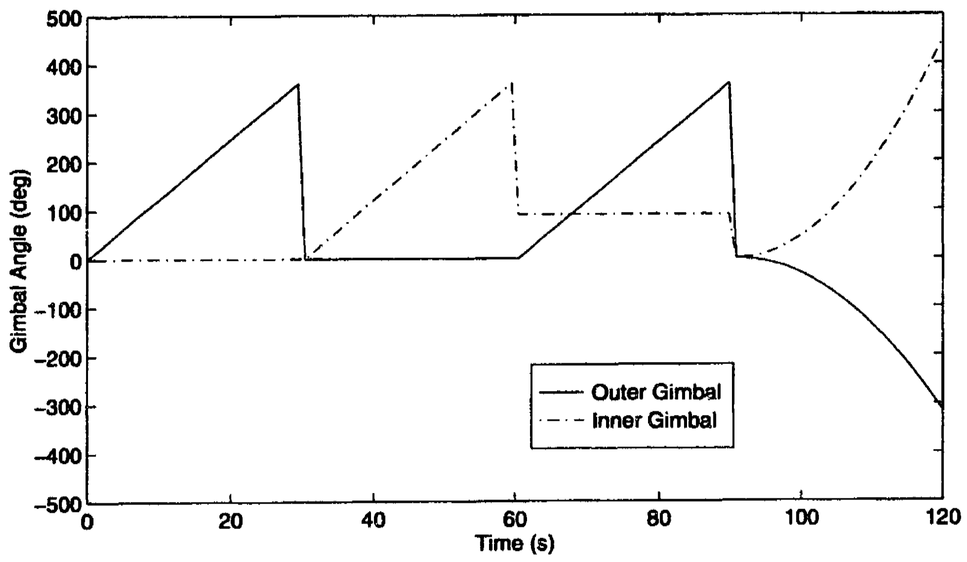
5.1 Ring Laser Gyro IMU

5.1.1 Benchmark Trajectory

The benchmark test table trajectory generated to calibrate the ring laser gyro IMU consists of 3, 30 second scale factor maneuvers and a final 30 second lever arm maneuver. Since a linear scale factor model is assumed for ring laser gyroscopes, only a single angular velocity is needed in each scale factor calibration maneuver. The gimbal angles and angular velocities for this test table trajectory are plotted in Figure 5-1.

The angular velocities selected for this trajectory are governed by the numerical stability of the covariance propagation equation. Numerical problems caused by high scale factor observability occur each time the benchmark trajectory isolates a scale factor term by spinning about a gyroscope input axis. Although faster spin rates during the scale factor maneuvers would result in a better calibration, they cannot be handled by the Kalman filter.

The final calibration uncertainties are listed in Table 5.1. These numbers represent the calibration error standard deviations produced by the benchmark trajectory. They are computed by taking the square roots of the diagonal elements in the final Kalman



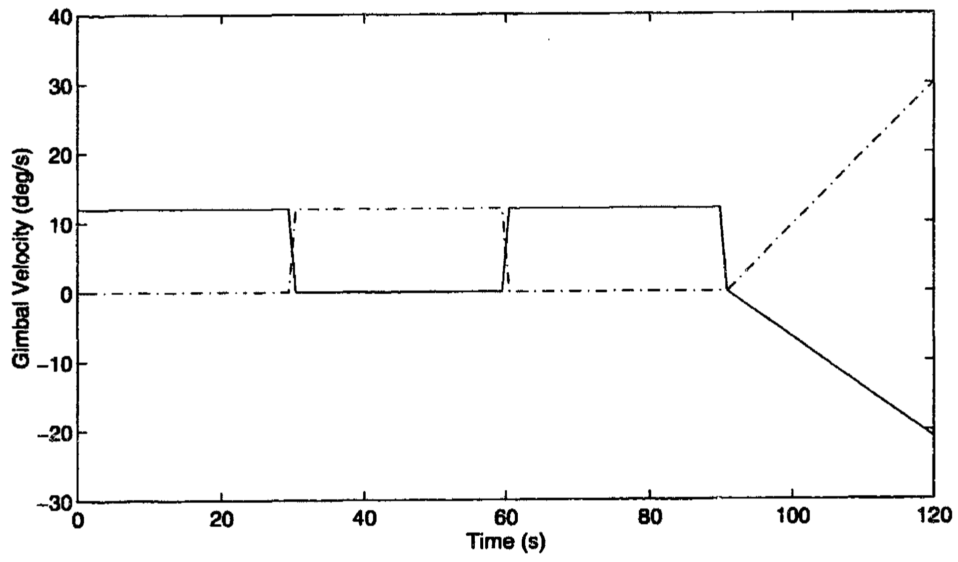


Figure 5-1: Laser IMU benchmark test table gimbal angles and velocities.

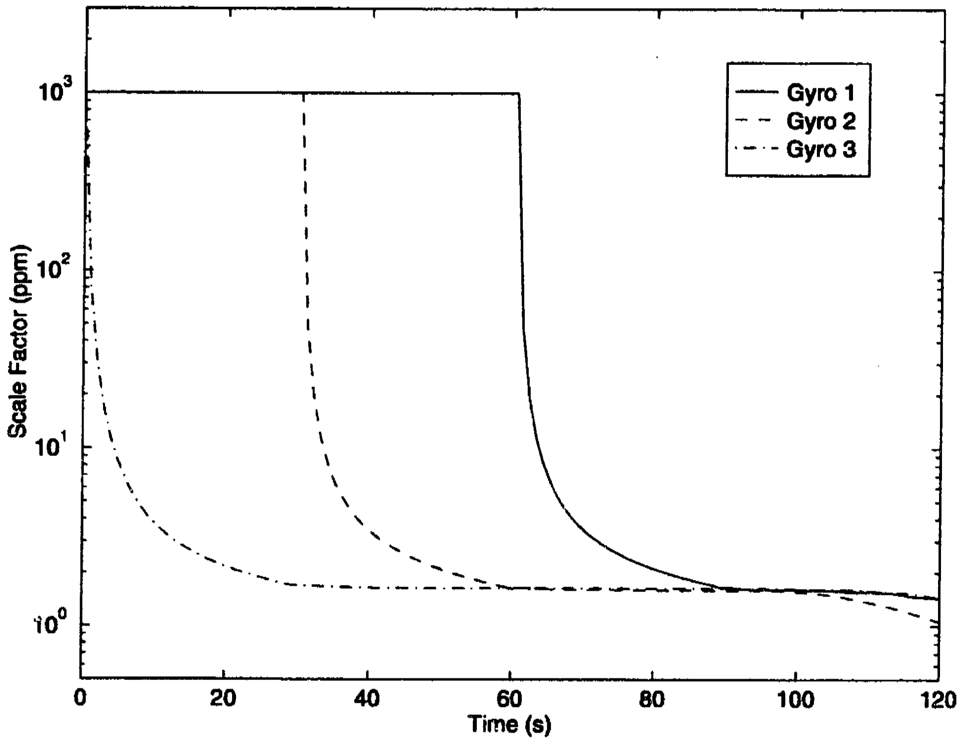


Figure 5-2: Benchmark trajectory gyroscope scale factor 1-σ histories.

filter covariance matrix. Time histories of these calibration uncertainties may be plotted to reveal how the calibration term observabilities vary with the test table maneuvering. Figure 5-2 shows the effect of the benchmark test table trajectory on the uncertainty in the gyroscope scale factor estimates. As expected, the scale factors are most observable during the first three scale factor maneuvers. The misalignment observabilities followed a similar trend as seen in Figure 5-3.

During the scale factor maneuvers, only 1 lever arm component of each accelerometer is observable. This is clearly seen in the lever arm estimate uncertainty traces plotted in Figure 5-4. The reason for this effect is that an accelerometer may only measure acceleration along its input axis. When the IMU is spinning about one of its body axes, only the input axis component of centripetal acceleration is sensed by the instrument. The diagram in Figure 5-5 shows, as heavy dashed lines, the

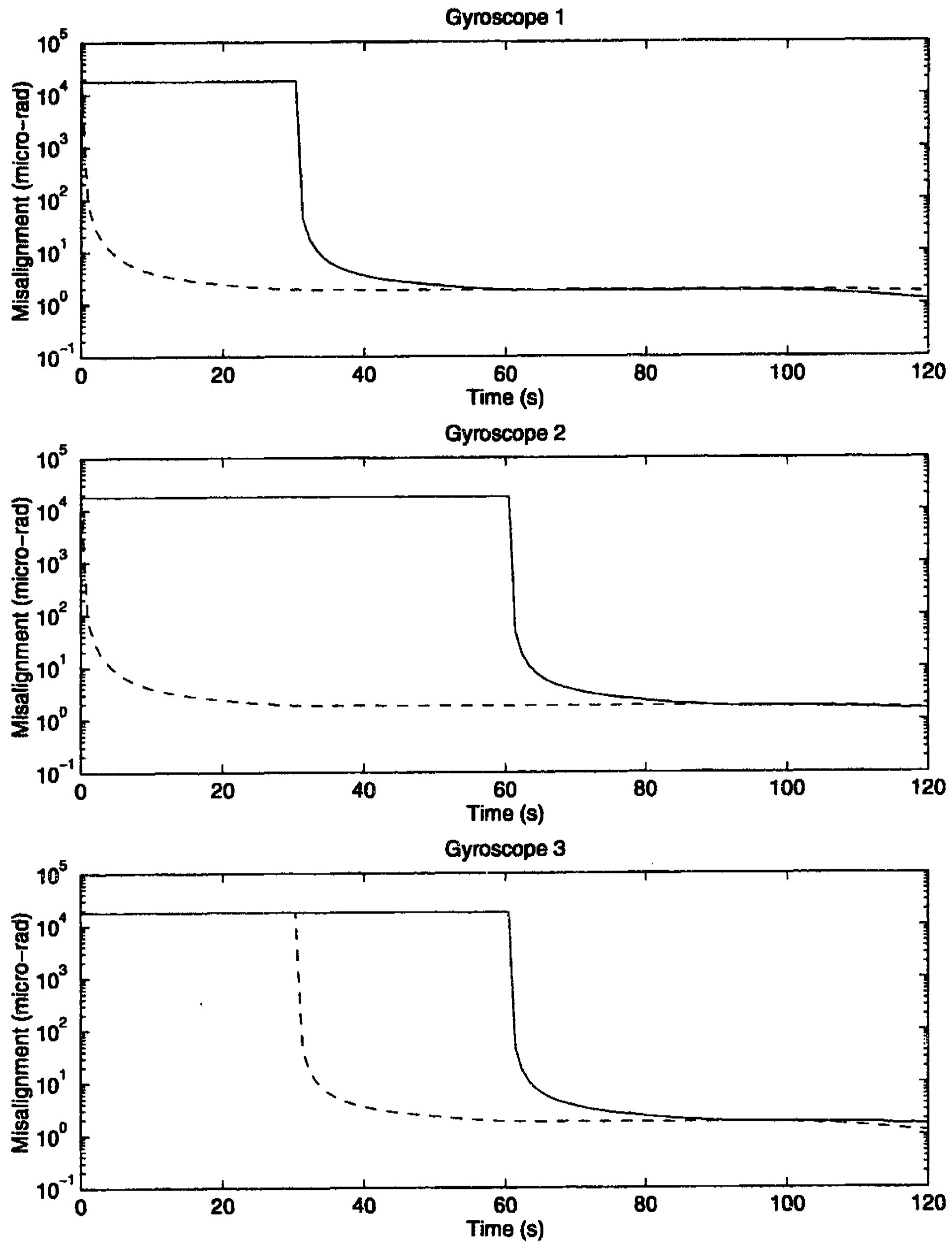


Figure 5-3: Benchmark trajectory gyroscope misalignment 1-σ histories.

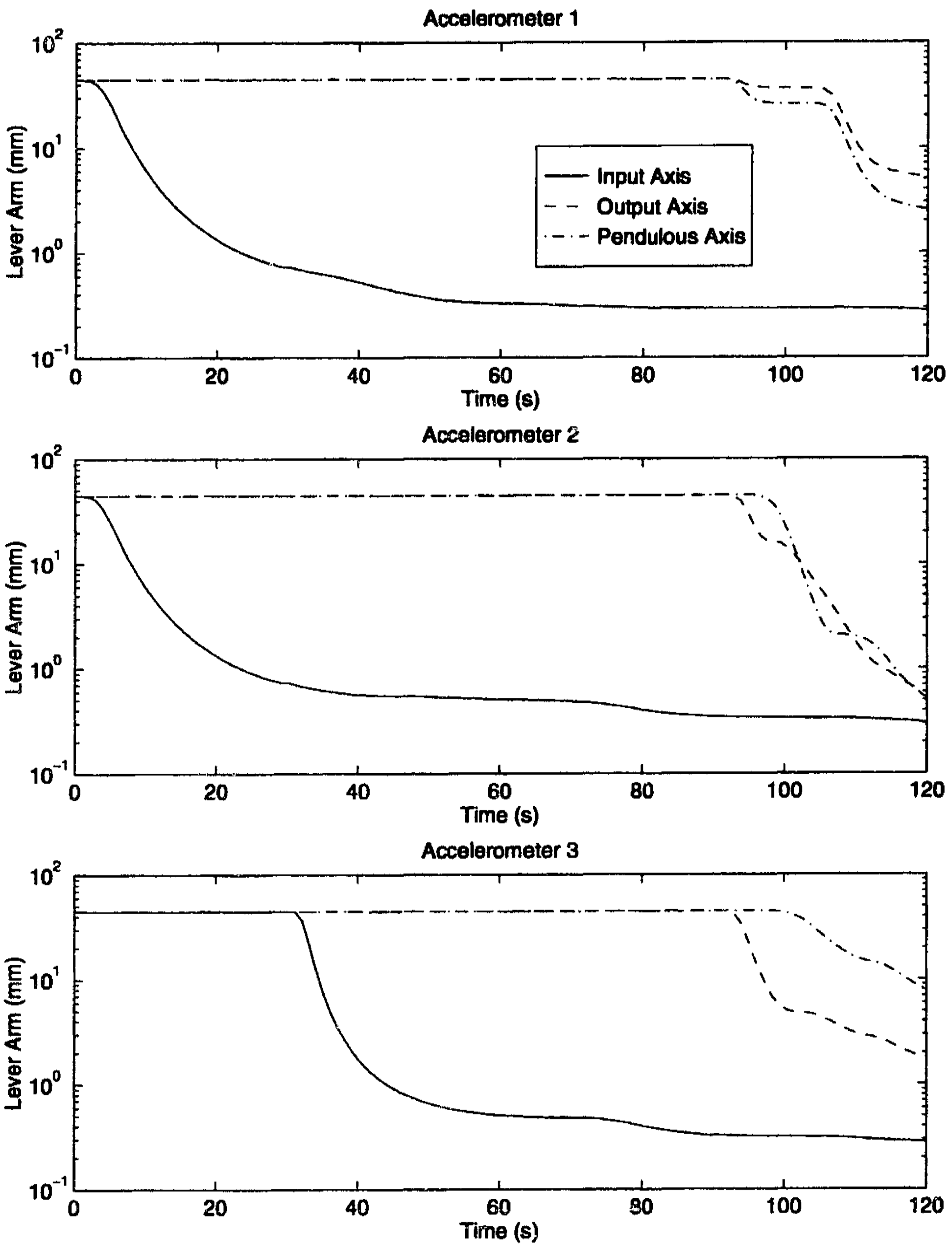


Figure 5-4: Benchmark trajectory accelerometer lever arm 1-σ histories.

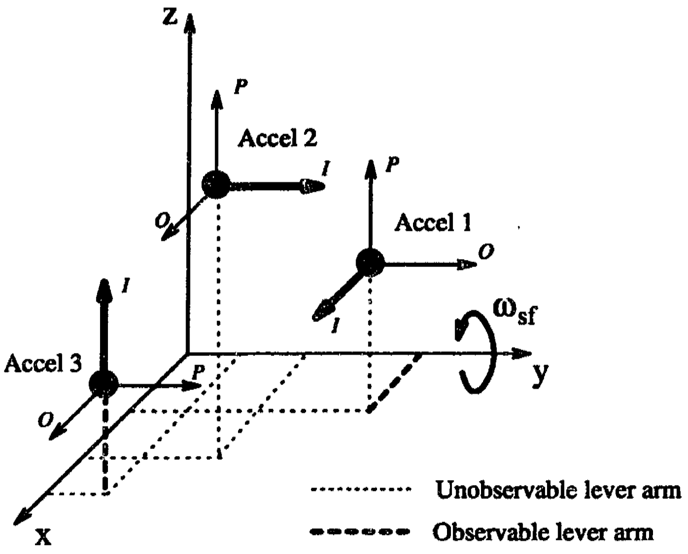


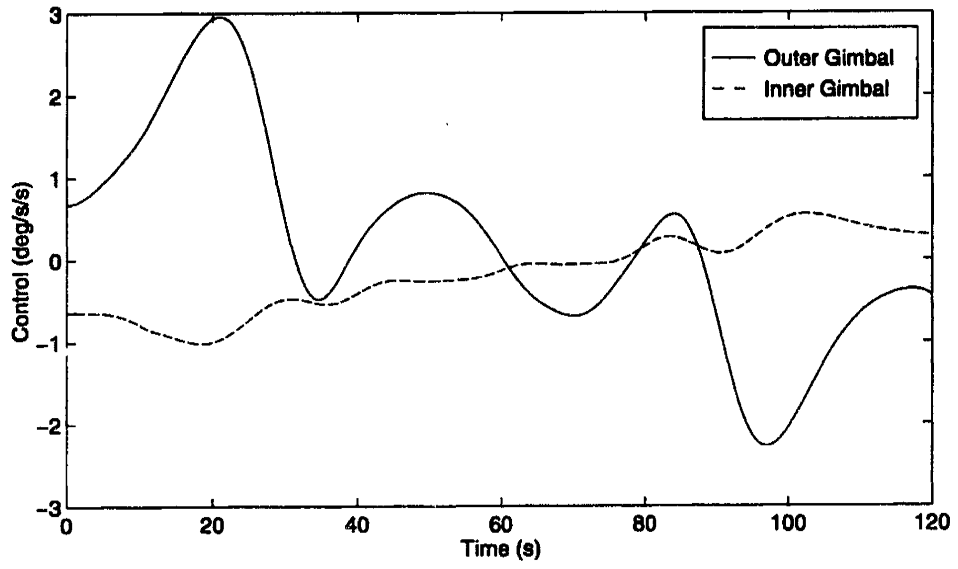
Figure 5-5: Lever arm component observability during body axis angular rate.

two accelerometer lever arm components that are observable when the IMU is spun at a constant angular velocity about its y-body axis. The remaining 6 lever arm components are calibrated during the final 30 second lever arm maneuver.

5.1.2 Optimized Trajectory

Several optimal trajectories for the ring laser gyro IMU were generated using different initial control trajectories. The test table control that produced the lowest covariance matrix trace is plotted in Figure 5-6 with the resulting calibration uncertainties resulting given in Table 5.2. These uncertainties represent a significant improvement on the benchmark case. A list of the average percent reduction in calibration uncertainty for each error term is shown in Table 5.3. For every error term, a greater than 50% improvement in calibration uncertainty is achieved. There would be a smaller difference between the calibration uncertainties, especially in scale factor, if the numerical problems were less severe. Even though the optimal trajectory is computed with same covariance propagation equation as the benchmark, it is able to attain higher gimbal angular velocities since it does not isolate any of the error terms.

The time histories of the scale factor, misalignment and lever arm uncertainties



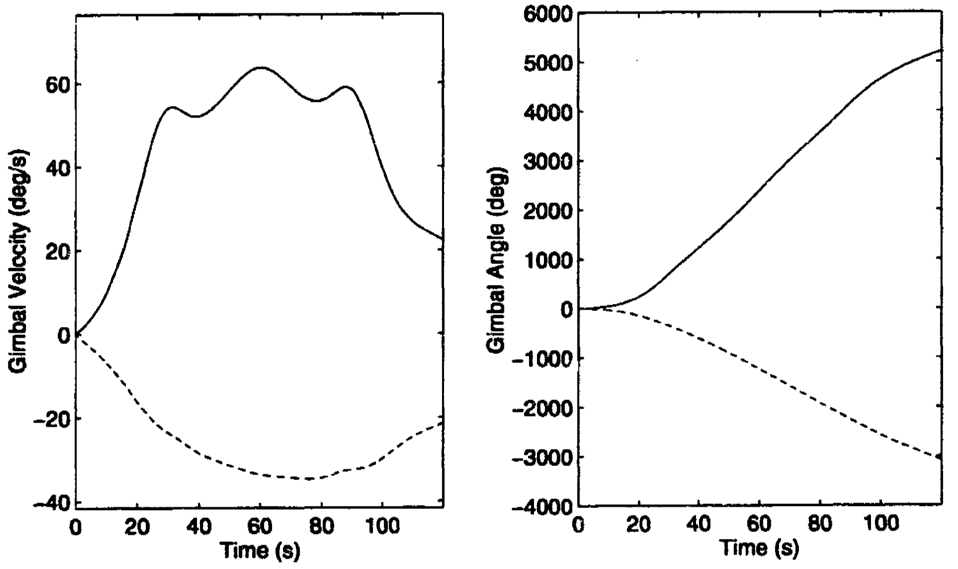


Figure 5-6: Optimized laser IMU test table gimbal controls, velocities and angles.

Optimized Trajectory: State Estimate 1-σ				
State	*Axis 1*	*Axis 2*	*Axis 3*	*Units*
Scale Factor	0.4176	0.8684	0.4187	ppm
Misalignment 1	0.7195	0.6417	0.6468	μrad
Misalignment 2	0.6512	0.6478	0.7243	μrad
Lever Arm 1	0.3369	0.3667	0.4775	mm
Lever Arm 2	2.076	0.3108	1.546	mm
Lever Arm 3	0.5501	0.3903	0.3495	mm

Table 5.2: Optimized laser IMU trajectory calibration accuracy.

Calibration term	*Reduction in 1-σ uncertainty*
Scale Factor	57 %
Misalignment 1	54 %
Misalignment 2	54 %
Lever Arm 1	55 %
Lever Arm 2	71 %
Lever Arm 3	62 %

Table 5.3: Calibration improvement by optimal laser IMU trajectory.

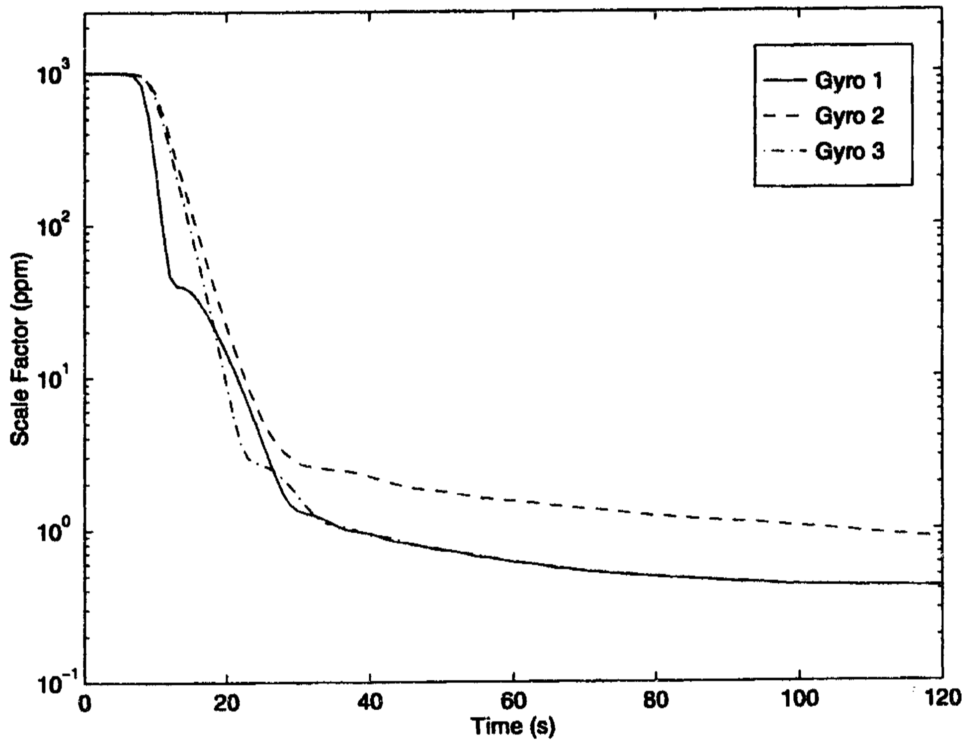


Figure 5-7: Optimized trajectory gyroscope scale factor 1-σ histories.

are plotted in Figures 5-7, 5-8 and 5-9 respectively. The plots show that the optimal trajectory is able to reduce the calibration uncertainties for every error term simultaneously. Also, most of the uncertainty is diminished within the first 30 seconds of test table maneuvering.

5.2 Micro-Mechanical IMU

5.2.1 Benchmark Trajectory

The benchmark test table trajectory used to calibrate the micro-mechanical IMU follows a similar pattern as the ring laser gyro IMU trajectory. It is also comprised of 3, 30 second scale factor maneuvers and a 30 second lever arm maneuver. In this case, however, three different angular rates are used in each scale factor maneuver.

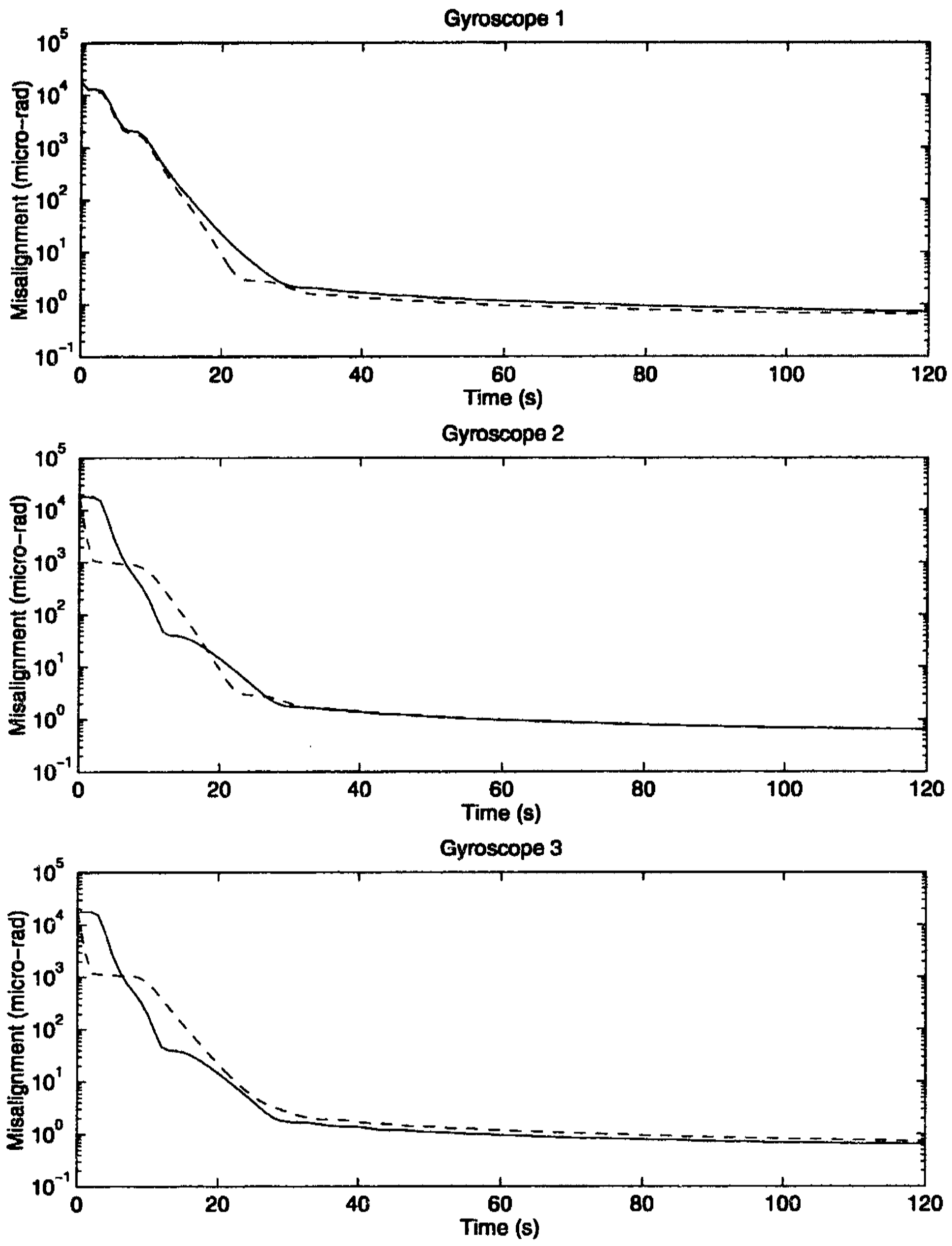


Figure 5-8: Optimized trajectory gyroscope misalignment 1-σ histories.

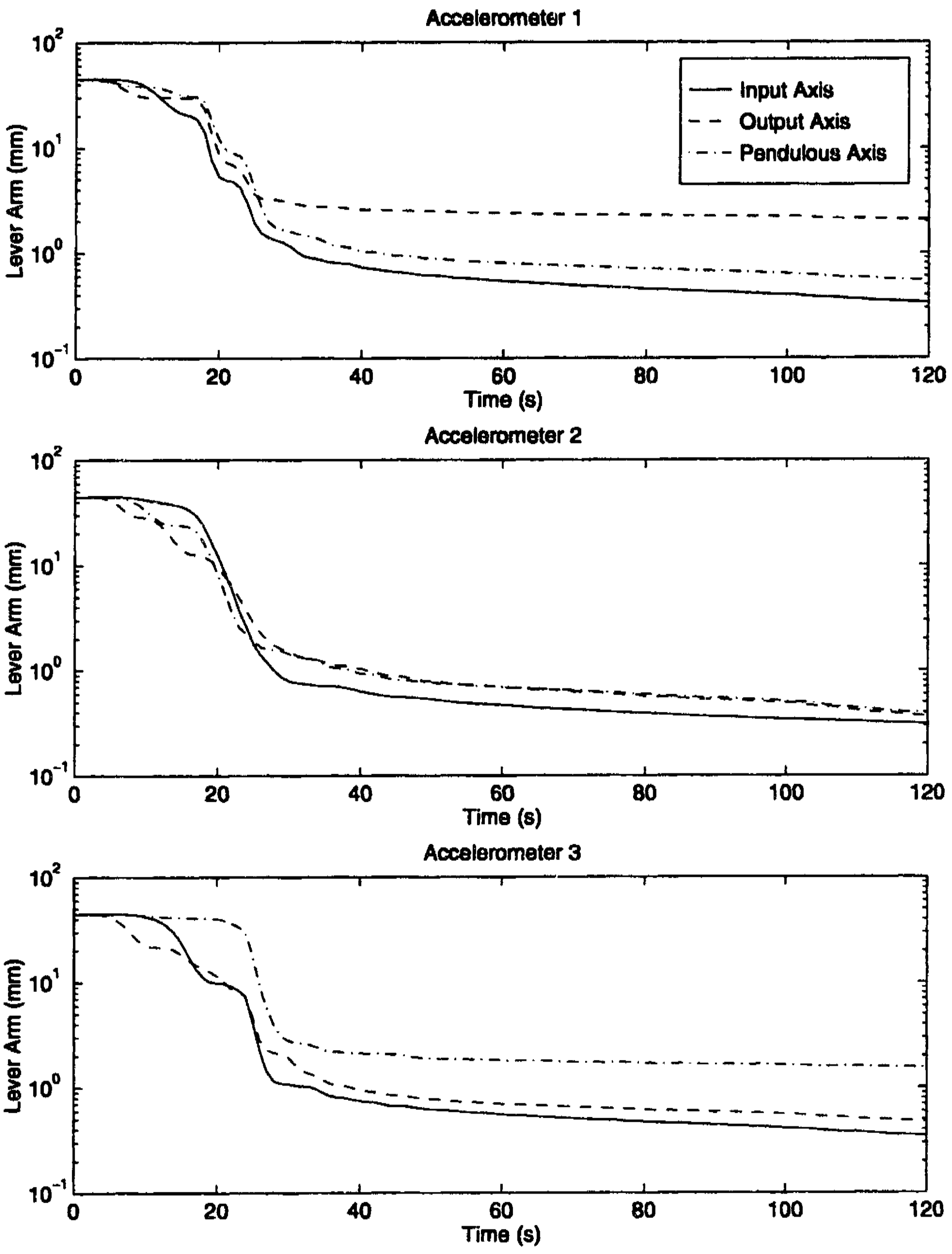


Figure 5-9: Optimized trajectory accelerometer lever arm 1-σ histories.

Benchmark Trajectory: State Estimate 1-σ				
State	*Axis 1*	*Axis 2*	*Axis 3*	*Units*
Scale Factor	85.30	76.24	85.62	ppm
ω^2	40.78	48.42	41.16	ppm/rad/s
ω^3	32.12	29.59	32.35	ppm/rad^2/s^2
Misalignment 1	48.81	50.05	50.05	μrad
Misalignment 2	50.07	50.16	48.82	μrad
Lever Arm 1	0.2069	0.3416	2.038	mm
Lever Arm 2	3.852	0.2082	6.363	mm
Lever Arm 3	1.925	0.3257	0.2014	mm

Table 5.4: Micro IMU benchmark trajectory calibration accuracy.

This more complicated maneuver is needed to calibrate the nonlinear scale factor. The test table gimbal angles and angular velocities for this trajectory are plotted in Figure 5-10.

Numerical problems encountered while integrating the covariance matrix differential equation are less severe in this case. This is because the greater process noise variance and the nonlinear scale factor make it difficult to isolate any calibration term. As a result, much higher gimbal angular velocities can be commanded during the scale factor maneuvers. The calibration error standard deviations produced when using this benchmark trajectory are listed in Table 5.4. The calibration uncertainty time histories are plotted to show how the benchmark maneuvers affected the observability of each error term. Scale factor uncertainty is traced in Figure 5-11. These plots show that Kalman filter has difficulty distinguishing the scale factor, ω^2 and ω^3 terms during the scale factor maneuvers. The drop in the uncertainties between 90 and 120 seconds indicates that the lever arm maneuver helps to increase the observability of these terms. The scale factor maneuvers, however, are able to produce high observability of the gyroscope misalignments as seen in Figure 5-12. Finally, the lever arm uncertainty time histories, plotted in Figure 5-13, behave as expected. Three lever arm components are highly observable during the scale factor maneuvers, and the remaining 6 terms are unobservable until the lever arm maneuver.

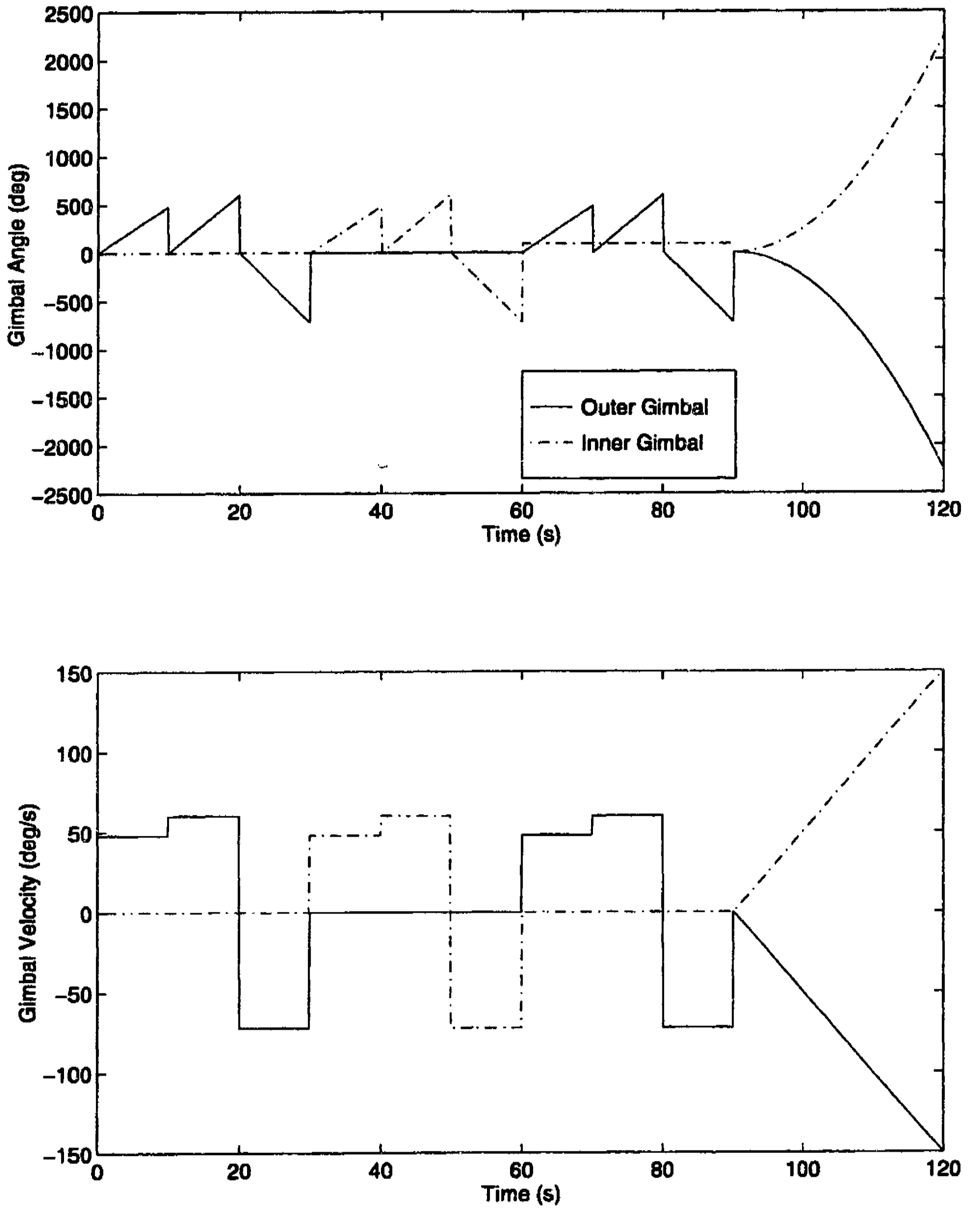
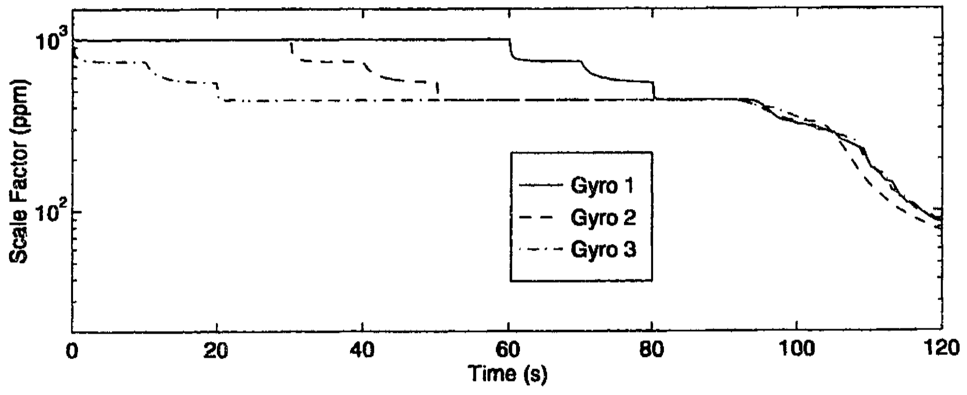
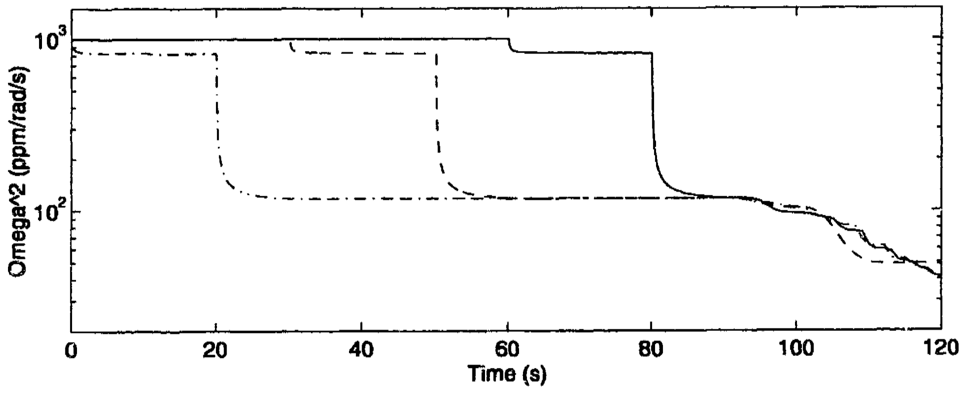


Figure 5-10: Micro IMU benchmark test table gimbal angles and velocities.





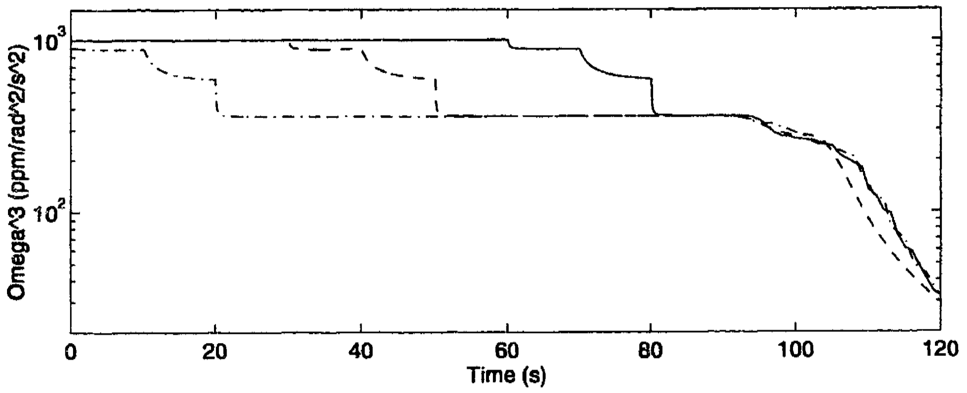


Figure 5-11: Benchmark trajectory gyroscope scale factor 1-σ histories.

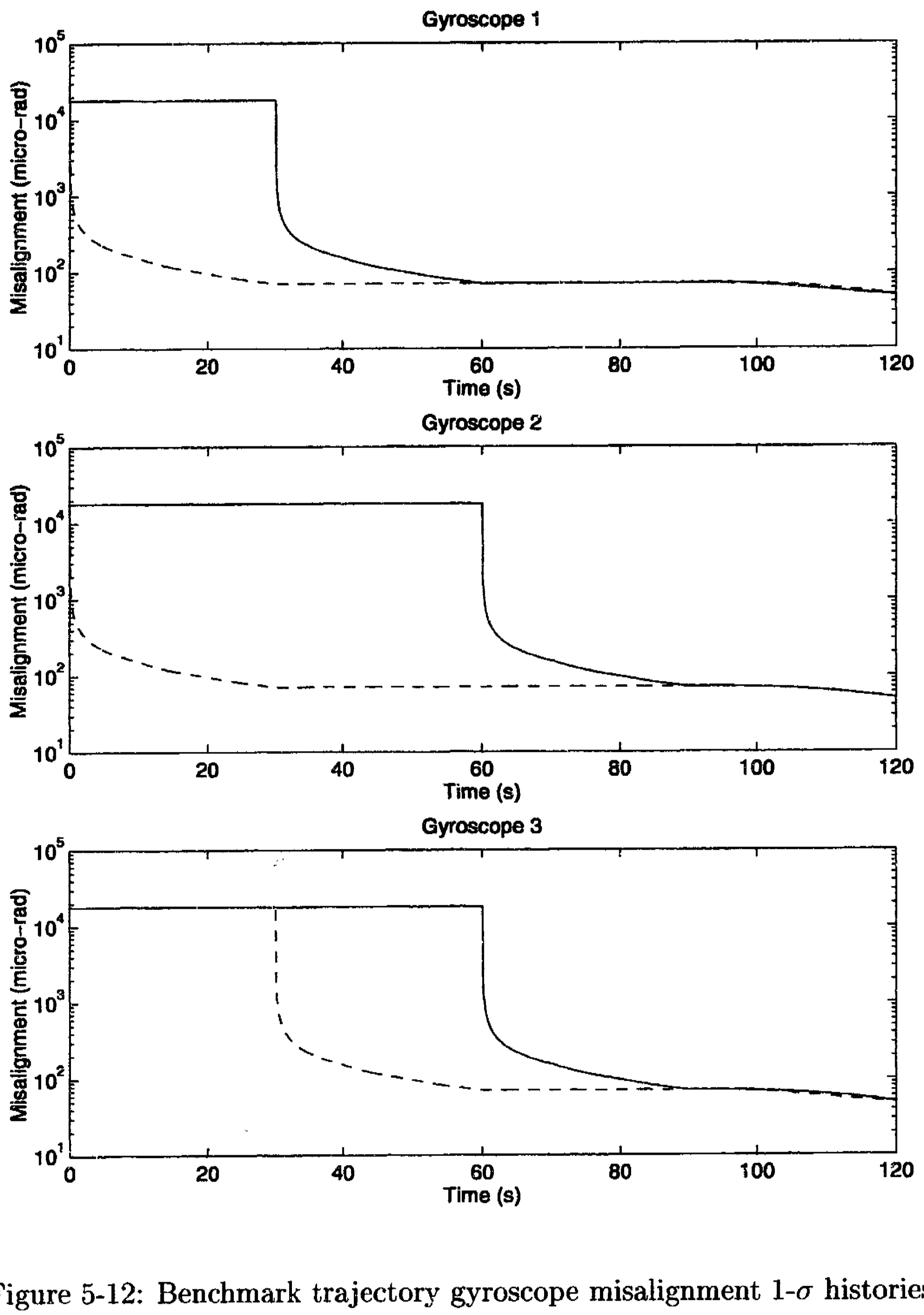


Figure 5-12: Benchmark trajectory gyroscope misalignment 1-σ histories.

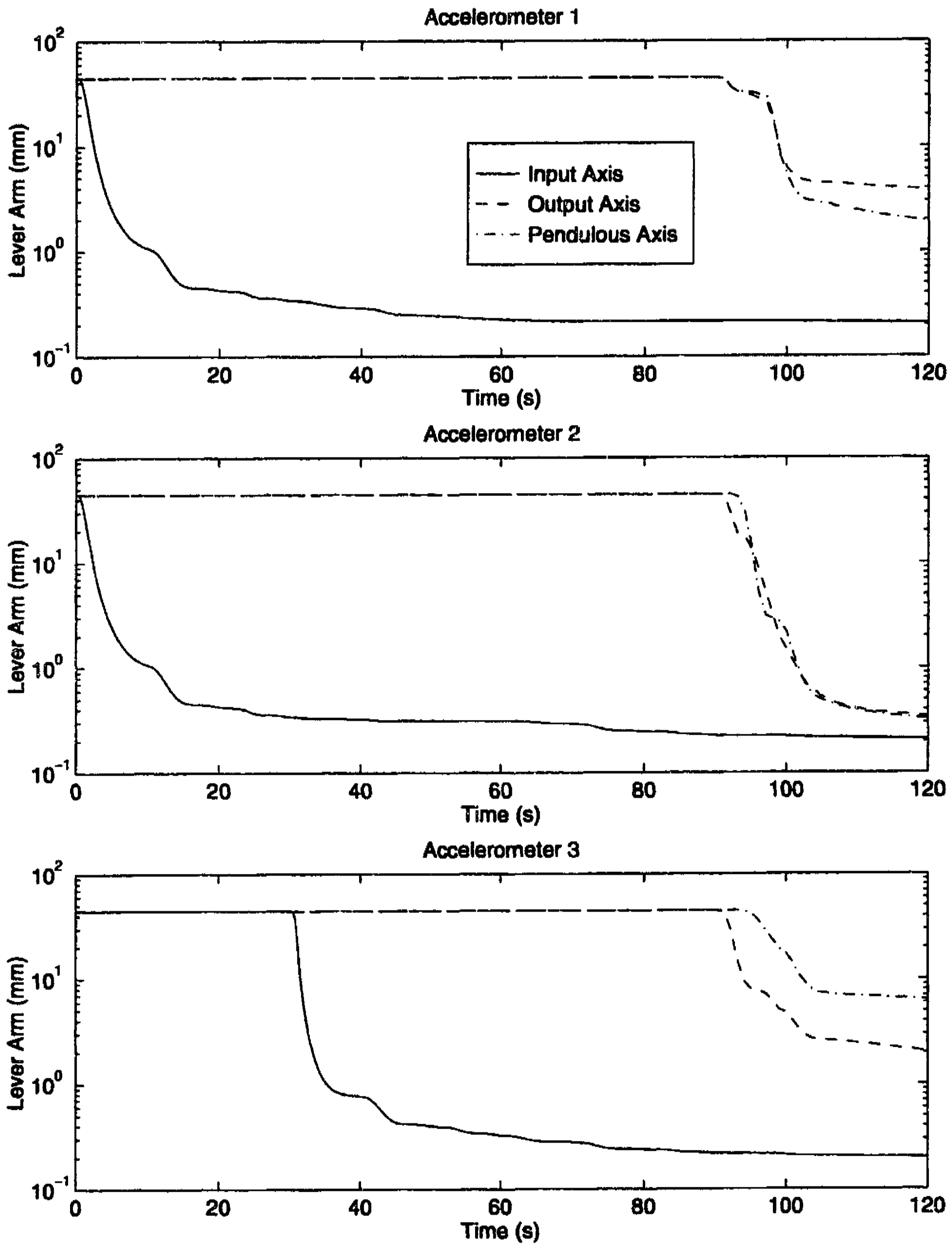


Figure 5-13: Benchmark trajectory accelerometer lever arm 1-σ histories.

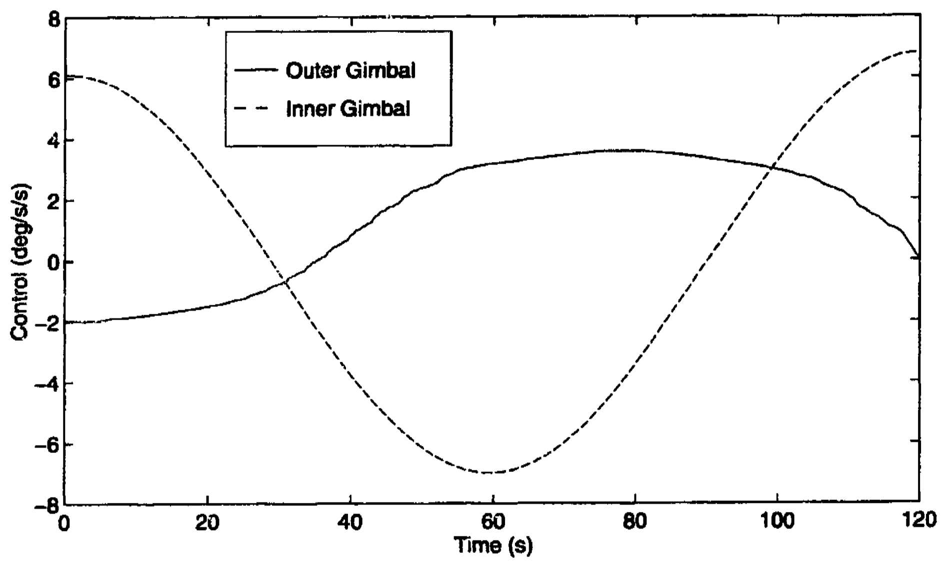
Optimized Trajectory: State Estimate 1-σ				
State	*Axis 1*	*Axis 2*	*Axis 3*	*Units*
Scale Factor	72.15	63.11	71.72	ppm
ω^2	21.29	16.45	19.94	ppm/rad/s
ω^3	15.05	14.56	14.16	ppm/rad^2/s^2
Misalignment 1	23.97	38.20	38.20	μrad
Misalignment 2	37.15	37.14	23.76	μrad
Lever Arm 1	0.3188	0.3302	0.4755	mm
Lever Arm 2	4.994	0.3297	3.578	mm
Lever Arm 3	0.4821	0.3284	0.3179	mm

Table 5.5: Micro IMU optimized trajectory calibration accuracy.

5.2.2 Optimized Trajectory

Arriving at a good optimal micro-mechanical IMU calibration trajectory required many trials with different initial trajectories. Several of the optimized trajectories represented local minima in the cost function that gave high observability to all calibration terms except the number 2 gyroscope scale factor, ω^2 and ω^3 terms. This problem was fixed by selecting an initial trajectory that made these neglected terms initially observable. The best of the optimized trajectories generated using this technique is plotted in Figure 5-14 with the resulting calibration uncertainties resulting given in Table 5.5. While calibration uncertainties are lower than those from the benchmark trajectory, the improvement is not as pronounced as in the ring laser gyro IMU calibrations.

A list of the average percent reduction in calibration uncertainty for each error term is shown in Table 5.6. The improvement in calibration uncertainty is between 15% and 60% for each error term. The time histories of the scale factor, misalignment and lever arm uncertainties are plotted in Figures 5-15, 5-16 and 5-17 respectively. These plots show that several calibration term uncertainties are reduced simultaneously, but the reduction is not as swift as in the ring laser gyro calibration.



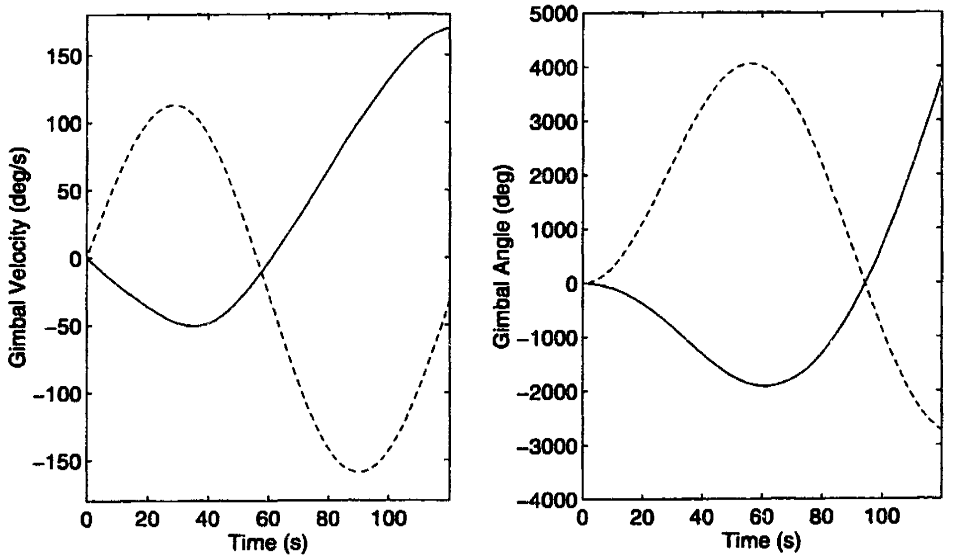


Figure 5-14: Optimized micro IMU test table gimbal controls, velocities and angles.

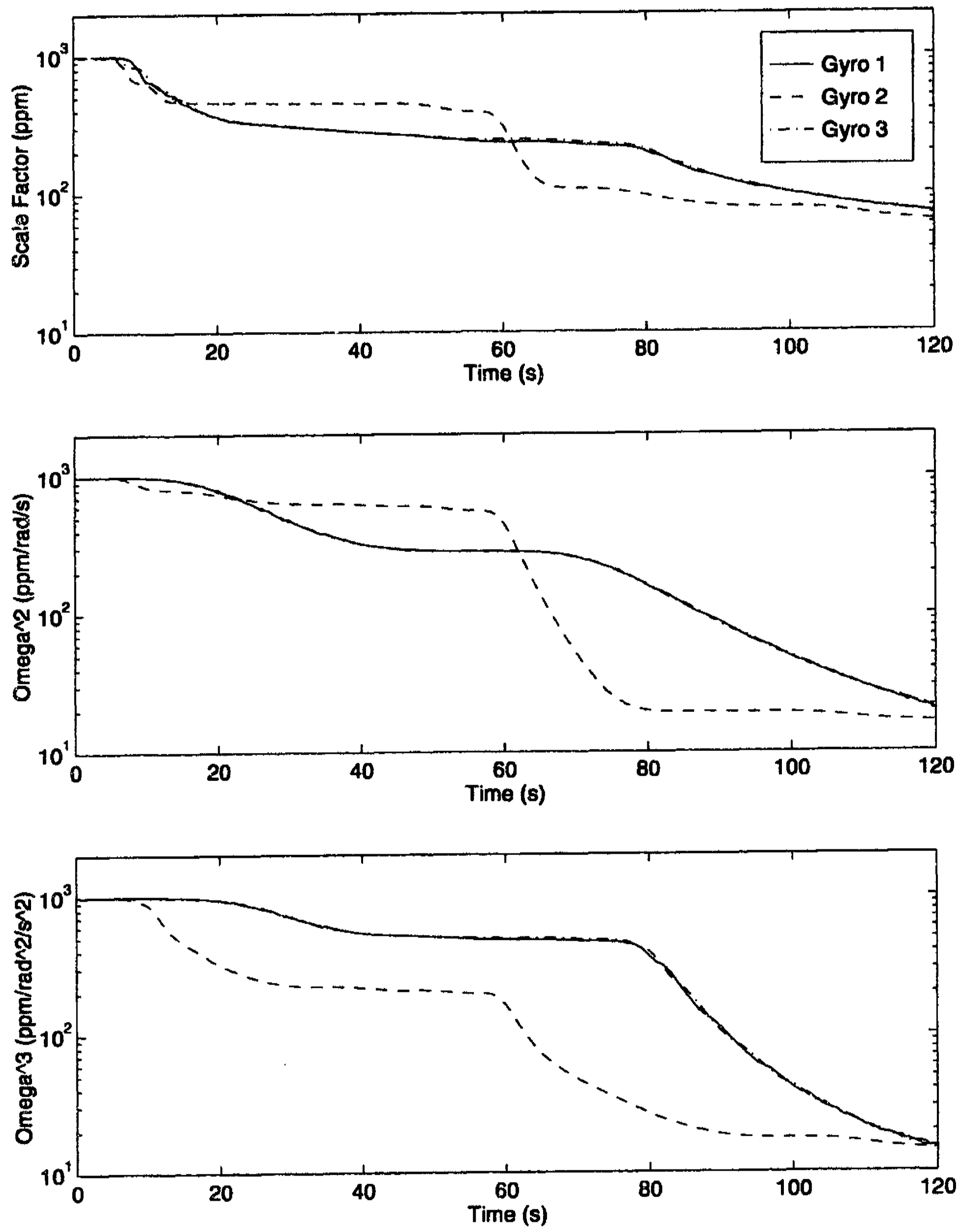


Figure 5-15: Optimized trajectory gyroscope scale factor 1-σ histories.

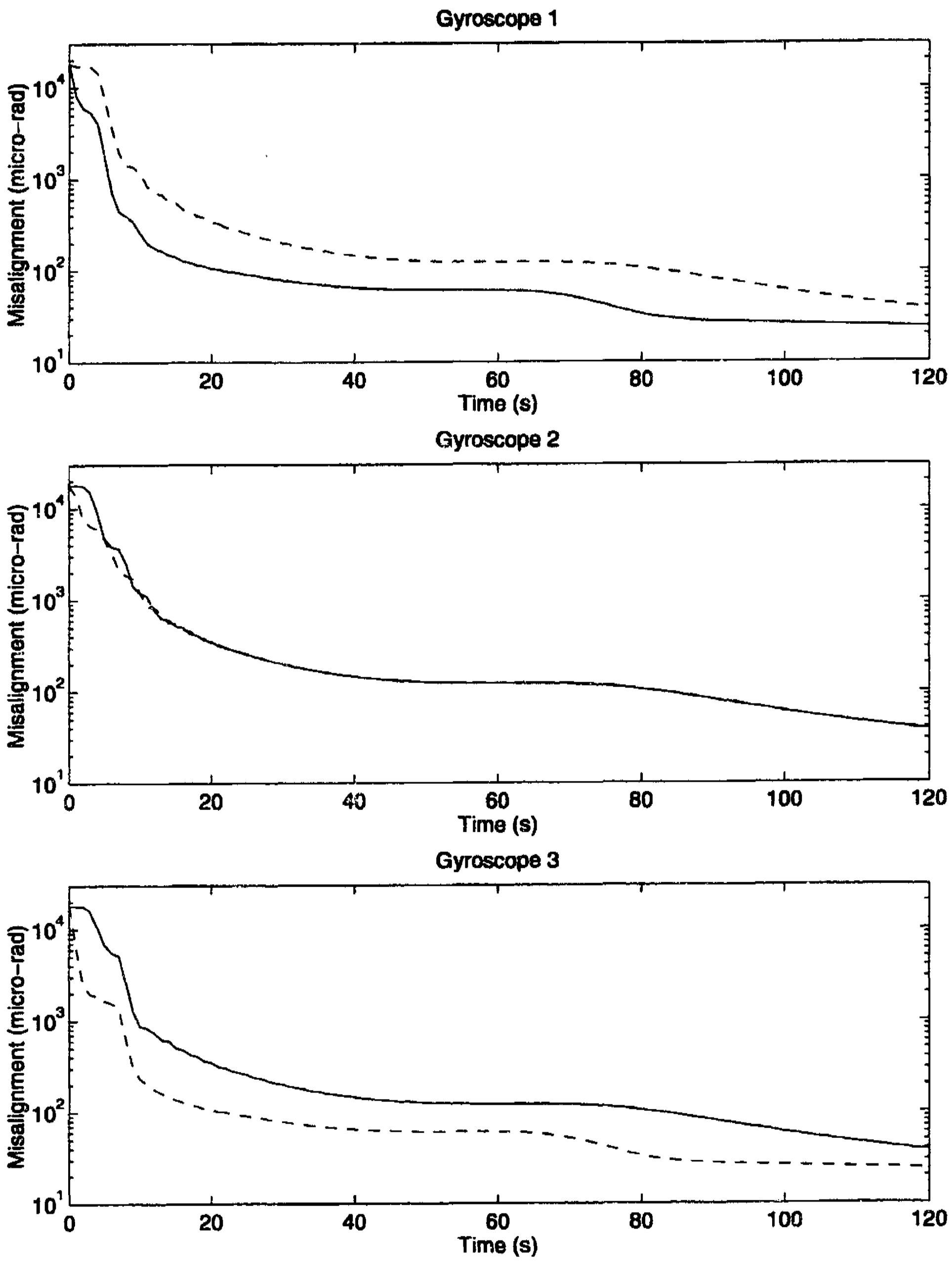


Figure 5-16: Optimized trajectory gyroscope misalignment 1-σ histories.

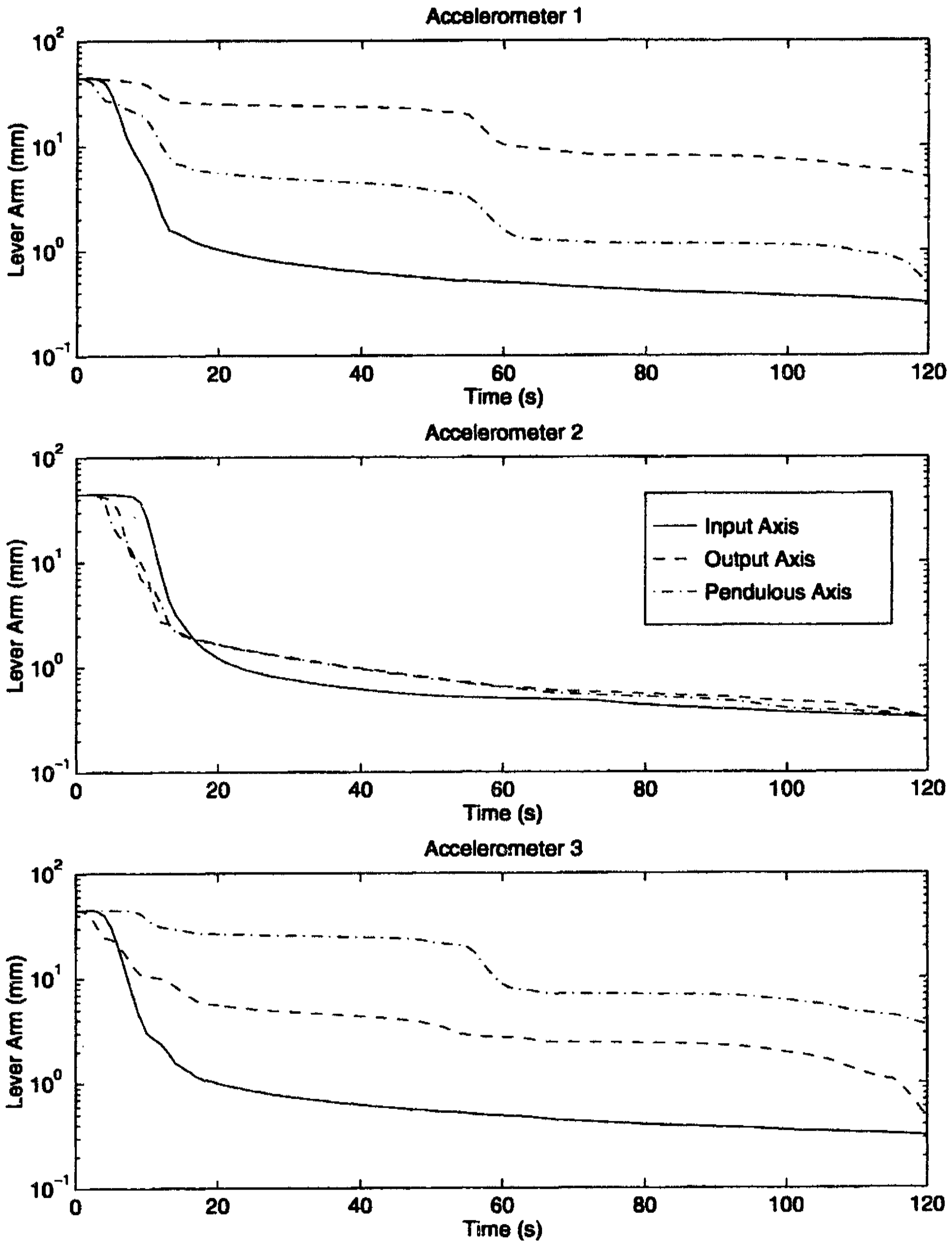


Figure 5-17: Optimized trajectory accelerometer lever arm 1-σ histories.

Calibration term	*Reduction in 1-σ uncertainty*
Scale Factor	16 %
ω^2	56 %
ω^3	53 %
Misalignment 1	33 %
Misalignment 2	34 %
Lever Arm 1	57 %
Lever Arm 2	15 %
Lever Arm 3	54 %

Table 5.6: Calibration improvement by optimal micro IMU trajectory.

Chapter 6

Conclusions

6.1 Summary

The goal of this thesis was to compute optimal trajectories for a 2-axis test table to calibrate dynamic errors in inertial systems. The covariance propagation equation from a Kalman filter was used to determine calibration uncertainty resulting from a test table trajectory. A conjugate gradient optimization algorithm computed trajectories that maximized IMU error observabilities by minimizing the trace of the final covariance matrix. Mathematical models were derived for both ring laser gyro and micro-mechanical IMUs. The models were incorporated in a script for the Mathematica software package which developed the Kalman filter covariance matrix differential equation as well as the adjoint and gradient equations needed for the optimization.

Optimal trajectories were generated that yielded more accurate calibrations than traditional test table trajectories. These trajectories achieved higher calibration accuracy by increasing the observability of several error terms simultaneously instead of isolating the individual errors. The ring laser gyro IMU calibration optimal trajectories could improve calibration accuracy by over 50% while the micro-mechanical optimal trajectories could achieve accuracy improvements that were between 15% and 60%.

Numerical problems associated with the Kalman filter covariance propagation equation limited the range of allowable test table motion. The equation became ill conditioned when an IMU error term observability was high compared to the other terms. This problem was more pronounced in the ring laser gyro IMU case since the linear scale factor model and low process noise variance made it easy to isolate certain error terms. As a result, a direct comparison of the optimal and traditional trajectory for this case unfairly favored the optimal trajectory.

6.2 Proposed Future Work

Numerical problems were frequently encountered while integrating the Kalman filter covariance propagation equation. These problems were caused by high observability of some states which lead to ill conditioned covariance equations. They could be alleviated by incorporating a continuous time version of a square root or U-D factorization Kalman filter algorithm. In these types of algorithms, a square root of the covariance matrix is propagated which increases its precision and guarantees positive definiteness [5].

The trajectory optimization required the storage of the entire control, covariance, adjoint, gradient and direction trajectories. This imposed a tremendous random access memory (RAM) requirement on the computer. As a result, the trajectory lengths were limited to 120 seconds. The amount of RAM required may be reduced by saving the optimization trajectories to disk and loading small portions into RAM as they are needed. This technique will, however, greatly increase the optimization computation time. Other memory savings could come from simplifications in the covariance matrix propagation equation. Since only the diagonal elements of the covariance matrix are required by the cost function, it may be possible to neglect some of the off-diagonal terms without adversely affecting the cost.

The optimal test table trajectories generated in this thesis were found to increase the observability of several calibration terms simultaneously. This property could be exploited to yield minimum time calibration trajectories which would be valuable for fast factory calibrations of mass produced inertial systems. Including time as a parameter to be minimized would require simple modifications to the optimization equations that have already been developed.

Appendix A

Micro-Mechanical System Model Mathematica Script

The trajectory optimization in this thesis requires the determination of a Kalman filter covariance matrix differential equation and its associated gradient and adjoint equations for two types of navigators. The large number of states and the complexity of the navigator models, however, make the derivation of these equations far too difficult to perform by hand. Instead, the Mathematica software package is used to execute the numerous and lengthy symbolic mathematical computations required. This appendix presents the Mathematica script used to derive the necessary trajectory optimization equations for the micro-mechanical IMU.

```
(* Need to add the vector analysis package to perform  *)
   cross product:  <<Calculus‘VectorAnalysis‘          *)


Print["Defining constants and direction cosine matrices."]
(* Constants                                           *)
nu = 2;             (* number of controls              *)
n = 33;             (* number of navigator states      *)
nn = (n n + n)/2    (* number of upper triangular      *)
                    (* elements in an n x n matrix     *)
```

```
HOUR = 3600;        (* seconds/hour                      *)
LAT = 42.364 Degree //N;  (* latitude of gimbal table  *)
WE = 15.041 Degree / HOUR //N;  (* earth rotation rate *)
G = 1;              (* gravity                           *)

(* 2 axis test table rotations: x = outer gimbal,      *)
   y = inner gimbal                                     *)
C1 = {{ Cos[x[t]], Sin[x[t]], 0},
      {-Sin[x[t]], Cos[x[t]], 0},
      {         0,         0, 1}}
C2 = {{ Cos[y[t]], 0, -Sin[y[t]]},
      {         0, 1,          0},
      { Sin[y[t]], 0,  Cos[y[t]]}}

(* Direction cosine matrix from local level UEN to     *)
   sensor platform xyz  *)
Cnb = C2 . C1

(* DCM from body xyz to local level UEN                 *)
Cbn = Transpose[Cnb]

(* Compute angular velocity vector of the sensor       *)
(* platform frame in inertial space                     *)
omegaNB = {-x'[t]*Sin[y[t]], y'[t], x'[t]*Cos[y[t]]}
omegaEARTH = {WE*Sin[LAT], 0, WE*Cos[LAT]}
omegaEARTH = {0, 0, 0}     (* neglect earth rate *)
omegaIN = Cnb . omegaEARTH
omega = omegaIN + omegaNB
omegadot = D[omega,t]
```

```
(* Gravity vector in sensor platform frame                  *)
gB = Cnb . {-G, 0, 0}

Print["Assembling accelerometer errors."]
(* Assemble accelerometer errors                             *)
lax   = {laxx, laxy, laxz}
lax   = CrossProduct[omega, CrossProduct[omega, lax]] +
        CrossProduct[omegadot, lax]
lay   = {layx, layy, layz}
lay   = CrossProduct[omega, CrossProduct[omega, lay]] +
        CrossProduct[omegadot, lay]
laz   = {lazx, lazy, lazz}
laz   = CrossProduct[omega, CrossProduct[omega, laz]] +
        CrossProduct[omegadot, laz]
lever = {lax[[1]], lay[[2]], laz[[3]]}

Print["Assembling gyroscope errors."]
(* Assemble gyroscope errors                                 *)
sf = {sfx*omega[[1]], sfy*omega[[2]], sfz*omega[[3]]}
m1 = {m1x*omega[[2]], m1y*omega[[1]], m1z*omega[[1]]}
m2 = {m2x*omega[[3]], m2y*omega[[3]], m2z*omega[[2]]}
w2 = {w2x*omega[[1]]*omega[[1]], w2y*omega[[2]]*omega[[2]],
w2z*omega[[3]]*omega[[3]]}
w3 = {w3x*omega[[1]]*omega[[1]]*omega[[1]],
w3y*omega[[2]]*omega[[2]]*omega[[2]],
w3z*omega[[3]]*omega[[3]]*omega[[3]]}

eab = lever
egb = sf + m1 + m2 + w2 + w3
```

```
ea = Cbn . eab

eg = Cbn . egb


dV = {dVx, dVy, dVz}

psi = {psix, psiy, psiz}


(* Coriolis term                                          *)

cor = CrossProduct[omegaEARTH, psi]


psidot = eg - cor

dVdot = ea


dR = {dRx, dRy, dRz}

dRdot = dV


Print["Forming system and measurement matrices F and H."]

(* Matrix form of dRdot and dVdot equations               *)

dRMATt = { D[dRdot,dRx]   , D[dRdot,dRy]   , D[dRdot,dRz]   ,

           D[dRdot,dVx]   , D[dRdot,dVy]   , D[dRdot,dVz]   ,

           D[dRdot,psix]  , D[dRdot,psiy]  , D[dRdot,psiz]  ,

           D[dRdot,laxx]  , D[dRdot,laxy]  , D[dRdot,laxz]  ,

           D[dRdot,sfx]   , D[dRdot,m1x]   , D[dRdot,m2x]   ,

           D[dRdot,w2x]   , D[dRdot,w3x]   ,

           D[dRdot,layx]  , D[dRdot,layy]  , D[dRdot,layz]  ,

           D[dRdot,sfy]   , D[dRdot,m1y]   , D[dRdot,m2y]   ,

           D[dRdot,w2y]   , D[dRdot,w3y]   ,

           D[dRdot,lazx]  , D[dRdot,lazy]  , D[dRdot,lazz]  ,

           D[dRdot,sfz]   , D[dRdot,m1z]   , D[dRdot,m2z]   ,

           D[dRdot,w2z]   , D[dRdot,w3z] }
```

```
dVMATt = { D[dVdot,dRx]   , D[dVdot,dRy]   , D[dVdot,dRz]   ,
           D[dVdot,dVx]   , D[dVdot,dVy]   , D[dVdot,dVz]   ,
           D[dVdot,psix]  , D[dVdot,psiy]  , D[dVdot,psiz]  ,
           D[dVdot,laxx]  , D[dVdot,laxy]  , D[dVdot,laxz]  ,
           D[dVdot,sfx]   , D[dVdot,m1x]   , D[dVdot,m2x]   ,
           D[dVdot,w2x]   , D[dVdot,w3x]   ,
           D[dVdot,layx]  , D[dVdot,layy]  , D[dVdot,layz]  ,
           D[dVdot,sfy]   , D[dVdot,m1y]   , D[dVdot,m2y]   ,
           D[dVdot,w2y]   , D[dVdot,w3y]   ,
           D[dVdot,lazx]  , D[dVdot,lazy]  , D[dVdot,lazz]  ,
           D[dVdot,sfz]   , D[dVdot,m1z]   , D[dVdot,m2z]   ,
           D[dVdot,w2z]   , D[dVdot,w3z] }

(* Matrix form of psidot equation                         *)
psiMATt = { D[psidot,dRx]   , D[psidot,dRy]   , D[psidot,dRz]   ,
            D[psidot,dVx]   , D[psidot,dVy]   , D[psidot,dVz]   ,
            D[psidot,psix]  , D[psidot,psiy]  , D[psidot,psiz]  ,
            D[psidot,laxx]  , D[psidot,laxy]  , D[psidot,laxz]  ,
            D[psidot,sfx]   , D[psidot,m1x]   , D[psidot,m2x]   ,
            D[psidot,w2x]   , D[psidot,w3x]   ,
            D[psidot,layx]  , D[psidot,layy]  , D[psidot,layz]  ,
            D[psidot,sfy]   , D[psidot,m1y]   , D[psidot,m2y]   ,
            D[psidot,w2y]   , D[psidot,w3y]   ,
            D[psidot,lazx]  , D[psidot,lazy]  , D[psidot,lazz]  ,
            D[psidot,sfz]   , D[psidot,m1z]   , D[psidot,m2z]   ,
            D[psidot,w2z]   , D[psidot,w3z] }

dRMAT = Transpose[dRMATt]
dVMAT = Transpose[dVMATt]
psiMAT = Transpose[psiMATt]
```

```
F = Join[dRMAT, dVMAT, psiMAT, Table[0, {n-9}, {n}]]
F = F /. {x[t] -> p[nn+1], y[t] -> p[nn+2], x'[t] -> p[nn+3],
    y'[t] -> p[nn+4], x''[t] -> u[1], y''[t] -> u[2]}
Ft = Transpose[F]

(* Measurement geometry matrix                              *)
H1t = Join[IdentityMatrix[3], Table[0, {n-3}, {3}]]
H2t = Join[Table[0,{6},{3}], IdentityMatrix[3],
           Table[0,{n-6-3},{3}]]
H = Join[Transpose[H1t], Transpose[H2t]]
Ht = Transpose[H]

Print["Forming Kalman filter covariance matrix
       differential equations."]
(* Process noise matrix                                     *)
Qvar = Array[q, n];
Q = DiagonalMatrix[Qvar]

(* Measurement noise matrix                                 *)
Rvar = Array[r, Dimensions[H][[1]]];
R = DiagonalMatrix[Rvar]
Rinv = Inverse[R]

(* Covariance matrix element index mapping                  *)
index[i_,j_] := 1+(i-1)n - (i-2)(i-1)/2 /; j==i
index[i_,j_] := index[i,i] + (j-i) /; j>i
index[i_,j_] := index[j,i] /; j<i

P = Table[p[index[i,j]], {i,n}, {j,n}]
```

```
(* Kalman filter covariance dynamics                          *)
Pd = F . P + P . Ft + Q - P . Ht . Rinv . H . P
Pd = Pd /. 0.->0

Print["Appending gimbal table dynamics equations."]
(* Gimbal table dynamics                                      *)
X = {p[nn+1], p[nn+2], p[nn+3], p[nn+4]} (* states are        *)
                    (* gimbal angles and angular rates        *)
U = {u[1], u[2]}    (* controls are gimbal accelerations      *)
A = {{0,0,1,0},{0,0,0,1},{0,0,0,0},{0,0,0,0}}
B = {{0,0},{0,0},{1,0},{0,1}}

Xdot = A . X + B . U

Print["Forming penalty and terminal cost functions."]
(* Integrand of state and control penalty function            *)
g =(p[nn+3]/WMAX)^20 + (p[nn+4]/WMAX)^20 +
   (u[1]/AMAX)^20 + (u[2]/AMAX)^20;

(* Terminal cost: Trace of the final covariance matrix        *)
tcost = Sum[p[index[i,i]], {i,n}];

Print["Defining Hamiltonian."]
(* Lagrange multipliers                                       *)
LAM = Array[lam, nn]
LAMX = {lam[nn+1], lam[nn+2], lam[nn+3], lam[nn+4]}

(* Hamiltonian formulation                                    *)
Hamtemp1 = 0;
```

```
Do[ Do[ If[ j>=i, Hamtemp1 = Hamtemp1 +
           LAM[[index[i,j]]] Pd[[i,j]]], {i, n}], {j, n}]
Hamtemp2 = LAMX . Xdot


Ham = g + Hamtemp1 + Hamtemp2


Print["Defining gradient."]
(* Gradient formulation                                      *)
grd = {D[Ham,u[1]], D[Ham,u[2]]}


Print["Defining adjoint equations."]
(* Adjoint formulation                                       *)
Do[lamdot[i] = -D[Ham, p[i]], {i, nn}]
Do[lamdot[i] = -D[Ham, X[[i-nn]]], {i, nn+1, nn+4}]


Print["Unwrapping covariance equations."]
(* Covariance dynamics unwrapped                             *)
Do[ Do[ If[ j>=i, Pdot[index[i,j]] =
           Pd[[i,j]]], {i, n}], {j, n}];
Do[ Pdot[i] = Xdot[[i-nn]], {i, nn+1, nn+4}];


Print["Writing C-form of gradient, adjoint and
      covariance equations."]
(* C-form of gradient, adjoint and system equations          *)
Do[ lamdot[i] = CForm[lamdot[i]], {i,nn+4}];
Do[ Pdot[i] = CForm[Pdot[i]], {i,nn+4}];
Do[ grad[i] = CForm[grd[[i]]], {i,nu}];
g = CForm[g];


Print["Saving equations to files."]
```

```
Print["   Covariance,"]
Save["pdot", Pdot];
Print["   Adjoint,"]
Save["lamdot", lamdot];
Print["   Gradient,"]
Save["grad", grad];
Print["   Cost function,"]
Save["g", g];
Print["DONE."]

(* NOTES:                                             *)
(* The system states in the optimization problem are the *)
(* Kalman filter state covariances given by p[1] through *)
(* p[nn] and the gimbal table angles and angular rates   *)
(* which are p[nn+1] though p[nn+4] for a two axis table.*)
```

Appendix B

User Guide to the General Unconstrained Trajectory Optimization Software

B.1 Introduction

The purpose of this document is to serve as a user's guide and reference manual to the software package created to optimize trajectories using the conjugate gradient method [14]. The software was developed at Draper Laboratory in conjunction with research towards a master of science degree at MIT to compute optimal test table trajectories for calibrating inertial systems. Although originally created to solve a very specific problem, the software was designed to be easily modified to solve a general class of trajectory optimization problems. The program is written entirely in the C programming language, however little knowledge of C is required to effectively modify the code to solve a particular problem.

B.2 Optimization Method

B.2.1 Problem Class

This program solves the class of continuous time, unconstrained, fixed final time, trajectory optimization problems. The general problem in this class has a cost function of the form

$$J = c(\mathbf{x}(t_f)) + \int_0^{t_f} d(\mathbf{x}(t), \mathbf{u}(t))\, dt \tag{B.1}$$

subject to the system dynamics

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \tag{B.2}$$

with the initial state $x(0)$. In these equations, x is an n vector, u is an m vector and t_f is the final time. The goal is to minimize the cost J by optimizing the control trajectory u.

B.2.2 Conjugate Gradient Algorithm

Because of their complexity, trajectory optimization problems typically need to be solved numerically. The relative simplicity of gradient algorithms make them among the most widely used numerical methods to solve these problems. Of the gradient algorithms, the first order method of steepest descent and the second order Newton-Raphson method are the best known [17]. Both of these methods however have disadvantages which limit their usefulness. Steepest descent is the most simple of the gradient algorithms but is slow to converge when the solution nears the optimum. Newton-Raphson is quicker to converge, but is prone to instability and requires a second order gradient computation which is usually difficult and sometimes impossible [8]. The conjugate gradient method of trajectory optimization requires the computation of only a first order gradient but has been shown to possess the convergence properties of second methods while maintaining greater stability [14].

Implementing the conjugate gradient algorithm requires the formation of a Hamiltonian function. The Hamiltonian is defined to be

$$H(\mathbf{x}, \mathbf{u}, \mathbf{p}) = d(\mathbf{x}, \mathbf{u}) + \mathbf{p}^T \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{B.3}$$

where $\mathbf{p}$ is obtained in the solution of the adjoint equation

$$\dot{\mathbf{p}}(t) = -\frac{\partial}{\partial \mathbf{x}} H(\mathbf{x}, \mathbf{u}, \mathbf{p}) \tag{B.4}$$

with the boundary condition

$$\mathbf{p}(t_f) = \left.\frac{\partial c}{\partial \mathbf{x}}\right|_{t=t_f}. \tag{B.5}$$

The gradient is then computed to be the following:

$$\mathbf{g}(\mathbf{u}) = \frac{\partial H}{\partial \mathbf{u}}. \tag{B.6}$$

Let the trajectory $\mathbf{u}^k(t)$ be the k^{th} approximation to the optimal trajectory. The algorithm is started by selecting an arbitrary initial trajectory $\mathbf{u}^0$, integrating the state equations (B.2) and storing the state trajectories. The adjoint equations (B.4) are then solved *backwards* in time and their trajectories stored. Finally the gradient is computed from equation (B.6). After these initial computations the algorithm proceeds as follows:

$$\begin{aligned}
\mathbf{s}^0 &= -\mathbf{g}^0 && \text{(B.7)}\\
\alpha^i &= \arg\min_{\alpha \geq 0} J(\mathbf{u}^i + \alpha \mathbf{s}^i) && \text{(B.8)}\\
\mathbf{u}^{i+1} &= \mathbf{u}^i + \alpha^i \mathbf{s}^i && \text{(B.9)}\\
\mathbf{g}^{i+1} &= \mathbf{g}(\mathbf{u}^{i+1}) && \text{(B.10)}\\
\mathbf{b}^i &= \frac{\langle \mathbf{g}^{i+1}, \mathbf{g}^{i+1} \rangle}{\langle \mathbf{g}^i, \mathbf{g}^i \rangle} && \text{(B.11)}\\
\mathbf{s}^{i+1} &= -\mathbf{g}^{i+1} + \mathbf{b}^i \mathbf{s}^i && \text{(B.12)}
\end{aligned}$$

where

$$\langle \mathbf{g}^i, \mathbf{g}^j \rangle = \int_0^{t_f} \mathbf{g}^i(t) \mathbf{g}^j(t)\, dt. \tag{B.13}$$

Equation (B.8) shows that the conjugate gradient method reduces an trajectory optimization problem into a series of line minimizations. This property unfortunately also represents a major difficulty with the algorithm. Depending on the complexity of the system, performing this line minimization can become very computationally expensive [14] [17]. A compromise which slows the convergence of the algorithm but is computationally feasible is to keep reducing the step size α_i by a constant reduction factor until the cost begins to increase and then choosing the step size that produced the lowest cost. In practice, this step size selection method does indeed work, but care must be taken in selecting the step size reduction factor. Choosing a reduction factor that is too small will require a large number of function evaluations while one that is too large may not produce a good approximation to the minimum cost. In this software a step size adjustment logic has been developed which attempts to maximize the speed of convergence while minimizing the number of computations. This logic will be discussed in Section 5.

Another limitation of the conjugate gradient algorithm common to all gradient-type algorithms is that only local minima or maxima may be determined. The local extremum that the algorithm finds depends on the choice of initial trajectory. If the problem is suspected of containing several local extrema, the algorithm should be executed numerous times each with a different initial trajectory. While there is still no guarantee that the global minimum or maximum will be found, one of the optimized trajectories may be found to be better than the others.

B.3 Basic Operation

B.3.1 Content and Compilation

The trajectory optimization software consists of nine files of C-code, two header files and a make file. The main program resides in `cg.c` and the remainder of the C-code files are `euler_int.c`, `rk_int.c`, `cg_func.c`, `system.c`, `adjoint.c` `adjoint_bc.c`, `gradient.c` and `cost.c`, the header files are `globe.h`, `cg.h` and the make file is

Makefile.

The instructions in this manual on the compilation and operation of the program are valid for a computer running the UNIX operating system. The preferred compiler is the GNU C compiler gcc. All user commands to be entered verbatim at the UNIX command line prompt denoted >> will appear in the typewriter font. Computer output will also be in typewriter font and it will be distinguished by a ## prompt at the beginning of each line. Input of the user's choice will be italicized. For example the lines

```
>> ls name
## ls: name:  No such file or directory
```

ask the user execute UNIX command to list a file or the contents directory with a name of the user's choice. The computer responds that it cannot find the file or directory the user asked for.

To compile the program for the first time type the commands

```
>> make
```

The program may also be explicitly compiled with the command

```
>> gcc -o sim *.c -lm
```

Both commands will automatically build the entire program the first time and create an executable file named `sim`. If the program is modified, it may be recompiled using one of the commands above. The advantage of using `make` is that if one file has been edited after the initial compilation, only that file will be recompiled where using `gcc` explicitly will rebuild the entire program.

B.3.2 Program Execution

Prior to executing the program the first time, a directory named `DATA` must be created in the same directory as the executable file `sim`. This is done with the command

```
>> mkdir DATA
```

To execute the program then simply type `sim` on the command line. The software responds by asking the user three questions about the final time of the trajectory, the integration time step and the numerical integrator type. An example of the series of commands and computer output seen during an execution is as follows:

```
>> sim
## Enter the final time in seconds:finaltime
## Enter the integration time step in seconds:dt
## Integration method (Forward Euler = 0, Runge-Kutta = 1):int
```

The program then begins the trajectory optimization and reports the latest cost and conjugate gradient step size from equation (B.8) to the screen. Upon completion, the program writes useful data to data files named `adjoint`, `control`, `direction`, `gradient`, `state` and `time_data` in the directory `DATA`.

B.4 Program Structure

The trajectory optimization software was designed to be easily modified so an optimization can be obtained for a general class of problems. The basic components are the main program, numerical integrator, system model, general function library and the output data files.

B.4.1 Main Program Algorithm

The main program resides in the file `cg.c`. This segment of the software calls the functions that control the user interface, memory allocation, the conjugate gradient algorithm and data output. The basic algorithm of the main program is as follows:

- Declare local variables.
- Ask user to define final time, integration time step and integration type.
- Initialize global parameters.

- Allocate memory to arrays.
- Set initial state.
- Generate arbitrary initial control trajectory.
- Integrate state equations.
- Compute initial cost.
- Compute adjoint boundary conditions.
- Integrate adjoint equations.
- Initialize gradient and direction trajectories.
- Iterate on conjugate gradient algorithm until optimization is achieved.
  1. Compute new control.
  2. Integrate state equations.
  3. Compute new cost. If (new cost $>$ old cost), then reduce step size and goto step 1.
  4. Compute adjoint boundary conditions.
  5. Integrate adjoint equations.
  6. Calculate gradient.
  7. Calculate direction.
- Write final data to data files and end.

To maintain modularity in the program each step in main program algorithm is broken down into a logical series of function calls. These functions are contained in the remaining `.c` files.

B.4.2 Numerical Integration Functions

There are two numerical integration methods available in the software: forward Euler and fourth order Runge-Kutta. These methods are used to solve the basic first order differential equation:

$$\dot{x}(t) = f(x(t), t) \quad \text{with} \quad x(0) = x_0 \tag{B.14}$$

The forward Euler method solve a system of first order equations by applying the formula:

$$x_{n+1} = x_n + \Delta T \cdot f(x_n, t_n) \qquad n = 0, 1, \ldots \tag{B.15}$$

where ΔT is the integration time step. This is the simplest and fastest way to solve the first order differential equations in the system but is subject to inaccuracy and instability if ΔT is chosen to be too large or when integrating stiff systems.

The fourth order Runge-Kutta method attempts to improve the accuracy and stability of the forward Euler method without the need to compute higher derivatives [7]. It accomplishes this goal by evaluating $f(x, t)$ at selected sub-points on each subinterval. Differential equations are solved using this method by applying the formula:

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{B.16}$$

where

$$k_1 = \Delta T \cdot f(x_n\,, t_n) \tag{B.17}$$
$$k_2 = \Delta T \cdot f(x_n + \frac{1}{2}k_1\,, t_n + \frac{\Delta T}{2}) \tag{B.18}$$
$$k_3 = \Delta T \cdot f(x_n + \frac{1}{2}k_2\,, t_n + \frac{\Delta T}{2}) \tag{B.19}$$
$$k_4 = \Delta T \cdot f(x_n + k_3\,, t_n + \Delta T) \tag{B.20}$$

In the case of trajectory optimization, the function $f(x, t)$ may not be an explicit function of time but is always implicitly dependent on time through the time varying control trajectory that is being optimized. The control trajectory data is only available at each integer time step but not at the half time steps required by the Runge-Kutta method. An approximation of the half time step control data is obtained by performing a linear interpolation on the surrounding data at the integer time steps. The interpolation reduces the accuracy of the method but in practice it has been observed to still be both more stable and accurate than forward Euler.

The forward Euler and Runge-Kutta numerical integration functions are contained in the files `euler_int.c` and `rk_int.c` respectively. Each file contains two functions, `integrate_system(·)` which integrates the state equations and computes the integral and terminal costs, returning the total cost, and `integrate_adjoint(·)` which integrates the adjoint equations backwards in time.

B.4.3 System Model Files

The files `system.c`, `adjoint.c`, `adjoint_bc.c`, `gradient.c` and `cost.c` contain functions which hold the user defined equations for the system, adjoint, adjoint boundary conditions, gradient and cost. Below is a listing of each file and the equation it contains.

$$\begin{aligned}
\texttt{system.c} &: \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\
\texttt{adjoint.c} &: \dot{\mathbf{p}}(t) = -\tfrac{\partial}{\partial \mathbf{x}} H(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t)) \\
\texttt{adjoint\_bc.c} &: \mathbf{p}(t_f) = \tfrac{\partial c}{\partial \mathbf{x}}\Big|_{t=t_f} \\
\texttt{gradient.c} &: \mathbf{g}(\mathbf{u}) = \tfrac{\partial H}{\partial \mathbf{u}} \\
\texttt{cost.c} &: J = c(\mathbf{x}(t_f)) + \textstyle\int_0^{t_f} d(\mathbf{x}(t), \mathbf{u}(t))\, dt.
\end{aligned} \tag{B.21}$$

The equations in each function must appear in a specific form having proper variable names and using the appropriate C language syntax. These standards are discussed in Section 5, *Modification and Customization.*

B.4.4 General Function Library

The largest library of functions is cg_func.c. This file contains the functions that set the global parameters, allocate memory to one and two dimensional arrays, generate the initial control trajectory and state, compute the conjugate direction trajectory and write data to the screen and data files. It also is home to the function min_J(·) which is responsible for iterating on the steps (1) – (3) in the conjugate gradient algorithm listed in Subsection 4.1 and adjusting the step size until a lower cost is obtained.

B.4.5 Data Handling and Storage

The various trajectories in the program are stored in arrays whose dimensions are determined by the data type, the final time and the integration time step. For instance, consider a system with three states, a final simulation time of $t_f = 10$ seconds and an integration time step of $\Delta t = 0.1$ seconds. The array containing the complete set of state trajectories consists of three rows and N= $\frac{t_f}{\Delta t} = 100$ columns. If the state array is named x, then the ninth element of the second state resides in x[2][9].

At points during the program execution and at its conclusion, several arrays are written to text files in the directory DATA. There are five trajectory files including control, state, adjoint, gradient and direction. These files contain the final trajectories described by the file names. Since these files may be very large, only the control trajectory is written in its entirety. The remaining data is written at a user defined interval. Control of the data write interval will be covered in the next section. An additional non-trajectory file named time_data contains only three numbers, the integration time step, the data file write interval and the final time.

B.5 Modification and Customization

When customizing the software to solve a particular problem, several modifications must be made to the code. The modifications range from substituting in the new system model to altering the conjugate gradient step size logic. The software is designed so that only a minimal number of functions need to be altered during customization. Once the program is modified in any way, it must then be recompiled before execution. This section discusses the details of how to modify the software to solve a new problem by presenting an example of a hypothetical system. To demonstrate the necessary modifications, consider the following trajectory optimization problem:

$$\text{System}: \quad \begin{aligned} \dot{x}_1 &= x_2 + u_1 \\ \dot{x}_2 &= x_1 \\ \dot{x}_3 &= \sin(4x_1) - 3e^{x_2} + 2x_3 + u_2 \end{aligned} \quad ; \quad \mathbf{x}(0) = \begin{Bmatrix} 1 \\ 0 \\ 2 \end{Bmatrix} \tag{B.22}$$

$$\text{Cost}: J = x_1^2 + x_2 + \int_0^{t_f} 4\,e^{x_1} + 5\,x_3^2 + (\ln u_1)^2 + 2\,u_2^2\,dt \tag{B.23}$$

The adjoint equations are then computed to be:

$$\dot{p}_1 = -4\,e^{x_1} - p_2 - 4\,p_3\cos(4x_1) \tag{B.24}$$

$$\dot{p}_2 = -p_1 + 3\,p_3\,e^{x_2} \tag{B.25}$$

$$\dot{p}_3 = -10\,x_3 - 2\,p_3 \tag{B.26}$$

with the boundary conditions

$$p_1(t_f) = 2x_1(t_f) \quad , \quad p_2(t_f) = 1 \quad , \quad p_3(t_f) = 0 \tag{B.27}$$

Calculating the gradient yields:

$$g_1 = \frac{2\,\ln u_1}{u_1} + p_1 \tag{B.28}$$

$$g_2 = 4\,u_2 + p_3 \tag{B.29}$$

The equations for this hypothetical system may be put into the software almost exactly as they appear on paper with a few notable exceptions. General rules for converting the equations into the proper form are:

- Subscripts map to numbers in square brackets, e.g. $p_3 \Rightarrow$ `p[3]`.
- Time derivatives appear as they are spoken, e.g. $\dot{x}_2 \Rightarrow$ `xdot[2]`.
- All multiplications must be explicitly typed with and asterisk, e.g. $4p_2x_1 \Rightarrow$ `4*p[2]*x[1]`.
- Trigonometric, logarithmic, etc. functions must be part of the C math library `math.h` (see [12] for details).
- All lines must end in a semicolon.

Applying these rules on all the equations and inserting the equations into the proper files is the first step in customizing the software. When inserting new equations into the functions, care must be taken to change only the equations and not tamper with the architecture of the function itself.

B.5.1 Equation File Modifications

The following is a list of the equation files that must be modified when inserting a new system into the trajectory optimization software. Each listing begins with the file name, describes its contents and shows exactly how the equations of the hypothetical system should look when inserted into each file.

`system.c` contains the complete system of first order state equations in a single function `sys_eqn(·)`. Converting the state equations of the example into the proper form yields:

```
xdot[1] = x[2] + u[1];
xdot[2] = x[3];
xdot[3] = sin(4.0*x[1]) - 3.0*exp(x[2]) + 2.0*x[3] + u[2];
```

`adjoint.c` contains the set of coupled of first order adjoint differential equations in the function `adj_eqn(·)`. Converting the adjoint equations of the example problem yields:

```
pdot[1] = -4.0*exp(x[1]) - p[2] - 4.0*p[3]*cos(4.0*x[1]);
pdot[2] = -p[1] + 3.0*p[3]*exp(x[2]);
pdot[3] = -10.0*x[3] - 2.0*p[3];
```

`adjoint_bc.c` contains the equations that determine adjoint boundary conditions in the function `adjoint_bc(·)`. Converting the adjoint boundary conditions of the example problem yields:

```
pf[1] = 2.0*x[1][N];
pf[2] = 1.0;
pf[3] = 0.0;
```

`gradient.c` contains the gradient equations in the function `gradient(·)`. The equations in this file are slightly different from the state and adjoint equations. The gradient equations are embedded in a loop, so an extra index is required on each state, control and adjoint element. Conversion of these equations yields:

```
g[1][i] = 2.0*log(u[1][i])/u[1][i] + p[1][i];
g[2][i] = 4.0*u[2][i] + p[3][i];
```

`cost.c` contains the integral and terminal cost equations in two functions `int_cost(·)` and `term_cost(·)`. Both sets of equations require an extra index similar to the gradient equations. Also since the integral cost is approximated as the sum of a series, the cost-to-date, `Jin`, must be added to the equation. The integral cost equations are then converted to:

```
J = Jin + 4.0*exp(x[1][i]) + 2.0*pow(x[2][i],2) +
    5.0*pow(x[3][i],2) + pow(log(u[1][i]),2) +
    2.0*pow(u[2][i],2);
```

and the terminal cost equations are converted to:

```
terminal = pow(x[1][N],2) + x[2][N];
```

If the system is very large, the equations may be quickly generated using the Mathematica software package. A simple script file may be created which derives the adjoint, adjoint boundary condition and gradient equations from the set of first order differential equations representing the system model and the cost function. These files are then be converted into a form similar to what is required by the conjugate gradient software and saved. Below, a Mathematica script is given which generates all the equations for the example system.

```
(* Mathematica script to formulate the gradient           *)
(* and adjoint equations for a basic unconstrained        *)
(* trajectory optimization problem with a given           *)
(* system model and cost function.                        *)

(* Number of states and controls                          *)
n =  3;                 (* number of states   *)
nu = 2;                 (* number of controls *)

(* System model differential equations                    *)
xdot[1] = x[2] + u[1];
xdot[2] = x[1];
xdot[3] = Sin[4*x[1]] - 3*Exp[x[2]] + 2*x[3] + u[2];

(* Integrand of state and control penalty function       *)
d = 4*Exp[x[1]] + 5*x[3]^2 + Log[u[1]]^2 + 2*u[2]^2;

(* Terminal cost: Trace of the final covariance matrix *)
c = x[1]^2 + x[2];
```

```
(*****************************************************)
(*********** CHANGE NOTHING BELOW THIS LINE ************)
(*****************************************************)
(* Create Lagrange multipliers                         *)
P = Array[p, n];


(* Hamiltonian formulation                             *)
Hamtemp = 0;
Do[Hamtemp = Hamtemp + P[[i]] xdot[i], {i, n}];


Ham = d + Hamtemp;


(* Gradient formulation                                *)
Do[grd[i] = D[Ham,u[i]], {i,nu}];


(* Adjoint formulation                                 *)
Do[pdot[i] = -D[Ham, x[i]], {i, n}];


(* Adjoint boundary conditions                         *)
Do[pf[i] = D[c, x[i]], {i,n}];


(* C-form of gradient, adjoint and system equations    *)
Do[ xdot[i] = CForm[xdot[i]], {i,n}];
Do[ pdot[i] = CForm[pdot[i]], {i,n}];
Do[ pf[i]   = CForm[pf[i]], {i,n}];
Do[ grd[i] = CForm[grd[i]], {i,nu}];
d = CForm[d];
c = CForm[c];
```

```
(* Save equations to files                              *)
Print["Saving equations to files."]
Print["   System,"]
Save["xdot", xdot];
Print["   Adjoint,"]
Save["pdot", pdot];
Print["   Adjoint boundary condition,"]
Save["pf", pf];
Print["   Gradient,"]
Save["grd", grd];
Print["   Integral cost,"]
Save["d", d];
Print["   Terminal cost,"]
Save["c", c];
Print["DONE."]
```

Unfortunately the `CForm` command in Mathematica does not actually produce standard C syntax and uses parenthesis instead of square brackets when indexing one and two dimensional arrays. These problems may be fixed by directly editing the equation files saved by the script. While any editor may be used for this task, in practice the UNIX string editor `sed` has been found very useful.

This script may be easily modified to accommodate any system model and cost function. An adept Mathematica programmer make further modifications to the script and may derive the mathematical model of the system itself from the underlying physics thereby eliminating human calculation error.

B.5.2 State and Control Initialization

The functions responsible for creating the initial control trajectory and state are in the file `cg_func.c`. For the example problem, the following equations describing the initial state are substituted into the function `initialize_ state(·)`:

```
x[1][1] = 1.0;
x[2][1] = 0.0;
x[3][1] = 2.0;
```

The initial control trajectories are generated in function `initialize_ control(·)`. Making modifications to this function is the only time a working knowledge of C syntax is necessary. A wide range of trajectories may be generated using `for` loops, `if` statements and various mathematical functions. For the example problem consider an initial control trajectory governed by the following equations:

$$\begin{aligned} u_1(t) &= \begin{cases} 0 & \text{for} \quad t \le \frac{t_f}{2} \\ \sin(t) & \text{for} \quad \frac{t_f}{2} < t \le t_f \end{cases} \\ u_2(t) &= 1 \end{aligned} \tag{B.30}$$

where t_f is the final time in seconds. The equivalent C-code that generates this trajectory is:

```
for (i = 1; i <= N; i++)
  {
     if (i <= N/2)
        u[1][i] = 0.0;
     else
        u[1][i] = sin(i*DT);


     u[2][i] = 1.0;
  }
```

An excellent reference for C syntax is [12].

B.5.3 Global Parameters

The final step in converting the software is properly setting seven global parameters that define the size of the system, the minimum allowable conjugate gradient step size, the step size adjustment logic and the data file sparseness. These parameters are set in the function `parameters(·)` which resides in the file `cg_func.c`.

The first two parameters `CONTROL_DIM` and `STATE_DIM` are defined to be the number of controls and states in the system. Since the example system has two controls and three states, these parameters would be set to 2 and 3 respectively.

The next parameter, `TOL`, is the optimization tolerance. It controls when the program exits based on the conjugate gradient step size. When the step size, α in equation (B.8), falls below the value of `TOL` the trajectory is considered to be optimized and the program exits. It is the responsibility of the user to determine this condition of optimality since there are no hard rules or techniques available.

The conjugate gradient algorithm was shown in Section 2 to require the computation of step size α that minimizes a cost. In practice however, calculating α exactly is usually either computationally not feasible or impossible. To alleviate this problem a step size adjustment logic has been developed which starts with an initial guess for α and chooses subsequent step sizes according to a predetermined set of rules. The three parameters `ALPHA`, `ALP_DEC` and `ALP_INC` are used to control the conjugate gradient step size adjustment logic in the function `min_J(·)`. The step size selection algorithm is started by evaluating the cost with an initial step size $\alpha =$ `ALPHA`. This initial step is deliberately chosen to be too large so that a lower cost will be achieved by reducing the step size. The step size is iteratively reduced according to $\alpha_{new} =$ `ALP_DEC` $\times\ \alpha_{old}$ until $J(\alpha_{new}) > J(\alpha_{old})$. The step α_{old} is then selected to generate the new control. When minimization is performed again, the initial step is chosen to be $\alpha =$ `ALP_INC` $\times\ \alpha_{old}$. The selection of parameters `ALPHA`, `ALP_DEC` and `ALP_INC` depends heavily on the system being optimized. In general, the most conservative practice is to set `ALP_DEC` close to one, and adjust `ALPHA` and `ALP_INC` to increase the rate of convergence.

The final global parameter under the user's control is the data file time increment DTF. Since the final control trajectory is the most important data produced by the program execution, every point of the trajectory is written to its output file DATA/control. To save disk space, the remaining trajectory output files in the directory DATA: state, direction, gradient and adjoint are normally written to at a longer time interval than than the integration time step DT. Thus the parameter DTF is the ΔT of the nonessential data files in seconds. For example, if the integration time step is set to 0.1 seconds but knowledge of the state at a one second time time interval is sufficient for the analysis, the parameter DTF would simply be set to 1.0.

B.6 Additional Comments

The software package documented in this manual implements a conjugate gradient optimization algorithm for solving unconstrained, fixed terminal time, trajectory optimization problems. However, if the problem involves constraints, this software may still be used effectively. When using gradient-type algorithms to solve constrained optimization problems, it is common practice to model state and control constraints by adding penalty functions to the cost function. The final answer will then be an approximation of the exact optimum solution. A detailed discussion of penalty functions may be found in [8].

Bibliography

[1] B.D.O. Anderson and J.B. Moore. *Optimal Filtering.* Prentice Hall, 1979.

[2] G. Arshal. Error equations of inertial navigation. *Journal of Guidance, Control and Dynamics*, 1987.

[3] K.R. Britting. *Inertial Navigation Systems Analysis.* Wiley-Interscience, 1971.

[4] A. Brown, R. Ebner, and J. Mark. A calibration technique for a laser gyro strapdown inertial navigation system. *Symposium Gyro Technology, Stuttgart, West Germany*, 1982.

[5] R.G. Brown and P.Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering.* John Wiley and Sons, 1983.

[6] A.E. Bryson and Y.C. Ho. *Applied Optimal Control.* Blaisdell Publishing Co., 1969.

[7] S.D. Conte and C. de Boor. *Elementary Numerical Analysis.* McGraw-Hill, 1980.

[8] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Siam, 1990.

[9] A. Gelb (ed.). *Applied Optimal Estimation.* The M.I.T. Press, 1974.

[10] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization.* Academic Press, 1981.

[11] M.S. Grewal, V.D. Henderson, and R.S. Miyasako. Application of kalman filtering to the calibration and alignment of inertial navigation systems. *IEEE Transactions on Automatic Control*, 1991.

[12] B.W. Kernighan and D.M. Ritchie. *The C Programming Language.* Prentice Hall, 1988.

[13] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems.* Wiley-Interscience, 1972.

[14] L.S. Lasdon, S.K. Mitter, and A.D. Warren. The conjugate gradient method for optimal control problems. *IEEE Transactions on Automatic Control*, 1967.

[15] R.L. Needham. Calibration of strapdown system accelerometer dynamic errors. Master's thesis, Massachusetts Institute of Technology, 1994.

[16] C.F. O'Donnell (ed.). *Inertial Navigation Analysis and Design.* McGraw-Hill, 1964.

[17] E. Polak. An historical survey of computational methods in optimal control. *SIAM Review*, 1973.

[18] P.B. Reddy. Laser gyro strapdown system alignment/calibration and land navigation using kalman filters. *IEEE Proceedings, NAECON, Dayton*, 1980.

[19] T. Thorvaldsen and H. Musoff. Optimum selection of test table positions for strapdown system calibration. *Fifteenth Biennial Guidance Test Symposium, Holloman AFB, New Mexico*, 1991.

THESIS PROCESSING SLIP

FIXED FIELD: ill. ______ name ______

index ______ biblio ______

► COPIES: [✓ Archives] [✓ Aero] Dewey Eng Hum

Lindgren Music Rotch Science

TITLE VARIES: ► ☐

NAME VARIES: ► ☒ Joseph

IMPRINT: (COPYRIGHT) ______

► COLLATION: 116p

► ADD. DEGREE: ______ ► DEPT.: ______

SUPERVISORS: ______

NOTES:

cat'r: date:

page: ► J127

► DEPT: Aero

► YEAR: 1996 ► DEGREE: M.S.

► NAME: LINTEREUR, Louis J.