2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN APPLICATION OF EXTENDED KALMAN FILTERING TO A MODEL-BASED, SHORT-RANGE NAVIGATOR FOR AN AUV

by

Christopher A. Miller

December 1991

Thesis Advisor: Roberto Cristi

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b. OFFICE SYMBOL (If applicable) EC	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) AN APPLICATION OF EXTENDED KALMAN FILTERING TO A MODEL-BASED, SHORT-RANGE NAVIGATOR FOR AN AUV

12. PERSONAL AUTHOR(S)
MILLER, Christopher A.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM ______ TO ______	1991 December	163

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the US Government.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
AUV; Extended Kalman Filter; Nonlinear Systems; Linearization; Navigation

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Autonomous Underwater Vehicles (AUV) are being considered by the Navy for performing a variety of missions. During the research and development state of the AUV project at the Naval Postgraduate School, a navigator is needed to provide vehicle position estimates for short-range missions performed in a test pool environment. This navigator should operate with inexpensive sensors and not require excessive digital processor time. This thesis presents the results of the design of a model-based navigator. The navigator uses nonlinear vehicle models of Extended Kalman filter theory. Simulation studies for both a 12,000 pound vehicle and the 435 pound testbed vehicle, designed and built at the School (NPS AUV II), are presented. Results of using data

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL CRISTI, Roberto	22b. TELEPHONE (Include Area Code) 408-64602223; 22c. OFFICE SYMBOL EC/Cx

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 UNCLASSIFIED

19. cont.

recorded from the gyroscopes and depth cell installed in the NPS AUV II vehicle in lieu of simulated data are also discussed. These results show that the navigator meets the goals of low cost and low processor burden for short-range missions.

An Application of Extended Kalman Filtering to a Model-Based, Short-Range Navigator for an AUV

by

Christopher A. Miller
Lieutenant, United States Navy
B.S., United States Naval Academy, 1986

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1991

Author: [illegible]
Christopher A. Miller

Approved by: [illegible]
Roberto Cristi, Thesis Advisor

[illegible]
Harold A. Titus, Second Reader

Michael G. Morgan
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

Autonomous Underwater Vehicles (AUV) are being considered by the Navy for performing a variety of missions. During the research and development stage of the AUV project at the Naval Postgraduate School, a navigator is needed to provide vehicle position estimates for short-range missions performed in a test pool environment. This navigator should operate with inexpensive sensors and not require excessive digital processor time. This thesis presents the results of the design of a model-based navigator. The navigator uses nonlinear vehicle models and Extended Kalman filter theory. Simulation studies for both a 12,000 pound vehicle and the 435 pound testbed vehicle, designed and built at the School (NPS AUV II), are presented. Results of using data recorded from the gyroscopes and depth cell installed in the NPS AUV II vehicle in lieu of simulated data are also discussed. These results show that the navigator meets the goals of low cost and low processor burden for short-range missions.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to express my thanks to Professor Roberto Cristi for his help and encouragement during this project. His technical expertise and insight were invaluable to me. His patience, maturity, and understanding of human nature helped me to overcome many difficulties.

I. INTRODUCTION

A. GENERAL

The United States Navy now uses Unmanned Underwater Vehicles (UUVs) for performing a variety of missions [Ref. 1:pp. 60-88]. Currently, these vehicles are tethered and are controlled by data links to human operators. Their small size and corresponding ability to go where manned vehicles cannot go are their primary advantages. These advantages also apply to Autonomous Underwater Vehicles (AUVs). An Autonomous Underwater Vehicle is a type of UUV that is not limited by the need for local human control. The freedom from requiring an external control interface theoretically allows this type of vehicle to perform a greater range of missions than its tethered counterpart [Ref. 2:pp. 571-575].

While autonomy has clear advantages, it does require a sophisticated level of on board processing ability. The organization for the control of the vehicle can be broken into a hierarchy of levels in which the vehicle senses, thinks and acts [Ref. 3:pp. 1-3]. At any of the levels the vehicle requires an interface to its real world environment in order to function properly.

In addition to the obvious need for information concerning the physical surroundings of the vehicle, such as obstacles, data needed at any of these levels will likely include the current vehicle location. Additionally, the vehicle will need to have its present state, in terms of velocities, angle rates, and attitude available for guidance and control. An on board navigator is designed to supply this information.

B. AIM OF THE STUDY

This thesis is concerned with the navigation problem of an AUV. As a result of the method chosen for addressing this problem, this thesis is also concerned with the guidance and control functions of the vehicle since these functions are assumed to require signals fed back by the navigator.

Due to the unavailability of radio navigation aids (such as LORAN, OMEGA, or GPS) in the underwater environment, the navigation system of the AUV is primarily based on inertial measurements. For reasons of covertness, the vehicle is not expected to broadcast its position, so it cannot rely on navigational processing by a mothership and thus the navigator must be self-contained.

Generally, inertial navigation systems (INS) are complicated and historically have relied on expensive gyroscopically stabilized platforms, known as gimballed systems. A triad of accelerometers is placed on the stable platform to measure rectilinear accelerations in the platform, i.e. inertial, coordinates [Ref. 4:pp. 85-86]. Generally, the platform could be stabilized to represent any coordinate system. Operating inertial navigation systems have been built in which the platforms are stable relative to the stars, to a non-rotating earth coordinate system, and to a locally level coordinate system, among others. In any case, the accelerations are measured in the inertial coordinate system, via the stabilized platform. These measurements are then properly integrated to obtain a position estimate. [Ref. 4:pp. 193-223]

Until the last 20 years, limitations in computer speed and physical size, as well as computer memory cost have prevented designers from using strapped-down systems. A strapped-down INS is similar to a gimballed INS as described above except that the inertial reference coordinate system is stored in computer memory rather in a stable platform, and the accelerometer triad is rigidly attached to the vehicle. The motion of the vehicle relative to the chosen inertial coordinate system is determined by combining measurements from rate gyroscopes and accelerometers. The angular rate information from the gyroscopes is transformed and integrated to obtain the vehicle's attitude in the chosen coordinate system and the accelerations are integrated and transformed using this attitude information to give vehicle position in the inertial frame [Ref. 5:p. 38].

Until microprocessors were developed, the amount of computing power required to perform these operations was beyond the capability of any on board computer. However, with the advent of VLSI CMOS technology, processors are small enough and memory is inexpensive enough to make the system feasible and such systems are in use.

For any navigation system to be useful, the accuracy of the instruments has to be such that the navigator's error is within the required tolerance of the vehicle which relies on the navigation system. However, accurate sensors tend to be large and expensive. It is the aim of this thesis to study the feasibility of *combining measurements* from inexpensive instruments on board an AUV to generate relatively good position estimates over a short time interval.

To study the feasibility of operating an AUV, the Naval Postgraduate School has designed and built a testbed AUV, known as NPS AUV II. It is for this vehicle that this thesis is designed.

C. METHOD OF APPROACH

This thesis is concerned with the short-range navigation problem for the NPS testbed AUV. Because this vehicle is small it cannot carry the stabilized platform required of a gimballed INS. It must therefore rely on a strapped-down system. Currently however, there is no commercially available strapped-down system which can be incorporated into the AUV. It is therefore necessary to design a specific navigator for the AUV.

Being limited in complexity and cost, the navigator has to rely on previously purchased and installed instruments. Also, because the system need only operate over short ranges the effects of the earth's orbit and rotation can be neglected. This approximation also has the effect of simplifying the navigator.

The approach taken in this thesis to solve the navigation problem is to use a nonlinear dynamic model of the submerged vehicle to filter gyroscope and accelerometer readings to estimate the vehicle's position as well as the vehicle's attitude, velocity and rotational rates. This approach is diagramed in Figure 1.1.

The thesis is presented in five parts. Chapter II discusses some background and the theory behind the navigator design. Chapter III discusses the details of the design and the techniques used in simulation. In Chapter IV we present the results obtained from the simulation study, while Chapter V shows the results obtained using recorded sensor readings taken from the Naval Postgraduate School's vehicle. Finally, Chapter VI summarizes the the results of this research and contains conclusions and recommendations for application and further study.

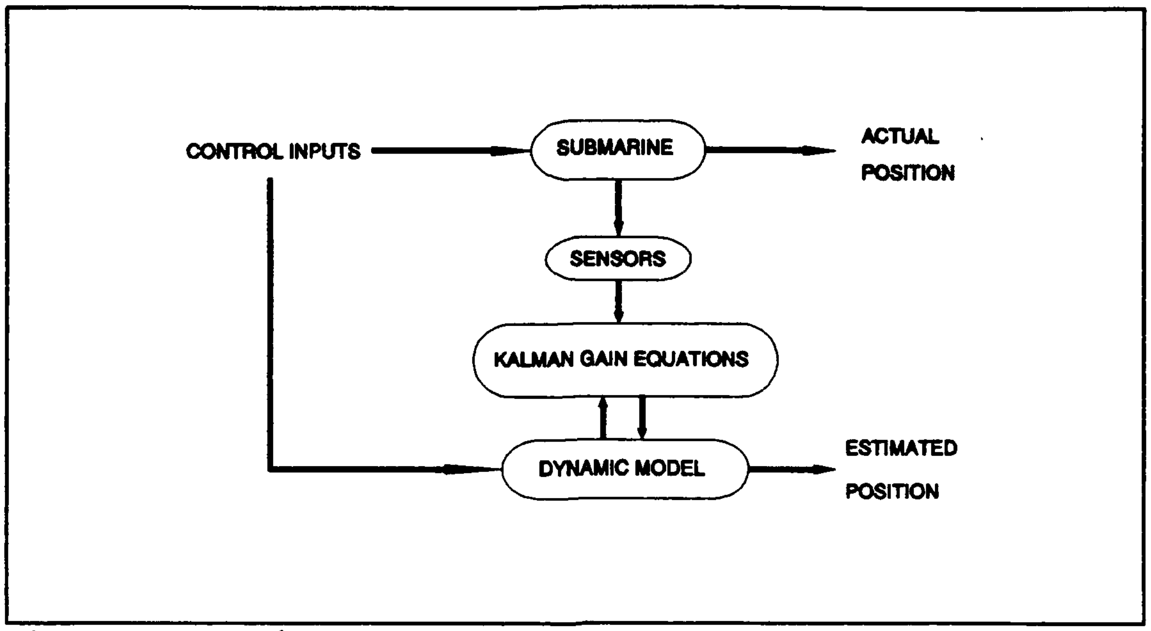

Figure 1.1 Navigator Concept

II. LINEARIZATION AND EXTENDED KALMAN FILTERING

A. GENERAL

This chapter discusses the general theory used to design the AUV navigators. A brief introductory presentation is given for each of the concepts used in the development of this work. None of the concepts is presented in an exhaustive manner and derivations are not provided. A prior understanding of these concepts is assumed; therefore, only a brief review is presented for clarity. For more detailed explanations and developments, the reader is directed to the References listed herein.

B. LINEAR MODELING

In control theory we encounter the problem of representing a dynamic physical system mathematically. A common way of doing this is to use a state-space model by breaking the Nth- order differential equation representing the system into N, coupled, first-order differential equations. These equations can be easily represented using matrix algebra and matrix notation as long as the system being modeled is linear. A linear system exhibits the properties of both homogeneity and superposition [Ref. 6:pp. 14-25].

A linear, time-invariant (LTI), state-space representation of a continuous-time system is given by

$$\dot{\underline{x}}(t) = \boldsymbol{A}\underline{x}(t) + \boldsymbol{B}\underline{u}(t) \tag{2.1}$$

where x is the system state vector, u is the input vector and **A** and ***B*** are the state transition and input matrices respectively.

In the time-invariant case, both x and u are functions of time, while **A** and ***B*** are not. If ***A*** and ***B*** were functions of time, then the system would be classified as time-varying.

Generally, this description is supplemented with a measurement equation which represents the output of the system as a function of the states and the inputs. Again, if the system is linear, matrix notation may be used, and the measurement is given by

$$\underline{y}(t) = \boldsymbol{C}\underline{x}(t) + \boldsymbol{D}\underline{u}(t) \tag{2.2}$$

where $\underline{y}$ is the system output. Generally, there is more than one output so $\underline{y}$ is a vector.

A system can be represented in discrete time with analogous equations. These are:

$$\underline{x}(k+1) = \mathbf{\Phi}\underline{x}(k) + \mathbf{\Gamma}\underline{u}(k) \tag{2.3}$$

$$\underline{y}(k) = \boldsymbol{C}\underline{x}(k) + \boldsymbol{D}\underline{u}(k) \tag{2.4}$$

where k is the time index.

Knowledge of the system state at all times is often required to control the system. However, it may not be feasible to measure all of the states of a system because systems can be complex and putting instrumentation in place to measure the states may not be possible. In such cases, a filter known as an observer is used. [Ref. 6:p. 259]

An observer is a dynamic subsystem based on a model of the system being observed which recursively predicts the system state. The error between the output of the observer and the output of the system is driven to zero by correcting the state prediction with the output error signal [Ref 6:p. 262]. This relationship is given by

$$\hat{\underline{x}}(k+1) = \mathbf{\Phi}\hat{\underline{x}}(k) + \mathbf{\Gamma}\underline{u}(k) + \boldsymbol{G}(\underline{y}(k) - \boldsymbol{C}\hat{\underline{x}}(k)) \tag{2.5}$$

where $\boldsymbol{G}$ is the observer feedback gain, and $\hat{x}$ is the observer, or estimated state.

The result is that an estimate of the system state is made available without directly measuring all the states. Because this thesis is written using discrete-time models and filters, this equation is given in its discrete-time form.

In the presence of noise, which may include unknown disturbances perturbing the system dynamics, known as plant noise, or noise in the instruments which measure the output of the system, known as measurement noise, the observer feedback gain becomes an important factor in minimizing the sensitivity of the overall system to the noise disturbances. The Kalman filter is an observer which provides optimal state estimates for linear systems in the presence of noise. The Kalman filter is based on the assumption that the plant noise and the measurement noise are independent, white, Gaussian processes, so the system equations become

$$\begin{aligned} \underline{x}(k+1) &= \boldsymbol{\Phi}\underline{x}(k) + \boldsymbol{\Gamma}_1\underline{u}(k) + \boldsymbol{\Gamma}_2\underline{w}(k) \\ \underline{y}(k) &= \boldsymbol{C}\underline{x}(k) + \boldsymbol{D}\underline{u}(k) + \underline{v}(k) \end{aligned} \tag{2.6}$$

where $\underline{w}$ is the plant noise vector, and $\underline{v}$ is the measurement noise vector.

The Kalman filter gain matrix is time-varying and is related to the plant noise and the measurement noise through their respective covariance matrices, $\boldsymbol{Q}$ and $\boldsymbol{R}$, by

$$\begin{aligned} \boldsymbol{P}(k+1|k) &= \boldsymbol{\Phi}\boldsymbol{P}(k)\boldsymbol{\Phi}^T + \boldsymbol{\Gamma}_2\boldsymbol{Q}\boldsymbol{\Gamma}_2^T \\ \boldsymbol{G}(k+1) &= \boldsymbol{P}(k+1|k)\boldsymbol{C}^T[\boldsymbol{C}\boldsymbol{P}(k+1|k)\boldsymbol{C}^T + \boldsymbol{R}]^{-1} \\ \boldsymbol{P}(k+1|k+1) &= [\boldsymbol{I} - \boldsymbol{G}(k+1)\boldsymbol{C}]\boldsymbol{P}(k+1|k) \end{aligned} \tag{2.7}$$

where $\boldsymbol{P}$ is the variance matrix of the estimated state. [Ref. 6:pp. 411-417]

With these gain equations, the filter equations are:

$$\begin{aligned} \hat{\underline{x}}(k+1|k) &= \mathbf{\Phi}\hat{\underline{x}}(k|k) + \mathbf{\Gamma}\underline{u}(k) \\ \hat{\underline{y}} &= \mathbf{C}\hat{\underline{x}}(k+1|k) \\ \hat{\underline{x}}(k+1|k+1) &= \hat{\underline{x}}(k+1|k) + \mathbf{G}(k+1)[\underline{y}(k+1) - \hat{\underline{y}}(k+1)]. \end{aligned} \tag{2.8}$$

This is known as the predictor-corrector form of the Kalman filter in which the next state is predicted using the state space model and the measurement is used to correct the prediction. [Ref. 7:p. 3]

The $\mathbf{Q}$ matrix describes the unknown noise that disturbs the system, while the $\mathbf{\Gamma}_2$ matrix describes the way this noise enters the system. If $\mathbf{Q}$ is diagonal then the disturbances are assumed to be independent, otherwise they are correlated as described by the off-diagonal terms in $\mathbf{Q}$. If $\mathbf{Q}$ has large values, then large deviations from the systems's predicted state will be expected and more reliance will be given to the measurements for those states which correspond to the terms in $\mathbf{Q}$ which are large.

The $\mathbf{R}$ matrix describes the unknown noise that disturbs the sensors which return the measurements. The Kalman filter considers the measurement noise to enter all the individual measurements so there is no $\mathbf{D}_2$ matrix. As with the measurement noise covariance matrix, if $\mathbf{R}$ is large, then the measurements are expected to deviate more from the states being measured, and the Kalman filter will rely more on the predicted state than on the measurements. [Ref. 8:pp. 127-132]

The initial ***P*** matrix affects the reliance the Kalman filter has on the initial conditions. Large values in ***P(0)*** mean that the filter will not rely on the initial conditions, but will instead give more weight to the measurements. This allows the estimated state to change rapidly as the filter goes through its transient stage. In steady state, however, ***P(k)*** has no effect on the Kalman filter because it approaches a constant value that is dependent only on the system and the noise covariance matrices.

Because the gain equations do not depend directly on time or on the state trajectory in a LTI system, the gain can be calculated a priori and recalled from memory as needed. There is no need for real-time gain computation. Moreover, the gain matrix approaches a steady-state value which is determined by the system equations and the ***Q*** and ***R*** matrices through the associated Riccati equation. In many cases, using the steady- state gain matrix instead of the time-varying gain matrix gives satisfactory results in a reduced-complexity algorithm which does not take into account prior knowledge of the initial conditions. [Ref. 8:pp. 238-244]

C. LINEARIZATION

Thus far the discussion has been limited to linear systems; however, many systems in which the control engineer is interested are nonlinear. Nonlinear system models are more general than linear models and can contain a wide of variety of nonlinear characteristics for which limited analytic tools exist. In general, nonlinear systems do not exhibit the properties of homogeneity or superposition, and may include transcendental, trigonometric or other nonlinear functions [Ref. 9:pp. 351-353]. Such is the case with the model chosen for this thesis which will be discussed in Chapter III.

A method for working with nonlinear systems is to linearize them using a truncated Taylor series approximation. The Taylor series approximates a function around a given point as an infinite sum of weighted, analytically determined, partial derivatives. A general Taylor series around the point x_0 is given by

$$f(x) = \sum_{n=0}^{\infty} \frac{d^n f(x_0)}{n!\,dx^n}(x-x_0)^n. \qquad \textbf{(2.9)}$$

A truncated Taylor series only uses a few of the terms of the sum. In order to realize a linear function from the Taylor series expansion of a nonlinear one, the series must be truncated at the $n = 1$ term. This method is general and can be applied to any nonlinear system, and it will be valid in a region surrounding the linearization point. [Ref. 10:p. 3]

For the case of a discrete-time dynamic system represented in state space, which, in general, is not a function of a single variable, but is rather a function of time, the input vector, and the past state vector, the Taylor series is defined over the partial derivatives of each of the independent variables. In the case of a time-invariant system, the series is expanded about an operating point described by the state, $\underline{x}_0$, and a corresponding input, $\underline{u}_0$. A nonlinear state space equation can then be approximated as

$$\dot{\underline{x}} = \mathbf{f}(\underline{x}_0, \underline{u}_0) + \frac{\partial \mathbf{f}(\underline{x}_0, \underline{u}_0)}{\partial \underline{x}}(\underline{x} - \underline{x}_0) + \frac{\partial \mathbf{f}(\underline{x}_0, \underline{u}_0)}{\partial \underline{u}}(\underline{u} - \underline{u}_0) \quad \textbf{(2.10)}$$

where **f** is the nonlinear system function. This notation implies that the partial derivatives are constant terms, calculated analytically and evaluated at $\underline{x}_0$, and $\underline{u}_0$.

The state, $\underline{x}_0$, and input, $\underline{u}_0$, around which the system is linearized, might not be constant. Most of the time, the system is linearized around the trajectories $\underline{x}_0(t)$, and $\underline{u}_0(t)$. Because

$$\dot{\underline{x}}_0(t) = \mathbf{f}(\underline{x}_0(t), \underline{u}_0(t)) \quad \textbf{(2.11)}$$

Equation 2.10 can be rewritten as

$$\Delta\dot{\underline{x}}(t) = \frac{\partial \mathbf{f}(\underline{x}_0,\underline{u}_0)}{\partial \underline{x}}\Delta\underline{x}(t) + \frac{\partial \mathbf{f}(\underline{x}_0,\underline{u}_0)}{\partial \underline{u}}\Delta\underline{u}(t) \qquad \mathbf{(2.12)}$$

where

$$\Delta\underline{x}(t) = \underline{x}(t) - \underline{x}_0(t)$$

$$\Delta\underline{u}(t) = \underline{u}(t) - \underline{u}_0(t)$$

so that the partial derivatives which form the right-hand side of Equation 2.12 are analogous to the **A** and ***B*** matrices of Equation 2.1.

Just as a nonlinear system equation can be expanded using a Taylor series, a nonlinear measurement equation can also be expanded *and linearized. The formulation* for this expansion is similar to the above expression except that the nonlinear measurement equation is used to generate the analogs of the ***C*** and ***D*** matrices.

D. EXTENDED KALMAN FILTER

The Kalman filter is derived for linear systems with linear measurement equations; however, using the linearization techniques described in the last section, this filter can find application to general, nonlinear, systems. This is known as the Extended Kalman Filter. The Extended Kalman filter is suboptimal and may suffer convergence and stability problems, but it has been shown to be useful in a variety of applications. [Ref 8:p. 189]

The form of the Extended Kalman Filter equations are essentially similar to those of the linear Kalman filter. The nonlinear model is used to predict the state and the nonlinear measurement is used to correct the prediction. The most significant difference between the Extended Kalman Filter and the linear Kalman filter is in the gain equations. Rather than using the state transition, the input, and the measurement matrices to calculate the gain, the Extended Kalman filter uses the linearized model of the nonlinear system. It uses the partial derivatives of the nonlinear state equations and the nonlinear measurement equations. These partials are evaluated at each estimated state. As such, the gain, $\boldsymbol{K}$, cannot be computed in advance because it is dependent on both the state trajectory and the input history.

Calculating partial derivatives of the nonlinear system and measurement equations for each measurement as well as calculating the Kalman filter gain matrix is a computational burden. In order to overcome this problem, it may be possible to use gain matrices calculated in advance by choosing discrete points in the system's state space about which to linearize the nonlinear system. Practically, only some of the system states are varied while others are kept constant. Using the chosen points, several linear approximations to the nonlinear system are calculated and then used to calculate the steady-state Kalman filter gain matrix associated with those particular points. Once the gains are calculated, the Extended Kalman filter is implemented by determining which of the chosen linearization points is closest to the current estimated state and the corresponding gain matrix is used in the Extended Kalman filter equation given as the last of Equations 2.8. [Ref 8:p. 189]

In using the Extended Kalman filter, the nonlinearities are modeled as plant noise. This means that the Γ_2 matrix is usually the identity matrix. Furthermore, because the nonlinear effects are not included in the gain equations, the Extended *Kalman filter can* be very sensitive to the Q and R matrices. Choosing improper values can make an Extended Kalman filter unstable. The values chosen for these matrices should not necessarily correspond to the actual noise expected in the system or in the measurements, but rather they must made large enough to provide a robust prediction in spite of the nonlinear effects.

III. INTEGRATED NAVIGATOR DESIGN

A. GENERAL

The design techniques used in this thesis are presented in this chapter. A brief description of the models used in the designs is given, as well as a summary of changes made to the models to facilitate their use in the navigator. An explanation of the measurements used to drive the navigator is also given. Finally, a description of the simulation environment, including an introduction to the use of MATLAB MEX files, as well as an outline of the navigator's program structure, is presented.

B. USE OF NONLINEAR MEASUREMENT MODELS

One of goals of this study has been the integration of nonlinear measurement models with the Extended Kalman filter navigator. The nonlinear measurements provide additional information about the vehicle's state which improve the accuracy of the filter.

In order to obtain more information about the vehicle's velocity, depth rate is measured. Depth rate is related to both the pitch angle of the vehicle and its velocity as illustrated in Figure 3.1. This diagram shows only two dimensions for clarity; however, a discussion of the full, three-dimensional depth-rate equation follows in Section D.

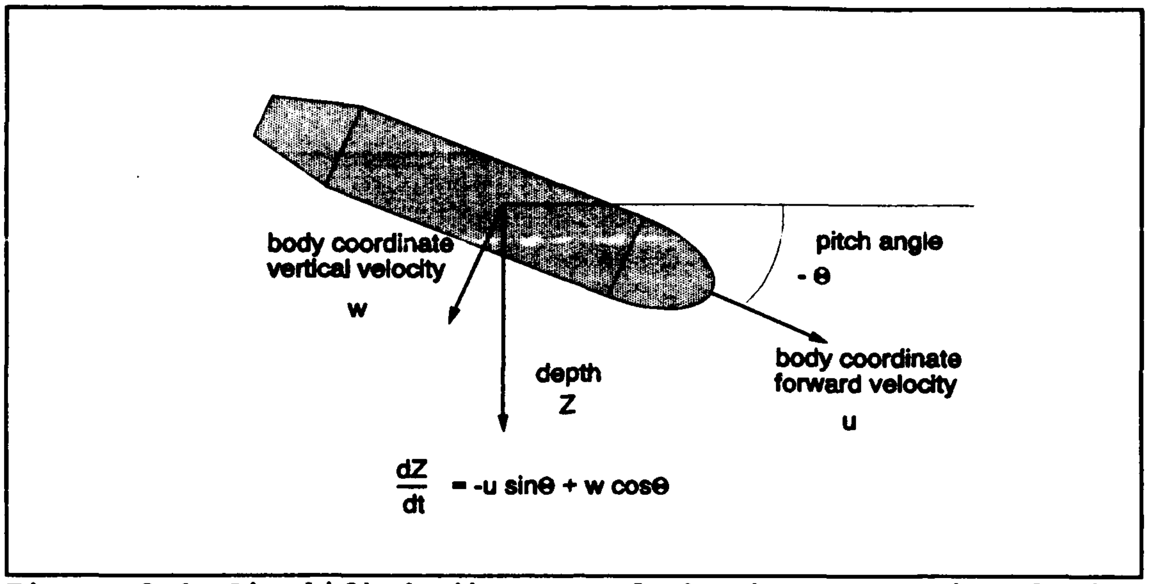
body coordinate vertical velocity
w
pitch angle
- θ
depth
Z
body coordinate forward velocity
u
$\frac{dZ}{dt} = -u \sin\theta + w \cos\theta$

Figure 3.1 Simplified diagram of depth rate and velocity relationship

The other nonlinear measurement model used in the navigator is for the accelerometers. Because accelerometers cannot distinguish between accelerations and gravitational forces, they can be used to measure gravity. In this case, they are used to provide a nonlinear measurement of the gravity vector in the vehicle coordinate system. This can be interpreted as a nonlinear measurement of the vehicle's attitude in earth coordinates. A simplified diagram of this concept is shown in Figure 3.2. A more detailed description of this measurement is given in Section D.

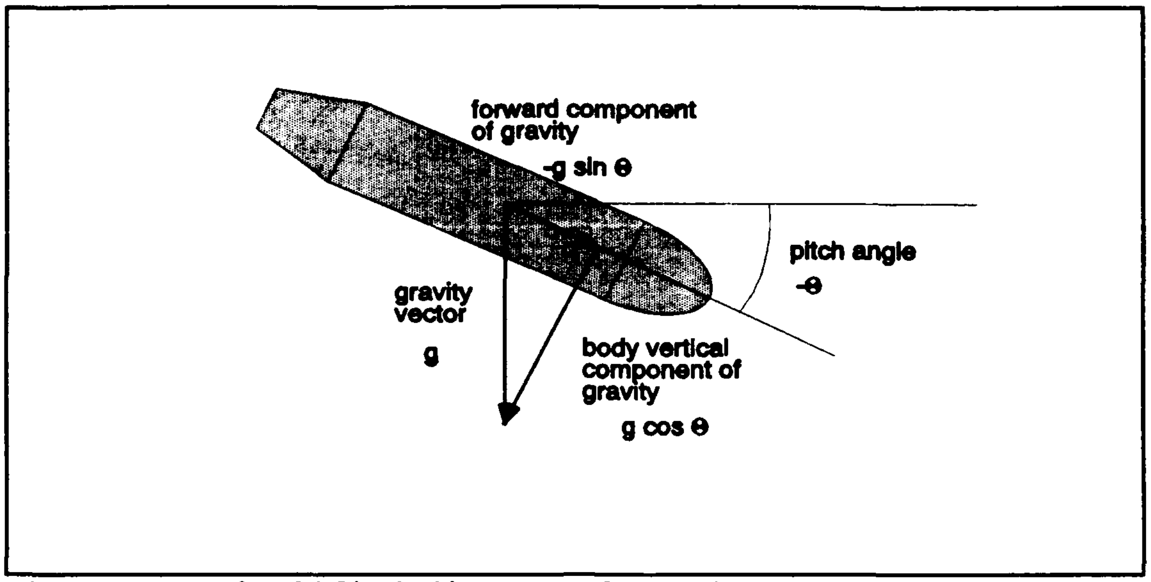
forward component of gravity
-g sin Θ
pitch angle
-Θ
gravity vector
g
body vertical component of gravity
g cos Θ

Figure 3.2 Simplified diagram of accelerometer measurement

C. VEHICLE MODELS

1. The Vehicles

Two vehicles are studied in this thesis. One is a 17.4 foot long, 12,000 pound Swimmer Delivery Vehicle (SDV), and the other is the 7 foot long, 435 pound testbed AUV designed and built at the Naval Postgraduate School known as the NPS AUV II. The SDV model was used in the preliminary stages of this work because an accurate model of the NPS AUV II was not yet available.

The two vehicles are geometrically similar but there are minor differences. They both have a rectangular cross section rather than the usual body of revolution more typical of submarine vehicles. The SDV differs from the NPS AUV II in that it has a deep keel in which a third propeller is housed for surface operation. Although the model has provision for this feature, it is not used in the simulations. The NPS AUV II differs from the SDV in that it has a bow rudder which increases its maneuverability. This feature is included in the model and is used in the simulations.

2. The Models

The models chosen for this work are based on modified equations of motion for submarine vehicles developed by Gertler and Hagen [Ref. 11]. As opposed to a typical inertial navigation system (INS), these models are representations of the vehicle dynamics rather than models of the sensors and of coordinate system relationships to inertial space. As such, no provision is made for the earth's curvature, its rotation, its orbit around the sun, or gravitational anomalies. This limits the applicability of the models to short-range missions.

Both models are 12-state, six-degree-of-freedom, nonlinear models which are based on three specific force equations which govern linear accelerations in body coordinates, three specific torque equations which govern angular accelerations in body coordinates, and six kinematic relationships which translate linear and angular velocities from body fixed to inertial coordinates. The specific equations of motion are given in the NCSC report by Crane, Sumney and Smith [Ref. 12]. The two coordinate systems used by the models are typical of those used to describe aircraft motion in that they are both right-handed with the vertical axis pointing down. In the reference system, down is in the direction of the gravity vector, and for the body coordinate system, down is through the bottom of the vehicle. These systems are shown in Figure 3.3.

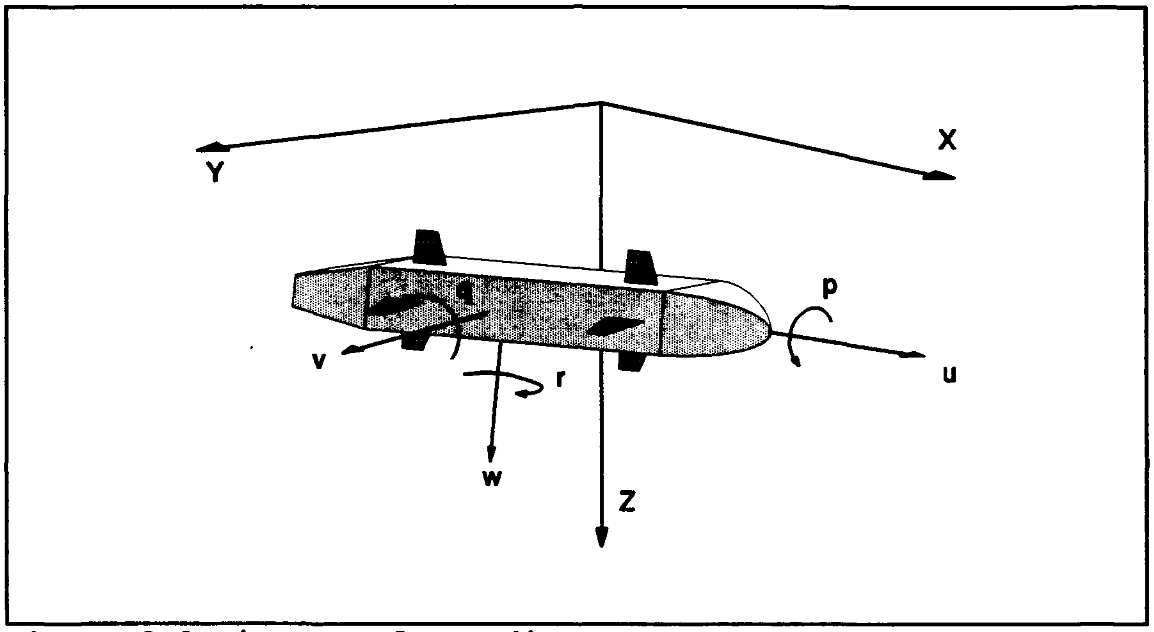

Figure 3.3 Diagram of coordinate systems

a. *Vehicle State*

As previously stated, these models have a 12-element state vector given by

$$\underline{X} = [u\ v\ w\ p\ q\ r\ X\ Y\ Z\ \Phi\ \Theta\ \Psi]^T. \tag{3.1}$$

The first three elements of the state vector, *u*, *v*, and *w*, are the 3 mutually orthogonal velocities in vehicle, or body coordinates. They are given in feet per second.

The next three elements, *p*, *q*, and *r* are the three angular rates associated with vehicle motion around the vehicle coordinate axes. They are in units of radians per second. The directions for these six states are shown in Figure 3.1.

X, Y and Z are the coordinates of the vehicle's location in inertial space, and Φ, Θ, and Ψ are the Euler angles which describe the vehicle's attitude in inertial space. Φ is the roll angle; Θ is the elevation angle; Ψ is the azimuth angle.

The transformation from body coordinate system velocities to inertial coordinate system velocities is given by

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} = \begin{bmatrix} c\Psi c\Theta & c\Psi s\Theta s\Phi - s\Psi c\Phi & c\Psi s\Theta c\Phi + s\Psi s\Phi \\ s\Psi c\Theta & c\Psi c\Phi + s\Psi s\Theta s\Psi & -c\Psi s\Phi + s\Psi s\Theta c\Phi \\ -s\Theta & c\Theta s\Phi & c\Theta c\Phi \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \qquad (3.2)$$

where *c* represents the cosine function and *s* represents the sine function. Φ, Θ, and Ψ are the vehicle Euler angles described above. This transformation matrix is orthogonal so its inverse is equal to its transpose. [Ref. 13:p. 115]

The transformation from body angle rates to Euler angle rates is given by

$$\begin{bmatrix}\dot{\Phi}\\ \dot{\Theta}\\ \dot{\Psi}\end{bmatrix} = \begin{bmatrix}1 & \tan\Theta\sin\Phi & \tan\Theta\cos\Phi\\ 0 & \cos\Phi & -\sin\Phi\\ 0 & \sin\Phi/\cos\Phi & \cos\Phi/\cos\Theta\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \qquad (3.3)$$

where the variables are as defined above. [Ref. 14:p. 12]

b. Inputs

The vehicle control inputs for both of the models are similar. Rudder angle, dive plane angle and engine rpm are the inputs. Angles are measured in radians. Because both vehicles have stern dive planes and bow dive planes which can operate independently, there is a separate input for each. The NPS AUV II also has bow and stern rudders which can operate independently so that model has provision for a separate input for each.

c. Differences in the Models

There are a few other differences between the SDV model and the NPS AUV II model. The main differences are in the hydrodynamic coefficients and the mass matrices of the two models. The hydrodynamic coefficients describe the effect that the vehicle velocity and angular velocity (in three-dimensional space) have on the hydrodynamic forces that act on the vehicle. The mass matrix is a convenient way to gather terms in the vehicle equations of motion which multiply linear and angular accelerations so as to facilitate a state-space formulation of the model. The mass matrix is made up of coefficients which describe vehicle's mass moment of inertia tensor, as well as the coefficients which describe the coupling between the linear and angular accelerations and the forces acting on the vehicle. The coefficients for both vehicles have been previously determined [Ref. 12][Ref. 15].

The propulsion models are also different. Both models utilize square-law thrust and drag relationships, as well as cross flow force and torque calculation. The SDV model uses a four term Simpson's Rule integration to calculate these forces. In fact, only two terms are calculated. One which corresponds to the cross flow force and torque in the vertical plane, NORPIT, and one which corresponds to the cross flow force and torque in the horizontal plane, LATYAW. The NPS AUV II model uses a 15 term trapezoidal numerical integration scheme to calculate these cross flow forces and torques. This model also distinguishes between the forces and the torques in that four separate terms are calculated. The propulsion model for the AUV is deemed to be the most accurate of the two.

3. Computational Aspects of the Equations

To facilitate their use in the navigator, some minor changes have been made to both models. The changes were necessary because of the way the C programming language handles floating point numerical operations. Logic statements have been added to ensure that the tangent and sine functions return a value of zero when passed a value of zero. This change makes the C programs behave more like their corresponding MATLAB functions.

Another change was to take the absolute value of all floating point numbers prior to performing any square root operation. This prevents square root domain errors.

D. MEASUREMENTS

In addition to the vehicle model, the simulations use a nonlinear measurement model. The instruments which are used to generate the measurements are rate gyroscopes, the depth cell, a triad of accelerometers and the heading gyroscope. The depth cell reading is also used to generate a depth-rate estimate by simple first order difference equation. The measurements from the rate gyros and the depth cell are essentially linear functions of the vehicle state; however, the other measurements are related to the state nonlinearly.

1. Depth Rate

Depth rate is approximated as the difference between two successive depth measurements divided by the sampling interval. Analytically, however, depth rate is related to the vehicle's orientation and its three-dimensional velocity vector. The equation which describes this relationship is one of the nonlinear functions within the vehicle models, and is given by

$$\dot{z} = -u\sin\Theta + v\cos\Theta\sin\Phi + w\cos\Theta\cos\Phi \qquad (3.4)$$

which is identical to the bottom equation of Equation 3.2. By taking the partial derivatives of this function with respect to the state, the associated portion of *C* matrix to be used with the Extended Kalman filter is generated.

2. Accelerometers

The measurement associated with the accelerometers is also a nonlinear function of the state. The accelerations of the vehicle fall within the dead zone of the accelerometers which were purchased for use in the NPS AUV II, and as such, these accelerometers could not be used in the normal sense. However, because an accelerometer cannot distinguish between accelerations and gravitational forces, the accelerometer triad would give the decomposition of the gravitational acceleration vector in vehicle coordinates. So while gravity remains constant, the accelerometer measurements would change depending on the attitude of the vehicle. The relationship between the accelerometer readings and the vehicle orientation is nonlinear, and is given by

$$\begin{bmatrix} a_u \\ a_v \\ a_w \end{bmatrix} = -\boldsymbol{R} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \qquad (3.5)$$

where ***R*** is the inverse (transpose) of the transformation matrix given in Equation 3.2, and g is the magnitude of the gravitational acceleration.

Additionally, because these accelerometers have a dead zone, it is likely that there will be times when they do not generate a signal. This dead zone is an additional nonlinearity which can be taken into account in the Extended Kalman filter by setting to zero the row of the linearized ***C*** matrix that is associated with whichever accelerometer is operating in its dead zone. The model used for the accelerometer dead zone is given in Figure 3.4.

3. Gyroscopes

While the measurements from the rate gyroscopes are linear functions of the state vector, that is, they should be measurements of p, q, and r, it is known that the gyroscopes installed in the NPS AUV II generate a bias term that is drifting time. For this reason, the vehicle model has been augmented to include three more states, namely, the rate- gyroscope biases. Carried as states, the rate gyroscope measurement can then be modeled as the vector sum of the three angular rate vehicle states, p, q, and r, and their corresponding bias terms. The Extended Kalman filter

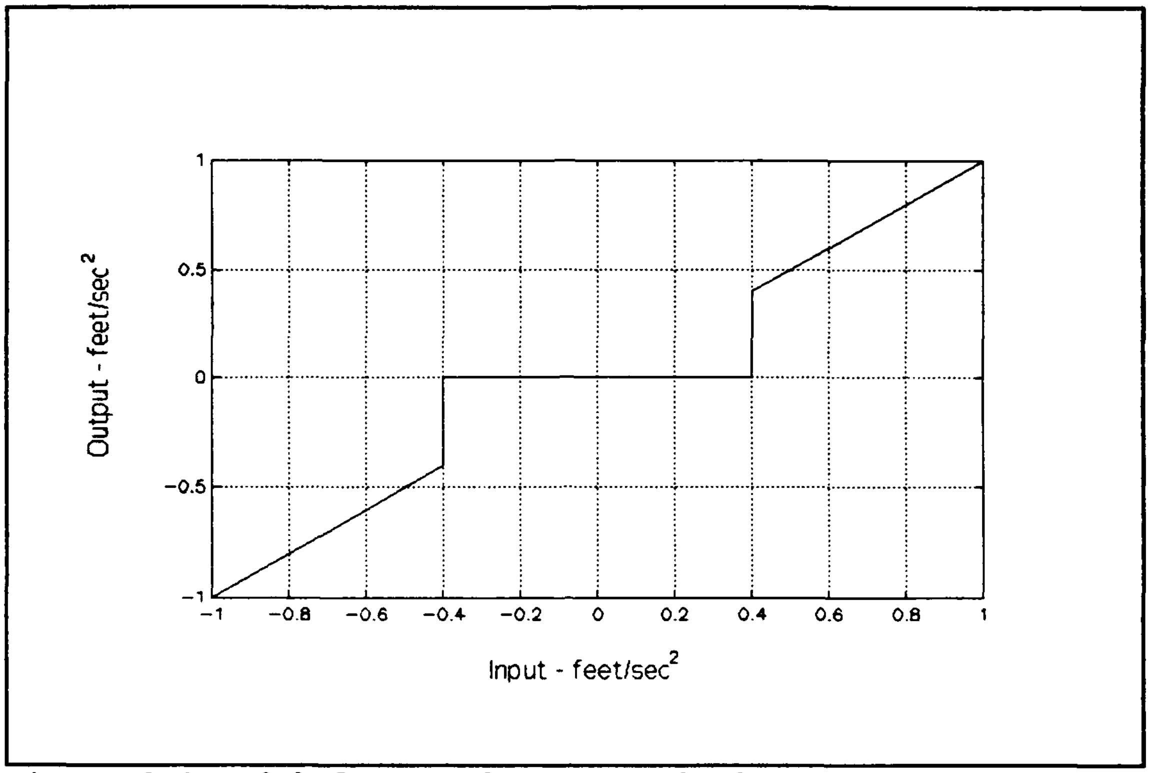

Figure 3.4 Model for accelerometer dead zone

is then able to estimate the bias terms and remove their effect from the estimation of the vehicle state.

In short-range problems, the heading gyroscope does not suffer from the bias problems that plague the rate gyroscopes. It has a clean signal that does not drift with time, and it has a magnetic flux gate sensor which helps it to maintain inertial alignment.

The NPS AUV II also has a vertical gyroscope which measures the vehicle roll and pitch angles which are used in lieu of the accelerometer measurements described above. The measurements from this unit suffer from two problems. The first is quantization noise. The roll measurements are so small relative to quantization levels in the analog-to-digital converter that the signal is lost in quantization noise. The second problem is with the pitch measurement, which is known to have a bias. As with the rate gyroscopes, the Extended Kalman filter for the vehicle can be augmented to include the pitch measurement bias which is modeled as a relatively constant value which is affected by a fictitious noise term.

E. MATLAB MEX FILE GENERATION

The programs developed for this thesis are written for MATLAB, an interactive application designed to use matrices as the basic computational entity. The specific version that is used is AT-MATLAB Version 3.5k for IBM compatible personal computers.

MATLAB provides an interpretive programming environment which is easy to use and easy to debug in the form of Script files. A Script file allows a user to write MATLAB commands to an ASCII file. When the file is invoked, MATLAB reads the file and performs the operations listed therein. In this context, MATLAB functions as an interpreted language.

MATLAB also has facilities for using Functions. A Function is like a Script file but it contains the reserved word, Function. A Function is different from a Script file in that the commands are interpreted, compiled and stored in RAM, ready for further uses. The advantage of using a

Function is that once used, the Function will run faster on future calls than an identical Script file because it is compiled. [Ref. 16:p. 2-86]

MATLAB also allows the use of specially written FORTRAN and C programs to be called from within the MATLAB environment as though they were either MATLAB Script files or Functions. Files which are written using this feature are called MEX files. The two largest subroutines written for this thesis are in the form of MEX files. They were written in C, and then compiled, and linked with MATLAB supplied libraries into executable code. This executable code was then operated on by a MATLAB supplied program to convert in to a MEX file. The only advantage to using a MEX file is speed of execution. A MEX file will typically run 25 times faster than its Script or Function file counterpart. [Ref. 16:p. 1-47]

The procedures for using this feature are contained in the MATLAB User's Guide. However, what is not explained in the manual is that the compilation and linking must be done from within the MEX subdirectory (where MATLAB is the parent directory). If this is not done, the linker will give "undefined function" errors for any Structures or Procedures defined within the MATLAB MEX libraries which are called by the C program.

There is a significant difference between the way MATLAB and FORTRAN store two-dimensional arrays and the way that C stores them which must be accounted for when using C language MEX files. Both MATLAB and FORTRAN store arrays in a column by column format in contiguous long-words of memory, whereas C uses a row by row scheme. This is not a difficult problem to overcome, but it forces the C programmer to either use transposed matrices in his code or to use a separate conversion routine before passing matrices into or out of MATLAB. The former approach is more compact but the latter approach makes the source code more understandable.

F. PROGRAM STRUCTURE

The programs used in this thesis are listed in Appendix A and Appendix B. For the SDV, the programs are SIMUL.M, MODEL.C, MKABMEX.C, GETMEAS.M, and MAKMEAS.M. For the NPS AUVII, the programs are AUVSIM.M, AUV2.C, AUV2AB.C, GETMEAS.M, MAKMEAS.M, MAKEK.M, and GETK.M. The names of the source code files for different versions of the programs are appended with the version number, for example, SIMUL6.M, MAKMEAS4.M.

The structure for both sets of programs is the same. Both SIMUL.M and AUVSIM.M are the main modules of the navigator. They initialize the filter parameters, the simulated vehicle state, the Extended Kalman filter state, the control input, and the ***A***, ***B*** and ***C*** matrices of the linearized model. In the case of the versions of AUVSIM.M which use data recorded from the actual NPS AUV II vehicle, the programs read the data file as a standard MATLAB matrix instead of running a separate vehicle model to generate simulated measurements.

The main loop in these programs is diagrammed in Figure 3.5. First, the input command is generated and formatted. Next, the linearized model ***A***, ***B*** and ***C*** matrices are recalculated around the last estimated state. These matrices and the error variance matrix are passed to KALM.M which calculates the next Extended Kalman filter gain matrix, ***K***, using the formulae described in Chapter II. The program then either calls MODEL.MEX or AUV2.MEX to generate the next simulated vehicle state. A call is made to GETMEAS.M to extract the simulated gyroscope and accelerometer readings from the vehicle state. GETMEAS.M uses the MATLAB RAND function to simulate noise in the sensor readings by adding a normally distributed, zero-mean, pseudo-random number to

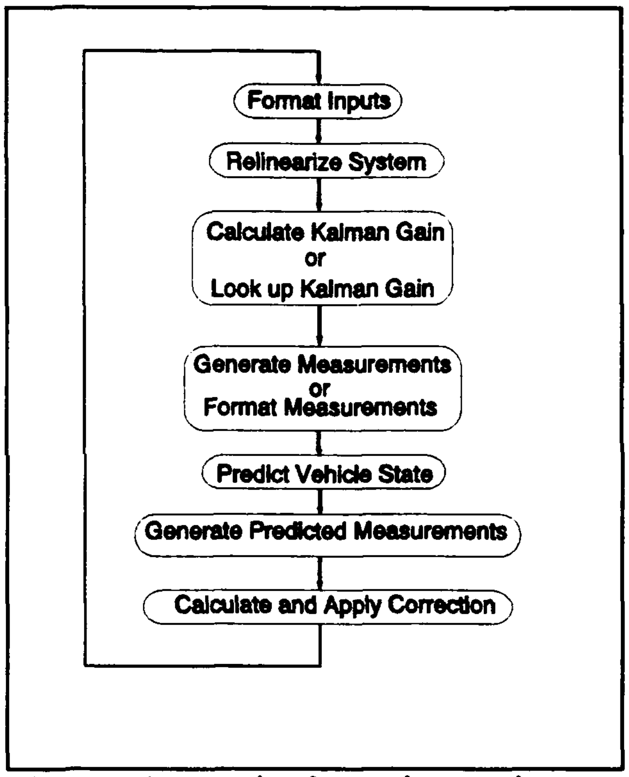
Format Inputs
Relinearize System
Calculate Kalman Gain or Look up Kalman Gain
Generate Measurements or Format Measurements
Predict Vehicle State
Generate Predicted Measurements
Calculate and Apply Correction

Figure 3.5 Main loop in navigator

the simulated measurement. The variance of this pseudo-random number is equal to the apparent variance of the actual sensor signal. If recorded data is used, then a version of GETMEAS.M is used to format the recorded data for use by the Extended Kalman filter and noise is not added.

The Extended Kalman filter is implemented in the predictor-corrector form. Either MODEL.MEX or AUV2.MEX is used to predict the next vehicle state from the last estimate and the given inputs. The measurement equation is applied to this prediction to form the estimated measurement. This estimate and the actual measurement are applied to the predicted state as described in the last chapter to correct prediction. This corrected prediction becomes the estimated state for the next iteration of the filter.

The sequence is somewhat different for that version of AUVSIM which uses the piecewise constant K matrix. MAKEK.M is called outside the main loop of the program to calculate the steady-state K matrices for several different values of u and Θ. Inside the loop, GETK.M is used to compare the current estimates of u and Θ to the values used to make the several K matrices and then return the K matrix which corresponds most closely to the estimated state.

IV. SIMULATION RESULTS

A. GENERAL

This chapter describes the results of the navigator simulations for the SDV and the NPS AUV II. Both navigators were driven by simulated measurements generated by nonlinear models.

The effectiveness of including a model of the bias in the rate gyroscopes and the use of accelerometer measurements to determine vehicle attitude are explored in this chapter. Also, the feasibility of using depth rate, as discussed in the Chapter III, to estimate forward speed is addressed, and the effect of filter parameters on the estimated vehicle states is discussed.

B. SWIMMER DELIVERY VEHICLE SIMULATION

Figures 4.1 through 4.14 show the simulated vehicle states and the estimated states, as determined by the navigator, for the SDV. For the particular simulation run from which this data is taken, the vehicle was given an initial forward speed, *u*, of 0.3 feet per second, a constant propeller speed of 650 rpm, a constant rudder command of 0.2 radians, and a sinusoidal dive plane command. A sampling interval of one second has been used. Also, an initial pseudo-random Gaussian error vector was added to the initial vehicle states to simulate an unknown initial condition.

The remainder of this section presents the details of the simulated responses and attempts to explain the behavior of the signals.

Figure 4.1 shows the X-Y position of the vehicle and the navigator. The two trajectories are not coincident indicating that the navigator does not generate a highly

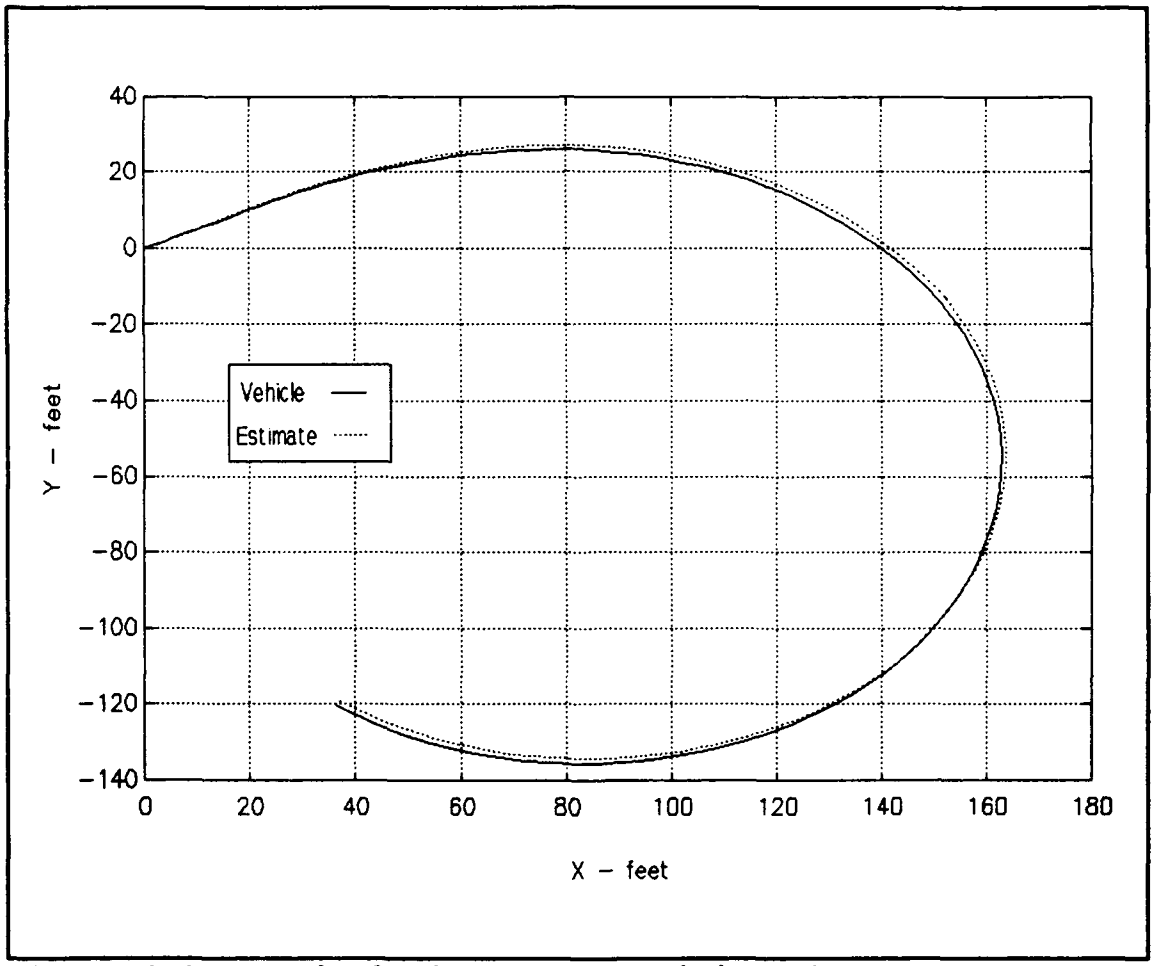

Figure 4.1 SDV simulation - X-Y position plot

accurate position estimate at all times. This is due in part to the lack of position error feedback which cannot be accomplished because position is not measured. This error is also due to the effects of the transients in the Extended Kalman filter. As shown in Figures 4.2 through 4.4, these transients affect the velocity estimates from which position is generated.

The forward speed is estimated accurately after approximately 40 seconds as shown in Figure 4.2. As previously mentioned, the estimation error at the beginning

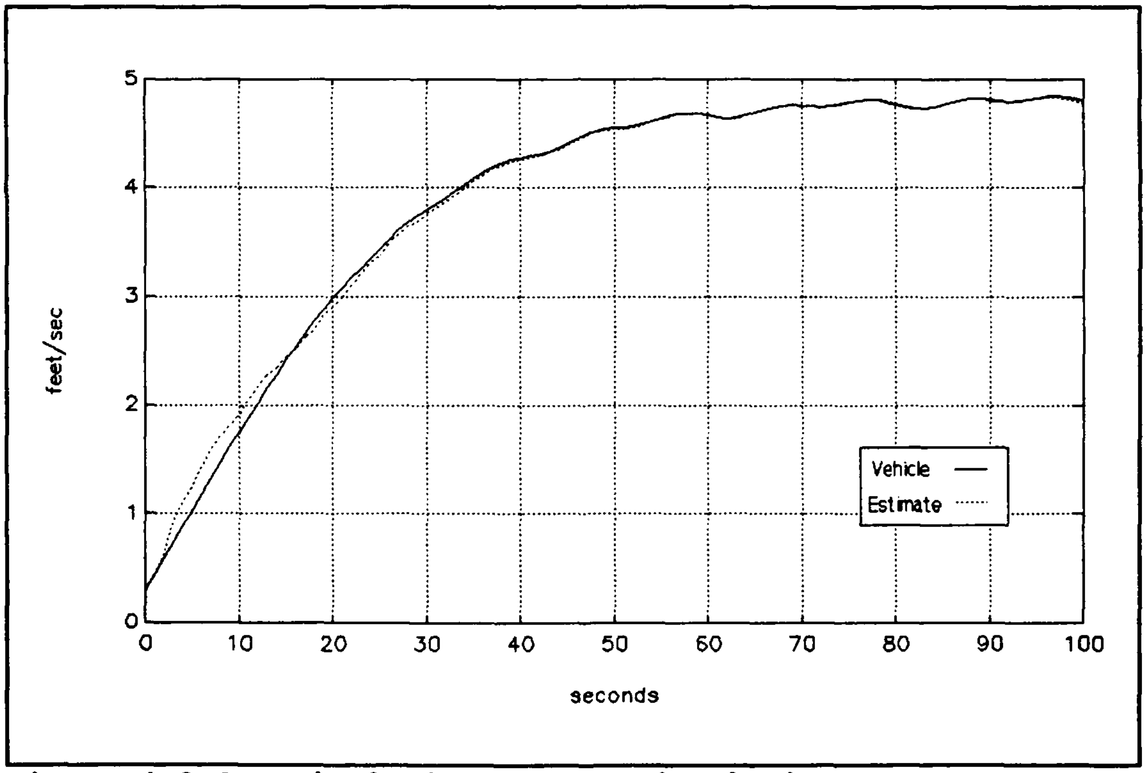

Figure 4.2 SDV Simulation - Forward velocity

of the run is due to filter transient effects. The sinusoidal variation of the speed is due to the drag indu‹:›ed by the dive plane angle and resulting porpoising maneuver. This effect is much more pronounced in *v* and *w*.

Both *v* and *w* exhibit similar transient error periods. Figure 4.3 shows that the lateral velocity, or side slip, *v*, approaches a constant value induced by the rudder command and resultant turn. The convergence of the filter is faster for this state than for the forward speed. The dive plane commands also affect the side slip as evidenced by the

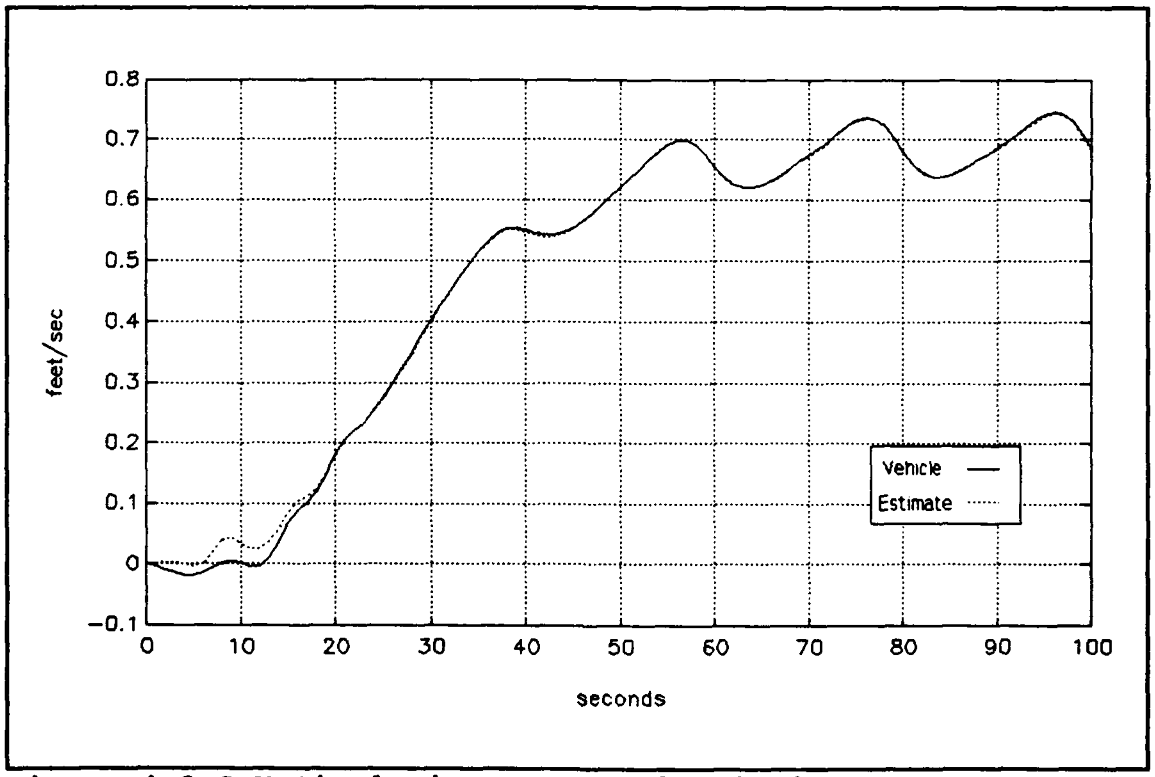

Figure 4.3 SDV Simulation - Lateral velocity

oscillation about 0.7 feet per second which is caused by the rolling motion of the vehicle, shown in Figure 4.5.

The sinusoidal variation in *w*, shown in Figure 4.4, is much more pronounced than for either of the other two velocity terms because the forcing function of the dive command is in the same plane as *w*. Like forward speed, the filter does not converge to the correct value of *w* as quickly as it does for lateral velocity. This is due in part to the large relative error between the vehicle state

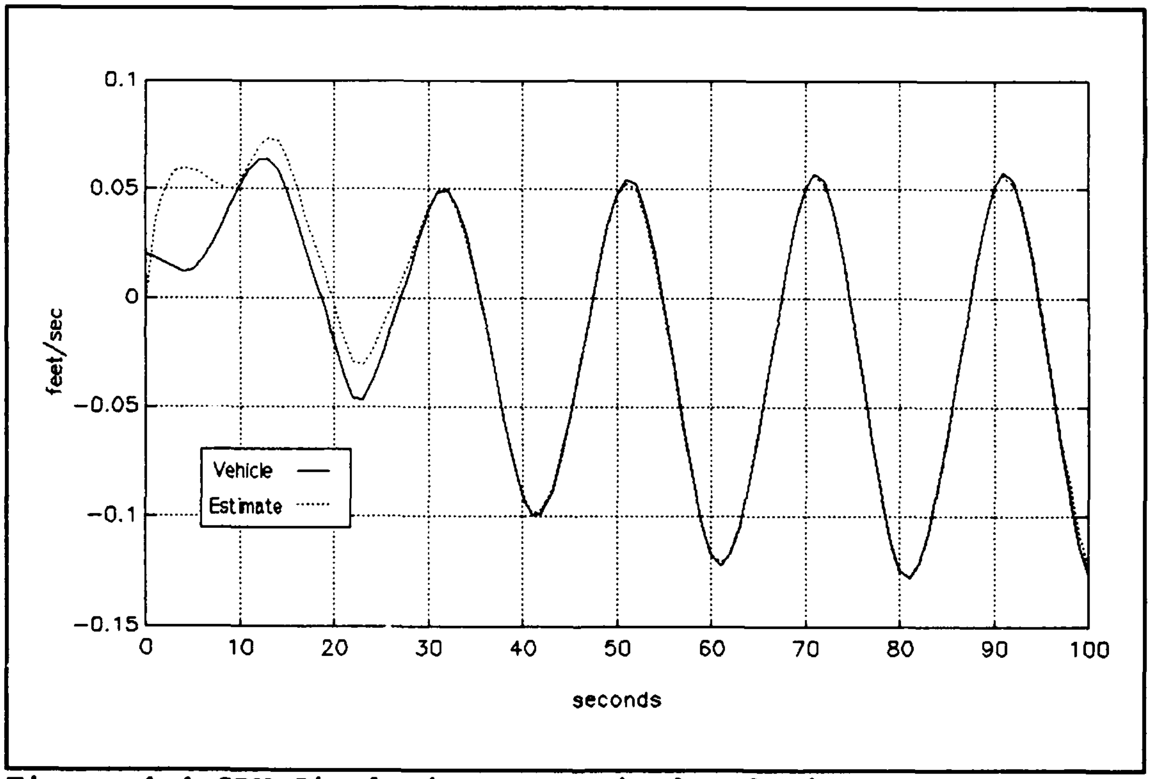

Figure 4.4 SDV Simulation - Vertical velocity

and the filter state not present in either of the two other velocity terms. This error also manifests itself in the pitch and pitch rate estimates. Finally, the mean value is not zero because the vehicle tends to roll and dive in a turn.

As mentioned above, the dive commands combine in the turn to create a varying roll effect. Figure 4.5 shows the vehicle's roll rate and estimated roll rate, *p*. The high frequency oscillations in the first 30 seconds of the simulation run are caused by the vehicle's response to an initial roll rate at low forward speed. At low speed, the

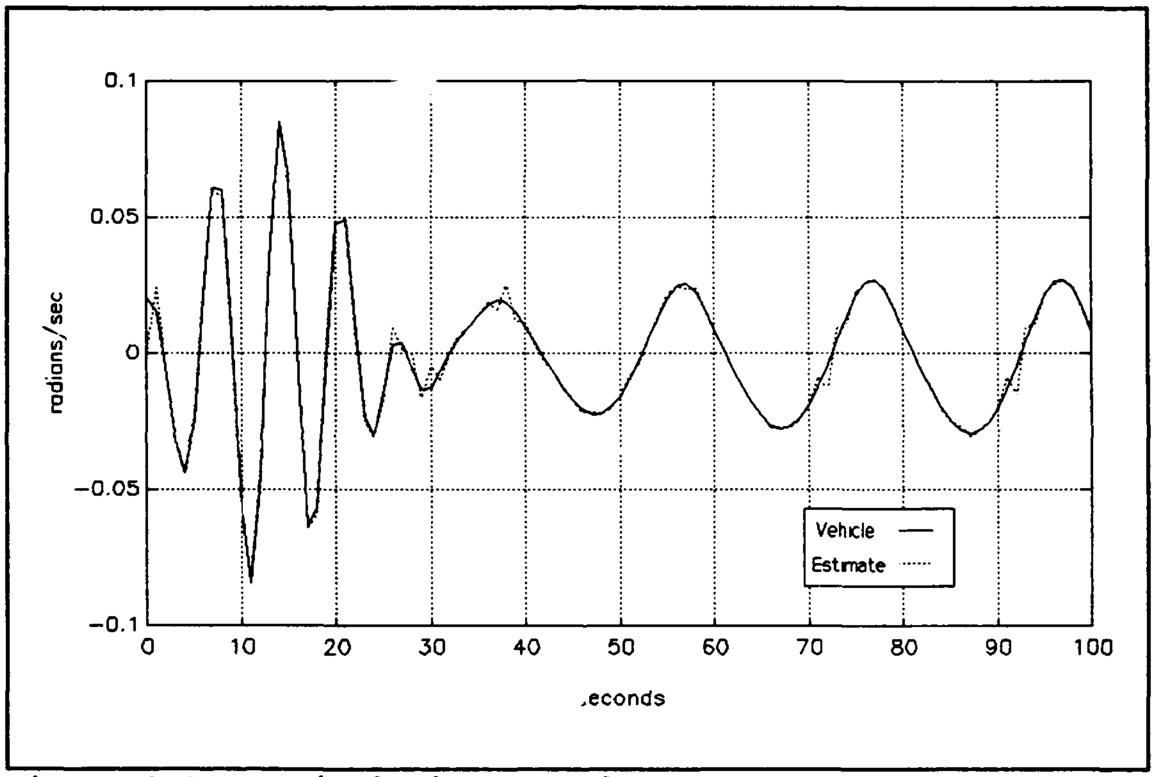

Figure 4.5 SDV Simulation - Roll rate

hydrodynamic forces on the vehicle are not sufficient to damp this mode of oscillation. The lower frequency oscillations are caused by the dive plane commands.

Because all three angular rates are measured, the filter converges to the proper values more quickly than for the velocity terms previously discussed. The measurement noise is also evident in these three estimated states.

Figure 4.6 shows the vehicle's pitch rate and the estimated pitch rate, *q*. The general shape of this plot corresponds to that of Figure 4.4 in that the magnitude of the oscillation increases with vehicle forward speed and the mean value is not zero. The filter transient effect is very pronounced in the *q* estimate. The spike occurring at the

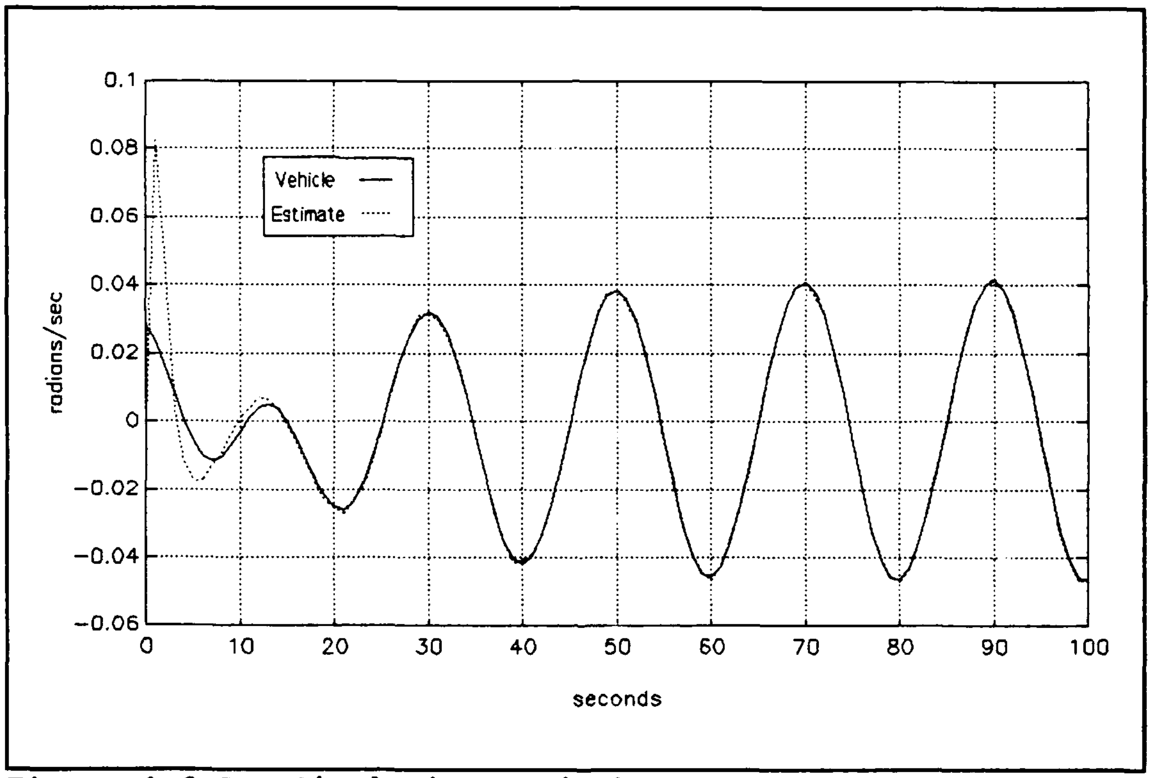

Figure 4.6 SDV Simulation - Pitch rate

one second point is related to the initial estimation error in body vertical velocity, *w*.

In a similar manner, the vehicle's yaw rate and the estimated yaw rate, *r*, reflect the estimation errors in *v*, as shown in Figure 4.7. Although the rudder command is constant for this run, the turn rate increases and approaches a steady-state value because the vehicle speed is increasing in the beginning of the turn and because the

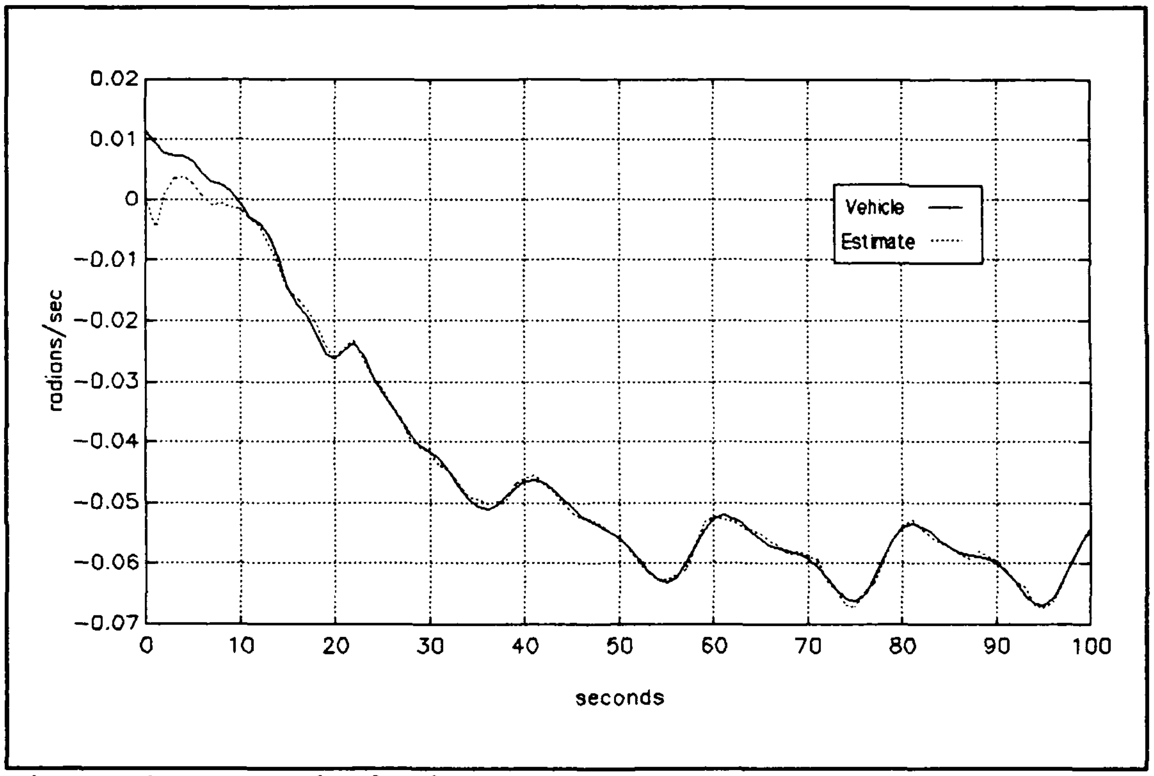

Figure 4.7 SDV Simulation - Yaw rate

vehicle has significant rotational inertia in the yaw direction. The nonlinearities in the model are evidenced by t.is plot in that the sinusoidal dive plane commands do not produce a strictly sinusoidal change in yaw rate.

The scale in this figure tends to hide the measurement noise, but examination of the data indicates that some portion of the measurement noise is present in the estimate. As with all the measured states, the sensitivity of the filter to this noise could be changed by adjusting the ***R*** matrix.

Figure 4.8 shows the vehicle's depth and the estimated depth. As previously mentioned, the vehicle's unusual shape causes it to dive in a turn. Because a constant turn is being simulated, the depth does not approach a steady-state value. Additionally, while the dive plane commands are symmetrical, the dive response in the turn is asymmetrical.

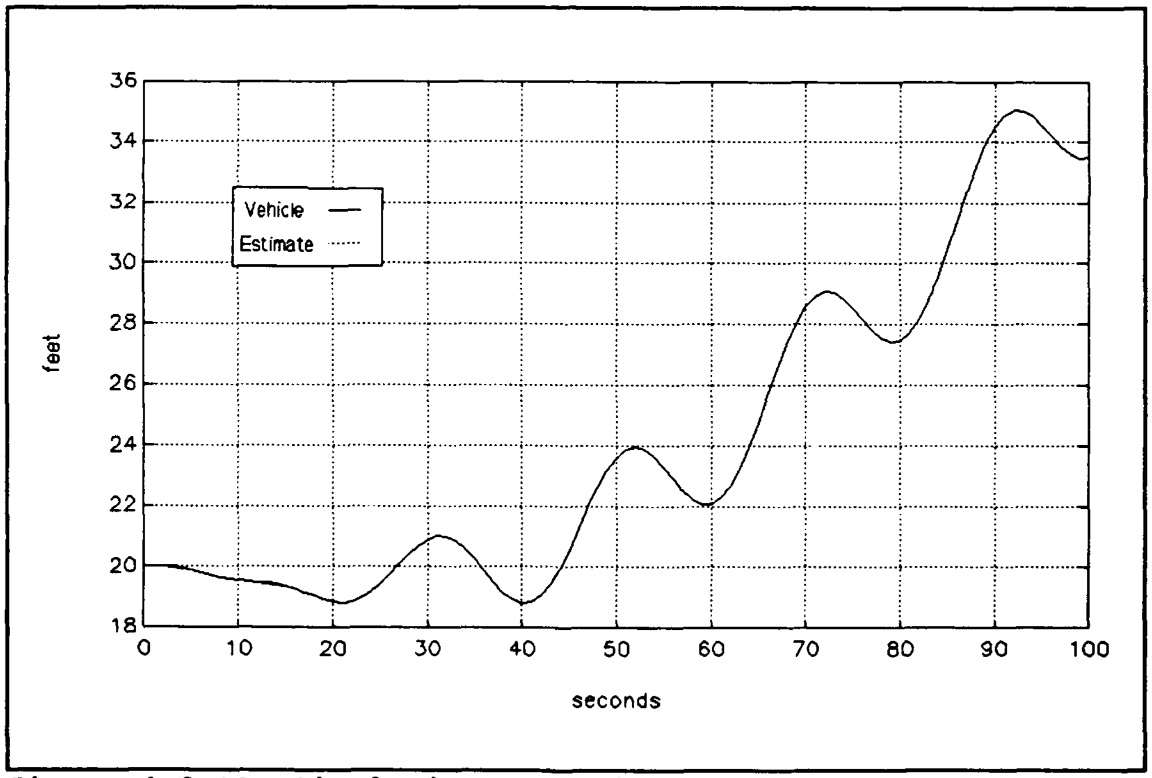

Figure 4.8 SDV Simulation - Depth

The relatively large range on the y-axis of this plot does not allow the filter transient to be seen, nor does it allow the difference between the vehicle depth and the estimated depth to be seen; however, these effects do exist.

The last three vehicle states to be discussed are the Eulerian attitude angles. As discussed in Chapter III, these estimates are derived from the nonlinear accelerometer measurement. The measurement is highly nonlinear, and therefore can lead to instabilities in the Extended Kalman filter. By making the corresponding elements of the ***R*** matrix relatively large, 10 (feet/second^2)^2, the filter is made more robust to this nonlinearity.

Figure 4.9 shows the vehicle's roll angle and the estimated roll angle, Φ. The mean value of the roll is not zero because the vehicle rolls in a turn. The horizontal intervals in the estimated roll angle are caused by the accelerometer dead zone. As discussed in Chapter III, the row of the C matrix which corresponds to an accelerometer estimated to be operating in its dead zone is set to zero so that the Extended Kalman filter does not expect a measurement. The graph has discontinuities because the estimated accelerometer measurements and the simulated accelerometer measurements do not reach their dead zones simultaneously.

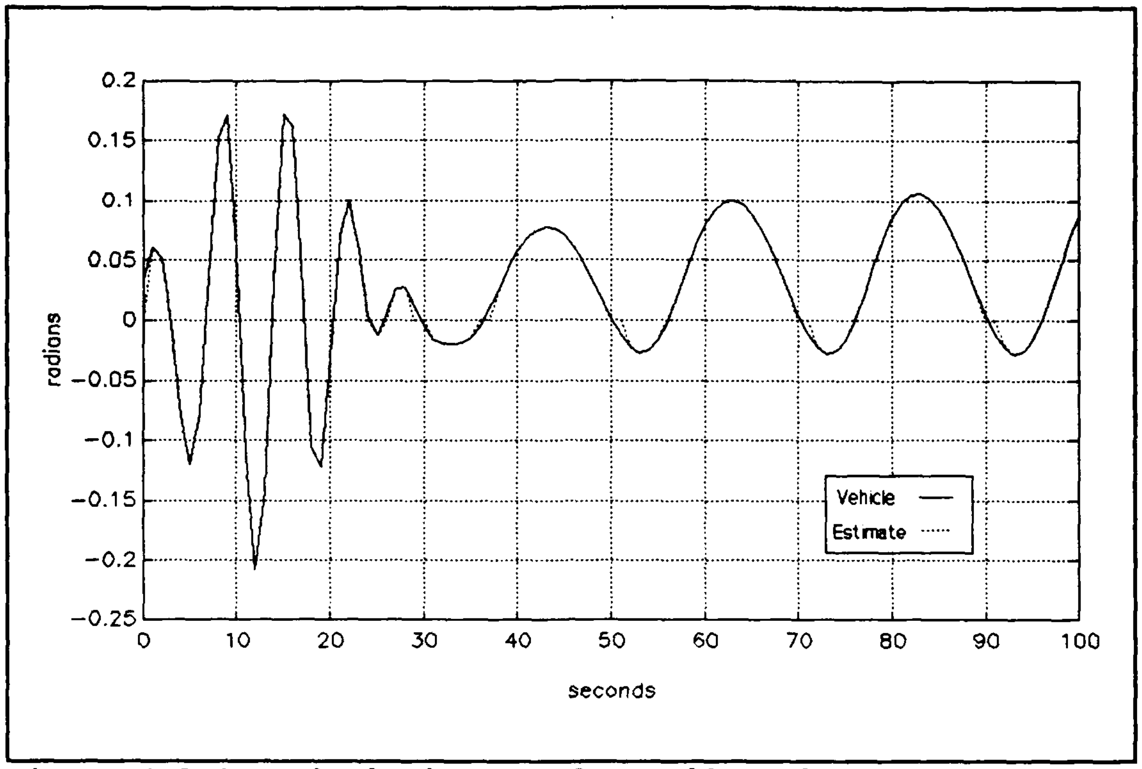

Figure 4.9 SDV Simulation - Euler roll angle

The Euler elevation angle (pitch), Θ, estimate, shown in Figure 4.10, exhibits a filter transient response that combines the effects of both the *w* and *q* estimates; however, the estimate converges well and tracks the vehicle pitch accurately in spite of the nonlinear accelerometer measurement associated with this state.

The heading angle, Ψ is not as sensitive to the accelerometer measurement, although it is in that measurement equation, because it is measured directly by the simulated heading gyroscope. Figure 4.11 shows that the

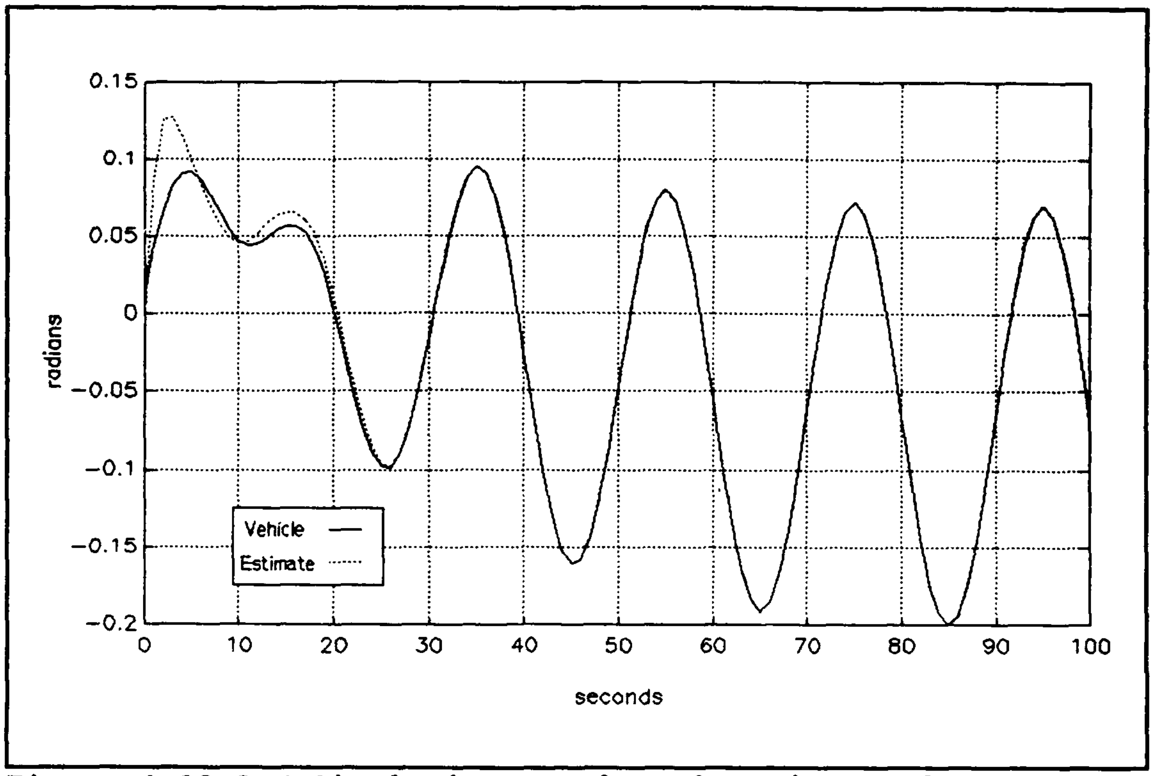

Figure 4.10 SDV Simulation - Euler elevation angle

estimate and actual state are virtually indistinguishable. As with the depth plot, the range of the vertical scale prevents differences between the vehicle state and the estimated state from being observed. Because the heading gyroscope is reliable, the elements of ***R*** matrix associated with Ψ were made small so the filter would track the measurement closely.

Finally, Figures 4.12, 4.13, and 4.14 show the estimated bias terms for each of the three rate gyroscopes. These bias terms were simulated by adding a constant term to each of the rotational rate states before passing them to the Extended Kalman filter as measurements.

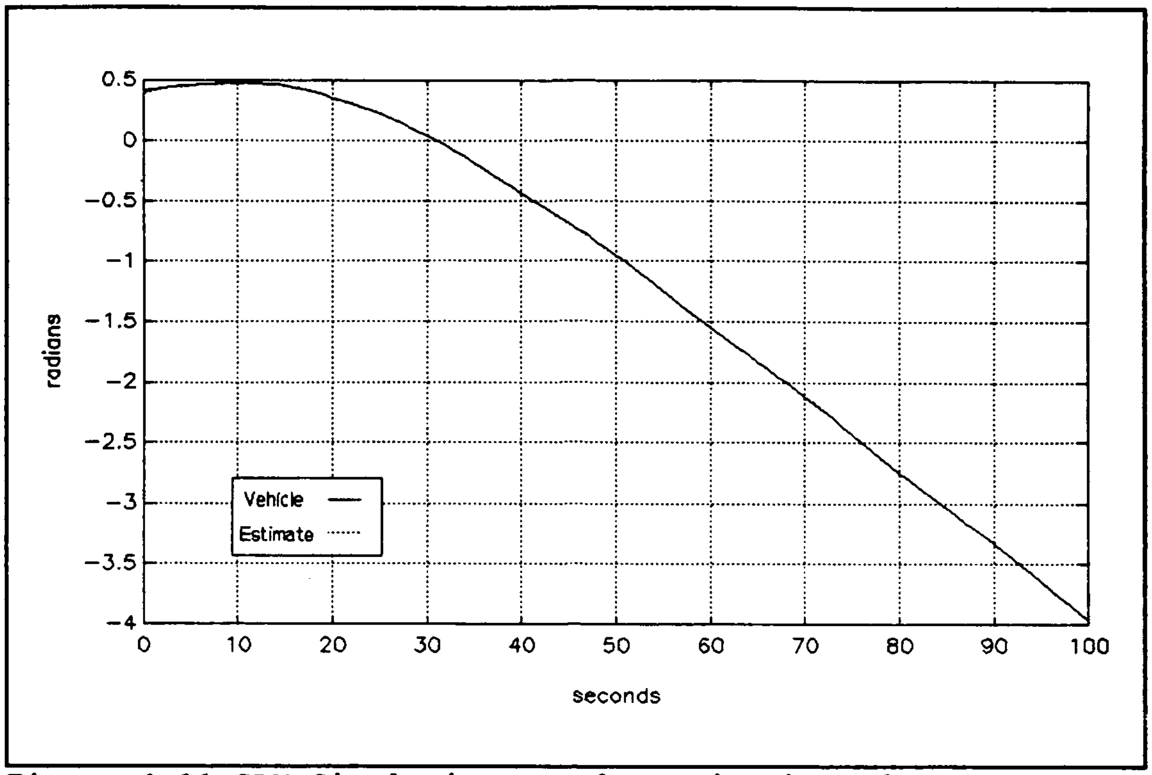

Figure 4.11 SDV Simulation - Euler azimuth angle

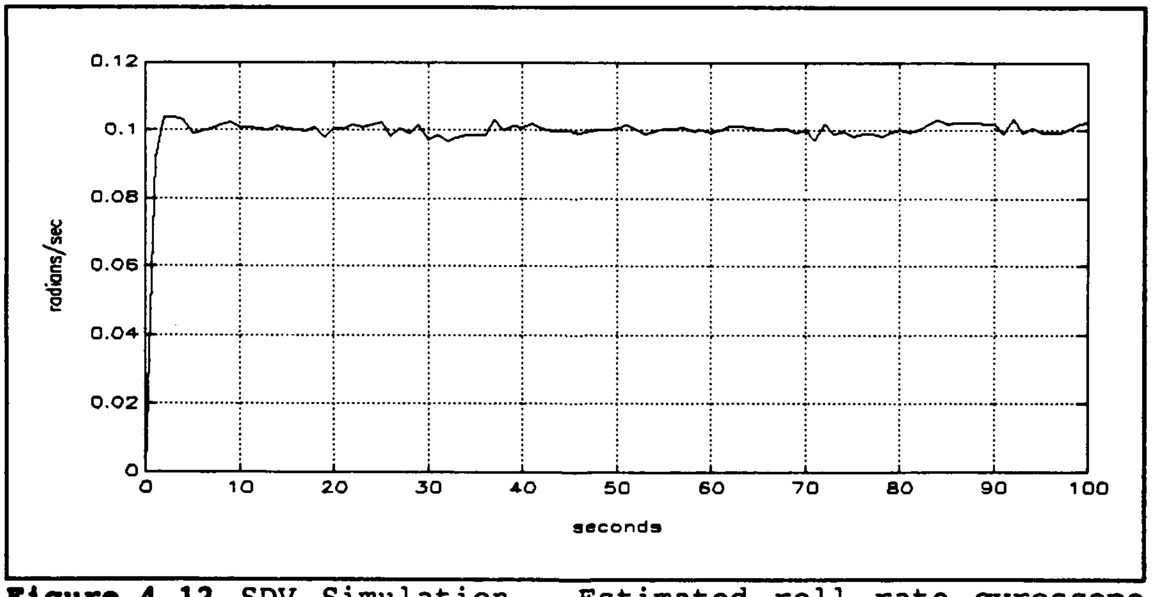

Figure 4.12 SDV Simulation - Estimated roll rate gyroscope bias

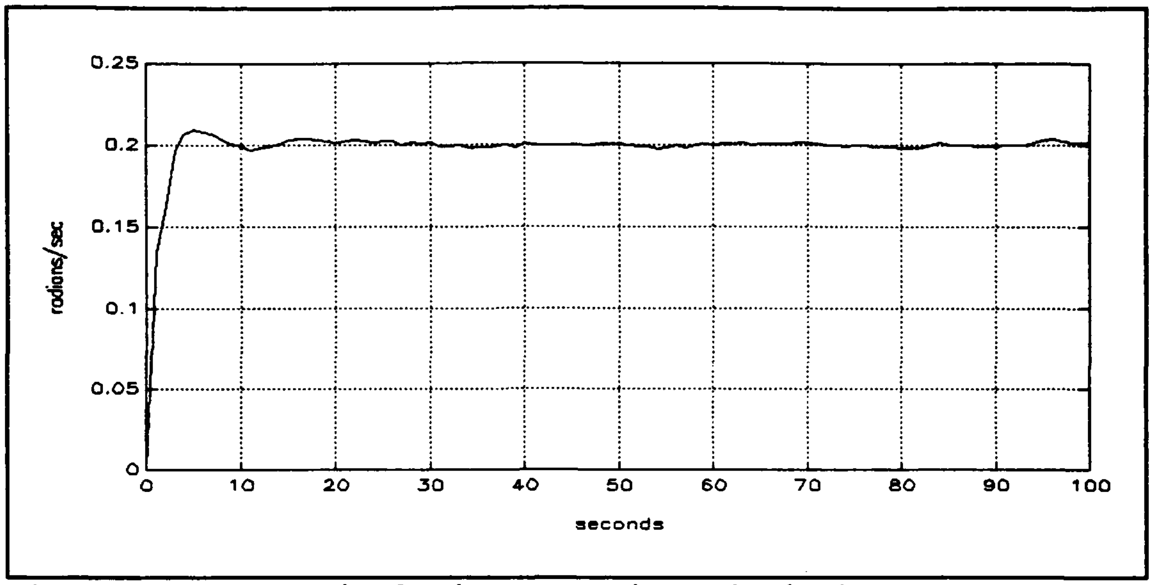

Figure 4.13 SDV Simulation - Estimated pitch rate gyroscope bias

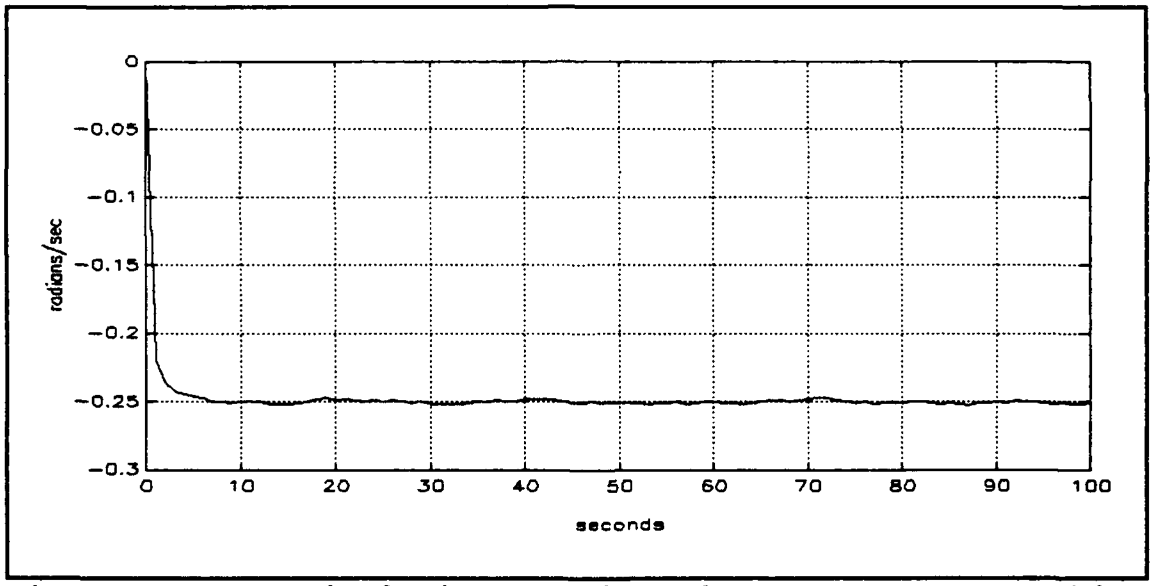

Figure 4.14 SDV Simulation - Estimated yaw rate gyroscope bias

In all cases, the filter is able to accurately estimate the simulated gyroscope bias and thus remove its effect from the measurements.

C. NPS AUV II SIMULATION

1. Typical Maneuver

Figures 4.15 through 4.28 show the simulated and estimated states of the NPS AUV II performing a maneuver similar to that of the SDV described in Section B. As in the previous section, the vehicle was given a low initial forward speed, *u*, of 0.3 feet/second, a constant propeller speed of 550 rpm, a constant bow and stern rudder command of 0.2 radian and -0.2 radian respectively, a sinusoidal dive plane command and an initial error between the filter states and the vehicle states was introduced. A sampling interval of 0.2 seconds has been used.

The results of this simulation are similar to those depicted for the SDV in the previous section. Because the two vehicles are geometrically similar, they share the common characteristic of diving in a turn. However, because the NPS AUV II is a much smaller vehicle, its dynamics are much faster and the Extended Kalman filter responds differently.

The noise rejection performance of this Extended Kalman filter is poorer than in the SDV simulation, although the noise covariance matrices used were the same for both, and the absolute magnitude of the additive pseudo-random noise was identical for both. The difference can be attributed to the different dynamic behavior exhibited by the vehicles.

There is one difference in the measurement noise covariance matrix, ***R***, between the two simulations. The elements associated with the accelerometer sensitive to pitch have been increased by a factor of ten to 100 $(\text{feet/sec}^2)^2$. If this were not done, the filter would be not be stable.

Although Figure 4.15 seems to imply that the navigator for the NPS AUV II is not as accurate as the navigator for the SDV, the scale is misleading. As with the

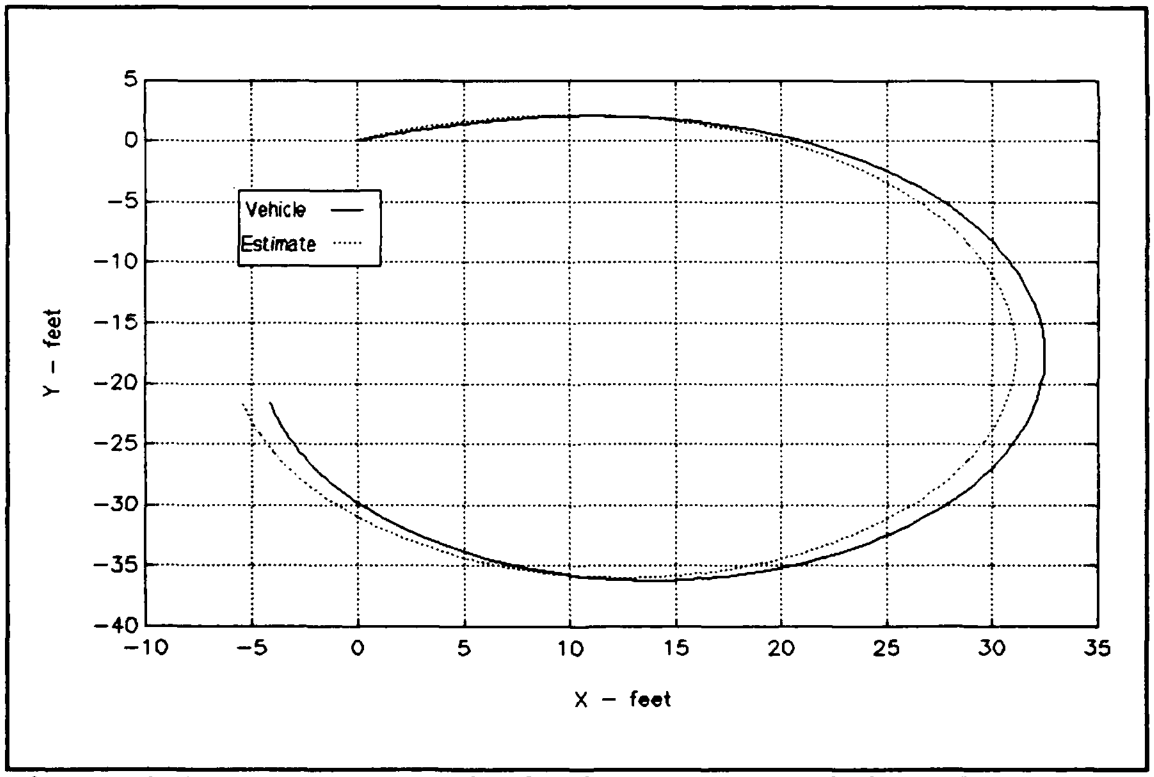

Figure 4.15 NPS AUV II Simulation - X-Y position plot

SDV, the position error shown here is due to the velocity errors in the state estimates which are caused by the transient response of the filter. The transient is induced in large part by the initial random vehicle state. Although other transient errors exist, it is the velocity error that has the most impact on the position estimate. Other simulation runs have produced more accurate position estimates, but this run is included because the random initial estimation error seems to have caused the filter to perform at its worst.

The random initial error does not appear to have affected the forward speed estimate, as shown in Figure 4.16. This figure also shows that the forward speed of the vehicle is not strictly first order as evidenced by the slight overshoot. This overshoot is attributed to the relative rotation of the three-dimensional vehicle velocity vector with respect to the vehicle coordinate system that occurs when the vehicle turns. Before the vehicle yaws into a turn all the velocity is along its longitudinal axis. As it begins to turn this velocity is distributed into the other velocity component directions, as shown in Figure 4.17 and Figure 4.18, and the forward speed drops. These effects would not be apparent in a dead-reckoning navigator, but they are predicted and taken into account by the Extended Kalman filter. The SDV does not exhibit this behavior because of its different size and kinetic characteristics.

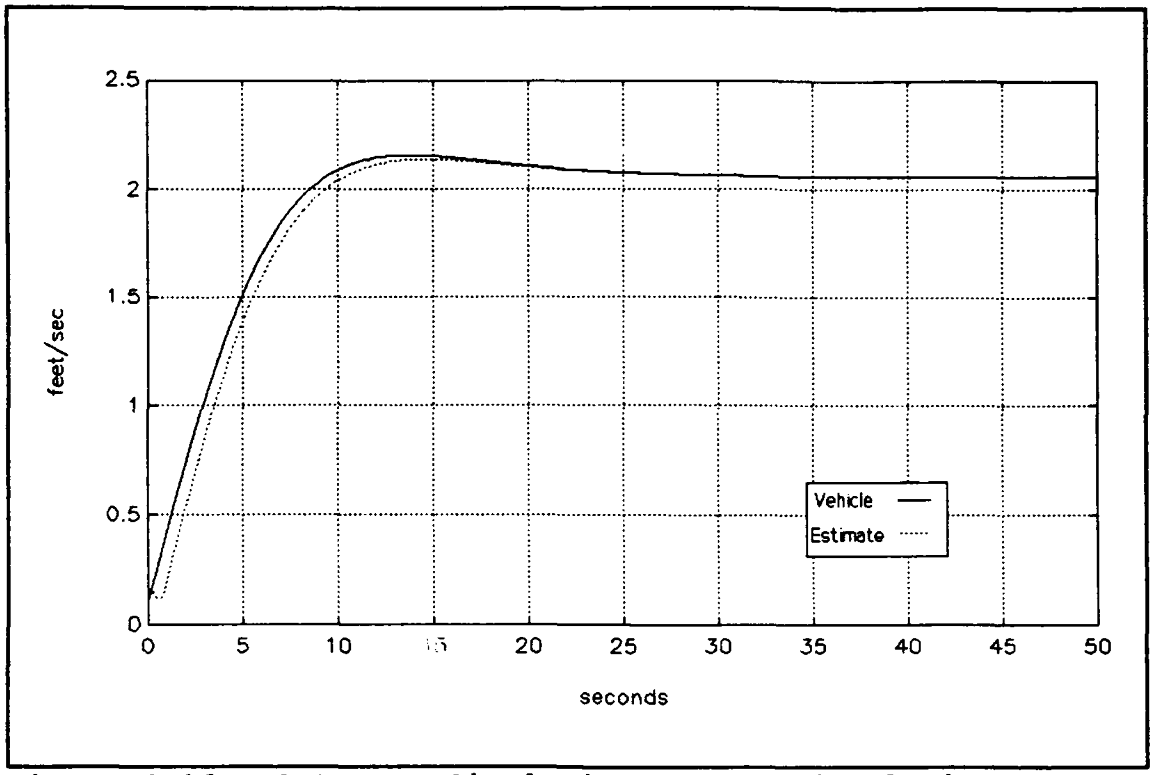

Figure 4.16 NPS AUV II Simulation - Forward velocity

The sinusoidal variation in *u* present in the SDV simulation is not present in the NPS AUV II simulation because the vehicles have different hydrodynamic characteristics.

The lateral velocity, *v*, shown in Figure 4.17, approaches a constant value as in the SDV simulation; however, unlike the SDV, the roll effect is not pronounced so there is little variation in the lateral velocity once it reaches steady state. The estimation error seen in the beginning of this run is induced by the random disturbance added to the initial vehicle state.

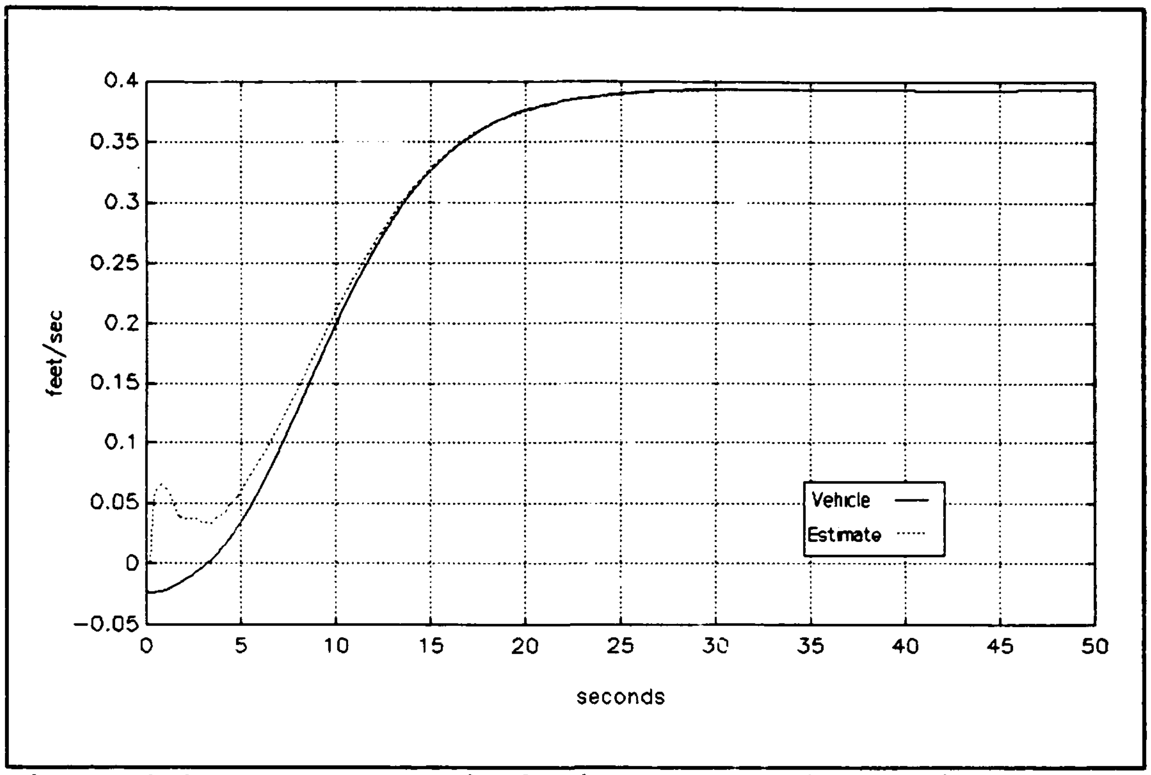

Figure 4.17 NPS AUV II Simulation - Lateral velocity

The initial error in *w* does not affect the position estimate as greatly as *u* and *v* do. However, the large initial estimation error and transient period in *w* is related to both vehicle's and the filter's transient response in pitch rate and pitch angle.

The sinusoidal variation in *w*, shown in Figure 4.18, is caused by the dive plane commands which induce a change in this component of the vehicle velocity as described in Section B.

The measurement noise rejection performance of the NPS AUV II navigator is exhibited by Figures 4.19, 4.20, and 4.21, which show the angular rate estimates. As with the

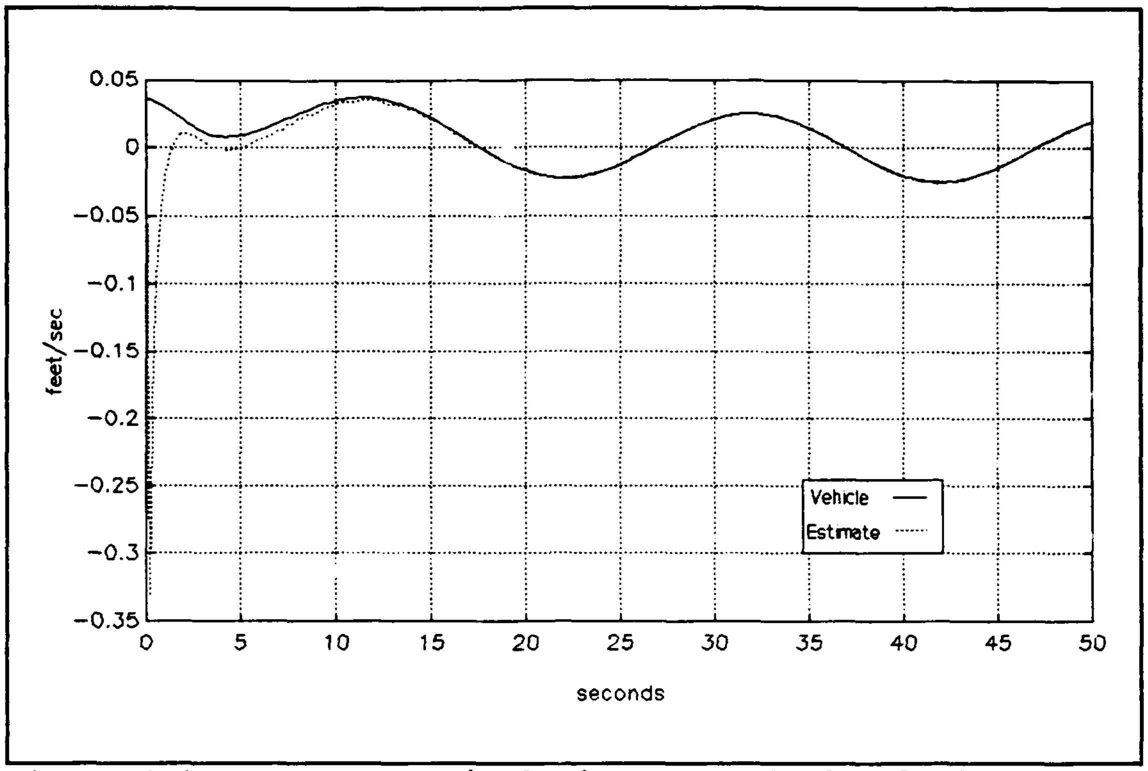

Figure 4.18 NPS AUV II Simulation - Vertical velocity

SDV, simulated measurements from the rate gyroscopes are used in the estimation of these states. Compared with the corresponding figures from Section B, these estimates are much more affected by measurement noise.

Although the roll rate, shown in Figure 4.19, exhibits the same high frequency oscillation as in the SDV, the magnitude of the roll rate induced by the dive commands is not as large. Moreover, the estimated roll rate does not converge to the actual value as quickly as for the SDV. While this is partially caused by the initial estimation error, it is more closely related to the different dynamic behavior of the two vehicle models.

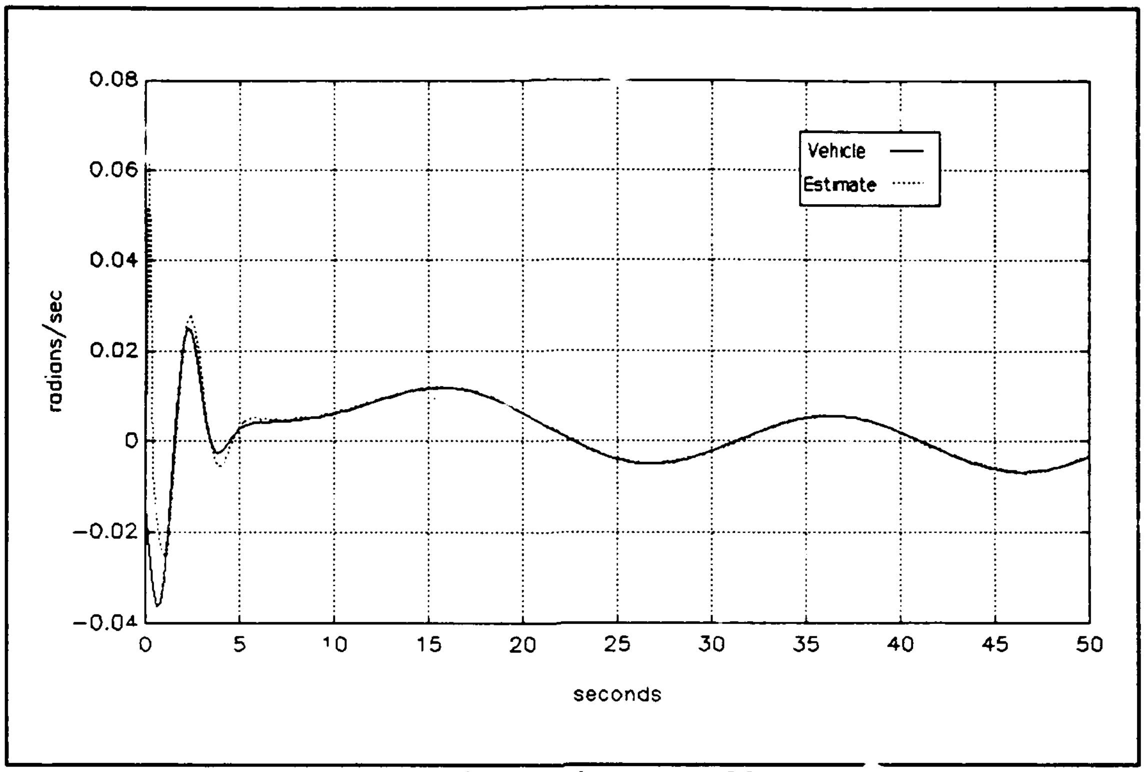

Figure 4.19 NPS AUV II Simulation - Roll rate

As with the roll rate, the pitch rate estimate, shown in Figure 4.20, exhibits high frequency measurement noise. The filter's transient response, evident in this figure, also shows that the initial error in pitch rate is related to both the initial error in *w* and in pitch, Θ, as previously mentioned.

The transient response of the vehicle is also evident in this figure. The behavior through the first 20 seconds of the simulation is due to the random initial condition of the vehicle causing it to follow a complicated trajectory which has the result of making the filter less effective in estimating the states.

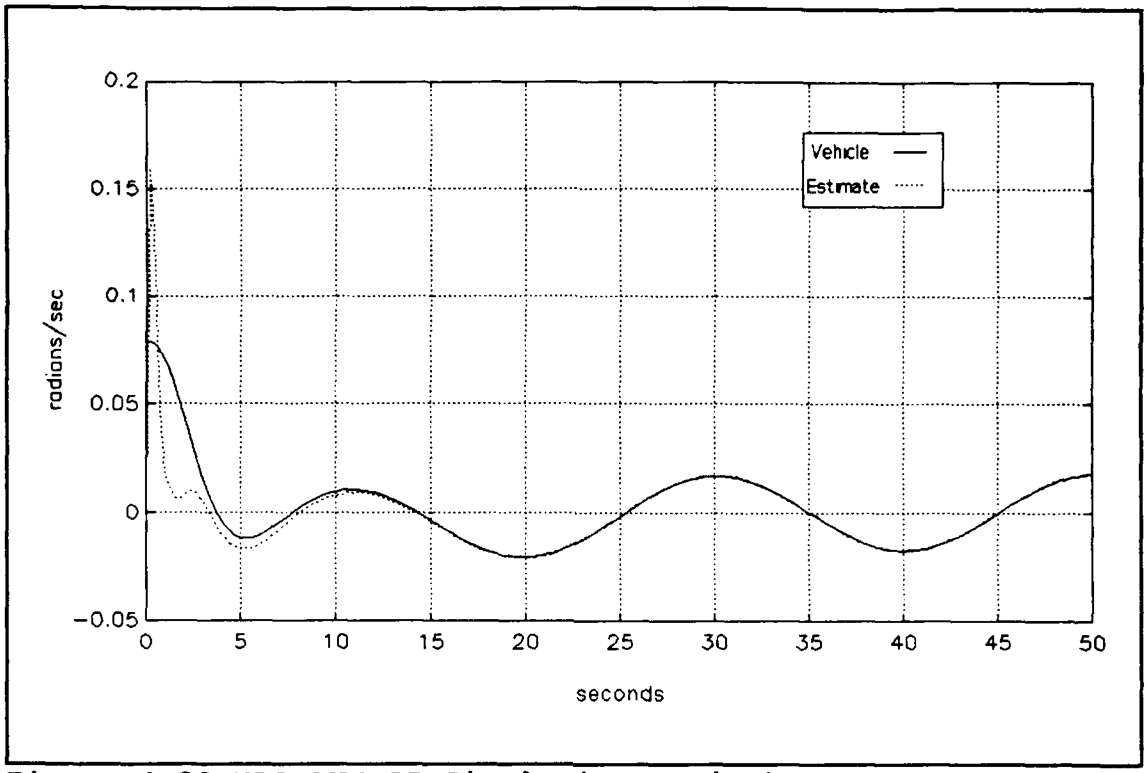

Figure 4.20 NPS AUV II Simulation - Pitch rate

As with the SDV, the NPS AUV II rolls and dives during the turn; therefore, the average roll rate and pitch rate shown in the two previous figures are not zero.

The yaw rate of the NPS AUV II, shown in Figure 4.21, differs from the yaw rate of the SDV shown in Section B in that the coupling between the dive plane commands and the yaw rate is not evidenced in the response of the AUV II. The oscillation evident in Figure 4.7 is not present in Figure 4.21. This is due to the lower cross-coupling effects between roll and pitch discussed above.

After the initial negative spike, the yaw rate estimate quickly converges to the actual yaw rate and tracks

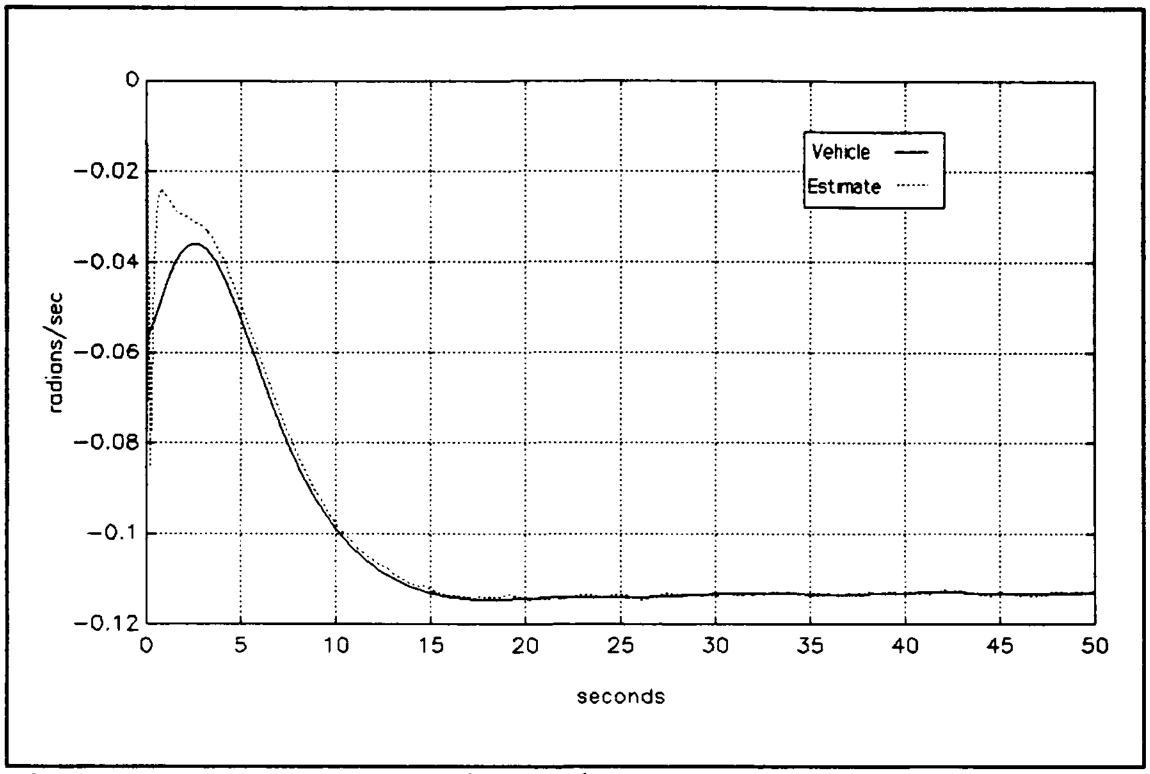

Figure 4.21 NPS AUV II Simulation - Yaw rate

it through its transient period, although it does exhibit some measurement noise corruption.

While both the SDV and the NPS AUV II dive in the turn, Figure 4.22 shows that the smaller vehicle does not exhibit this characteristic to as great a degree as the SDV. This fact is related to the smaller roll angle experienced by the vehicle during the turn shown in Figure 4.23 and evidenced in the roll rate and lateral velocity.

For the depth estimate, the Extended Kalman filter does not have a significant transient period and the estimated state converges more quickly to the actual value

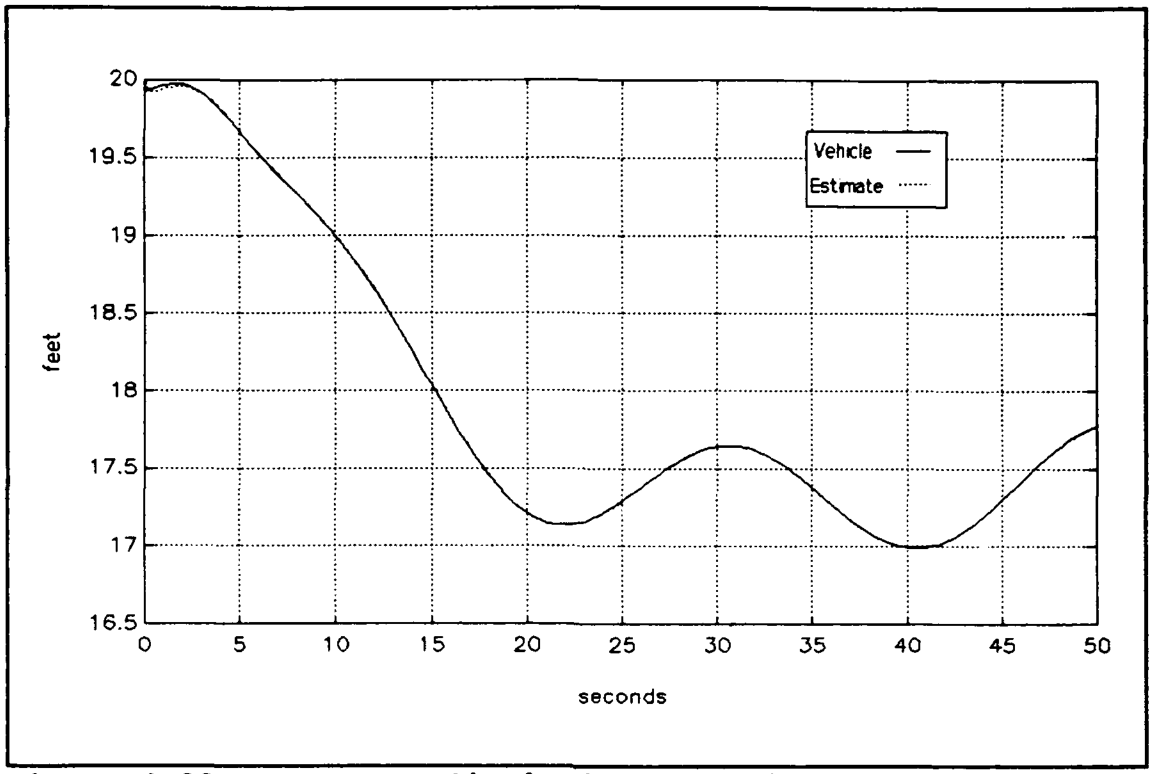

Figure 4.22 NPS AUV II Simulation - Depth

than the other states do, even in the presence of measurement noise and the initial estimation error.

Figures 4.23 through 4.25 show the last three state estimates, the Euler angles. In general, these show a more significant transient period than for any of the other estimates. Moreover, the Extended Kalman filter does not reject the noise for roll or pitch as well as it does for the rate gyroscope measurements.

The noise evident in both the roll and pitch estimates is caused partially by the noise added to the simulated accelerometer measurements. It is also caused by the nonlinear nature of these measurements including both

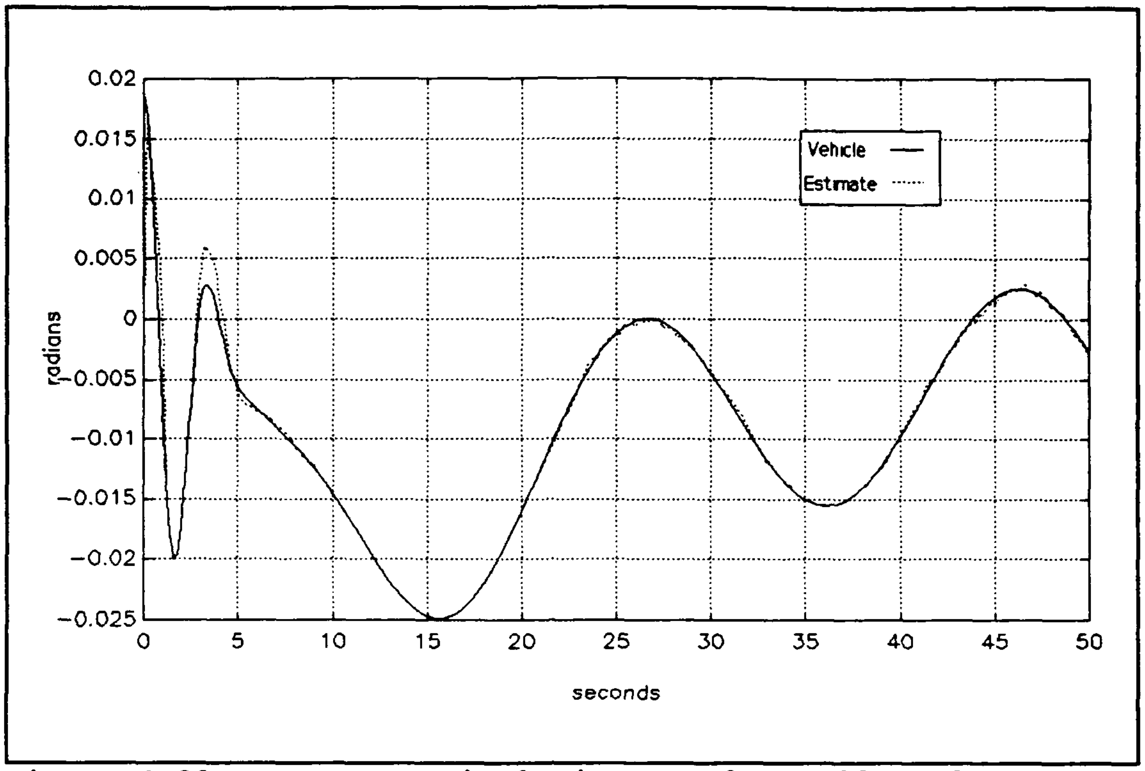

Figure 4.23 NPS AUV II Simulation - Euler roll angle

the trigonometric nonlinearities in the coordinate transformation equation and the accelerometer dead zone nonlinearity.

The heading estimate, shown in Figure 4.25, does not appear to have the noise corruption evident in the other two Euler angle estimates. This is because the heading is measured directly by the heading gyroscope. While the scale hides any noise present, the heading gyroscope does not generate a noisy signal and so little noise (standard deviation of 0.001 radians) was added in the simulation.

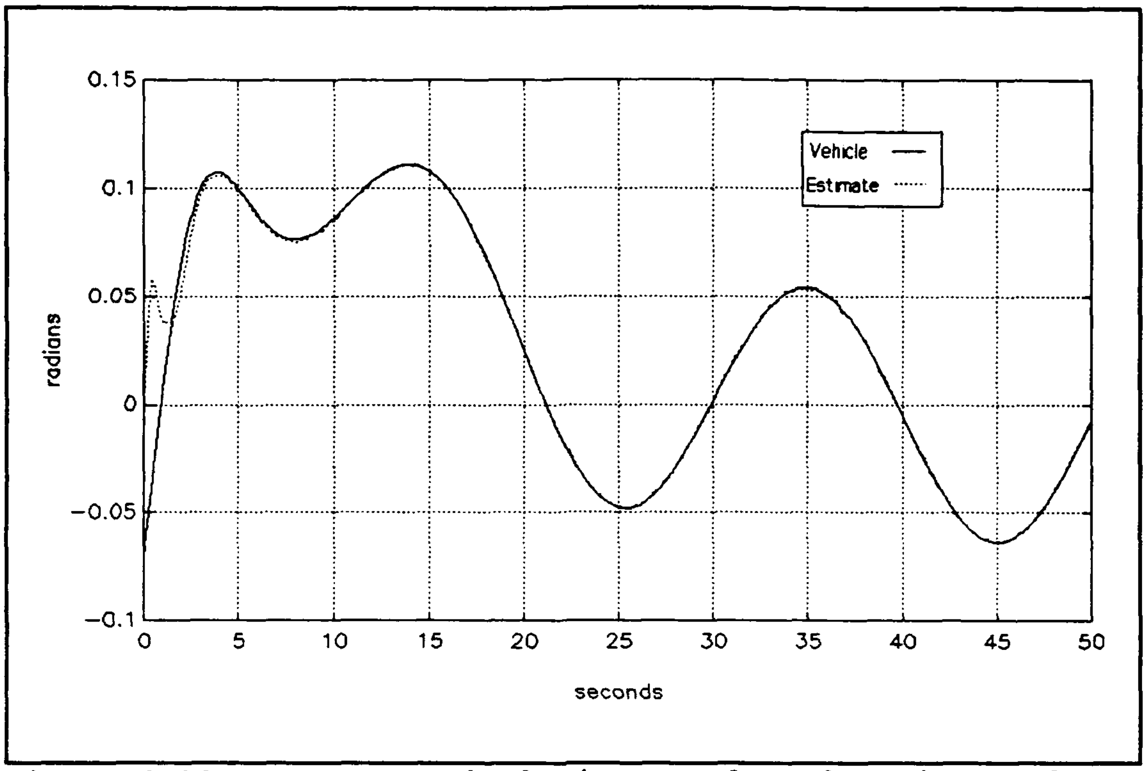

Figure 4.24 NPS AUV II Simulation - Euler elevation angle

Finally, Figures 4.26, 4.27, and 4.28 show the Extended Kalman filter's estimate of the rate gyroscope biases. These bias terms have been simulated as described in Section B and were given the same values as in the SDV simulation run for more consistent comparison. As with the SDV navigator, the NPS AUV II navigator is able to estimate this simple bias model.

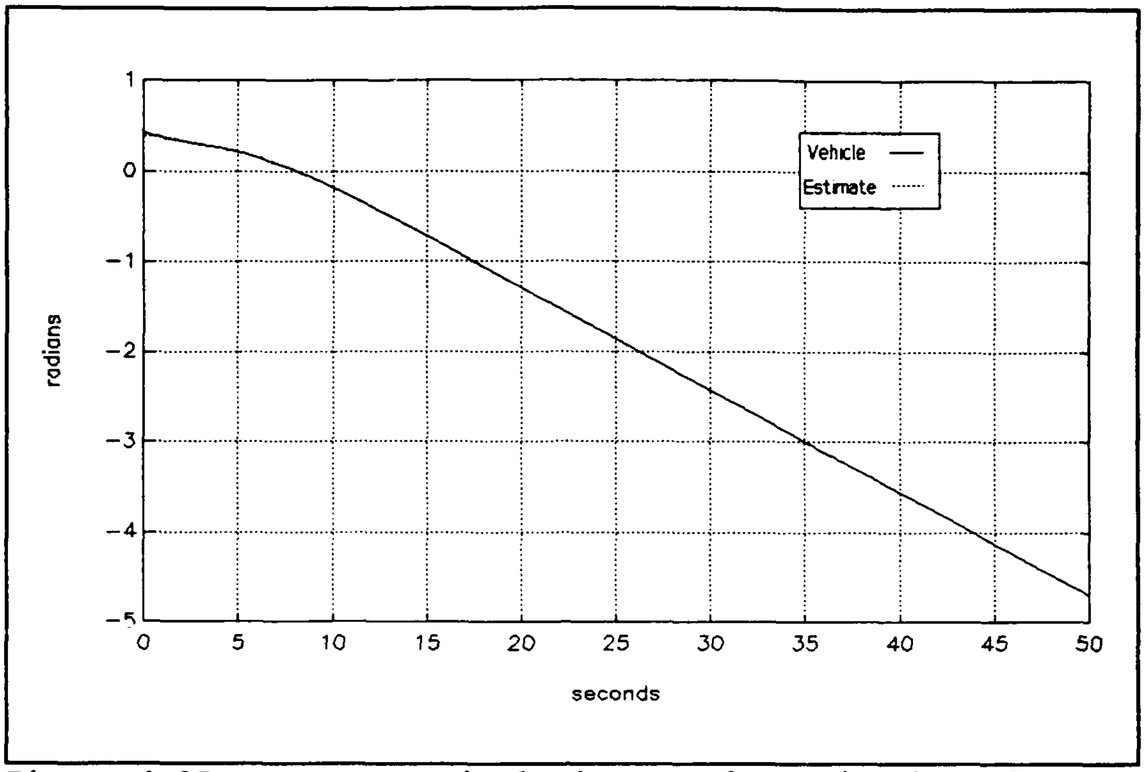

Figure 4.25 NPS AUV II Simulation - Euler azimuth angle

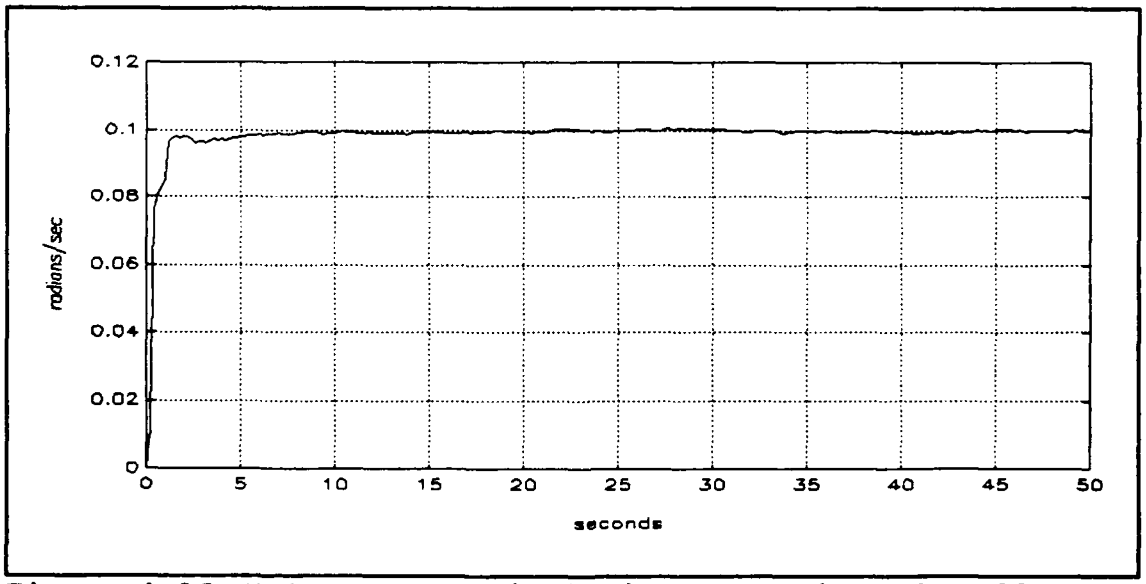

Figure 4.26 NPS AUV II Simulation - Estimated roll rate gyroscope bias

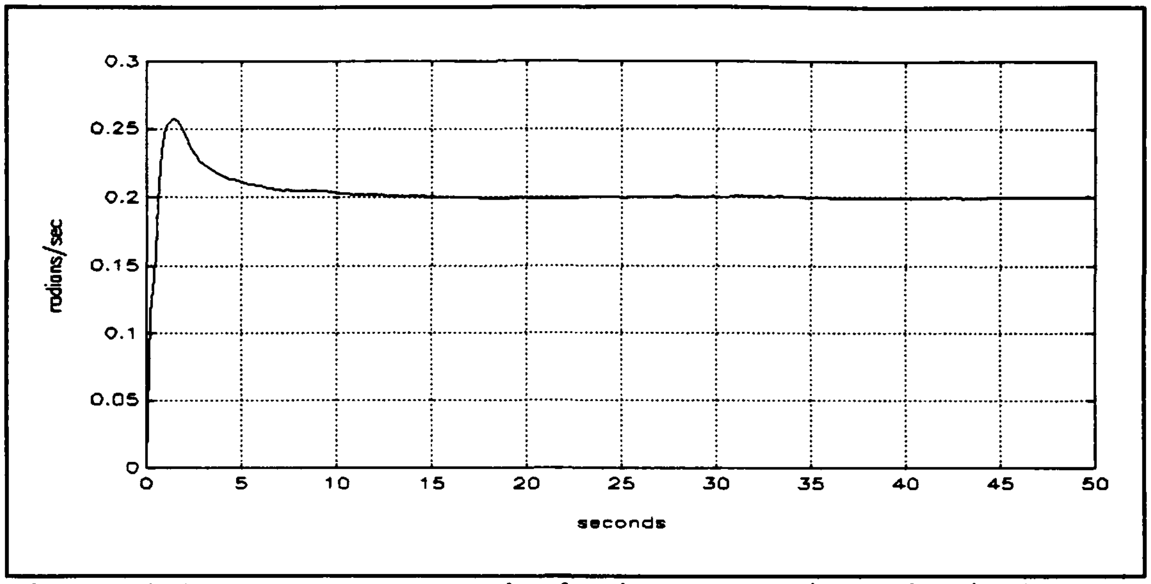

Figure 4.27 NPS AUV II Simulation - Estimated pitch rate gyroscope bias

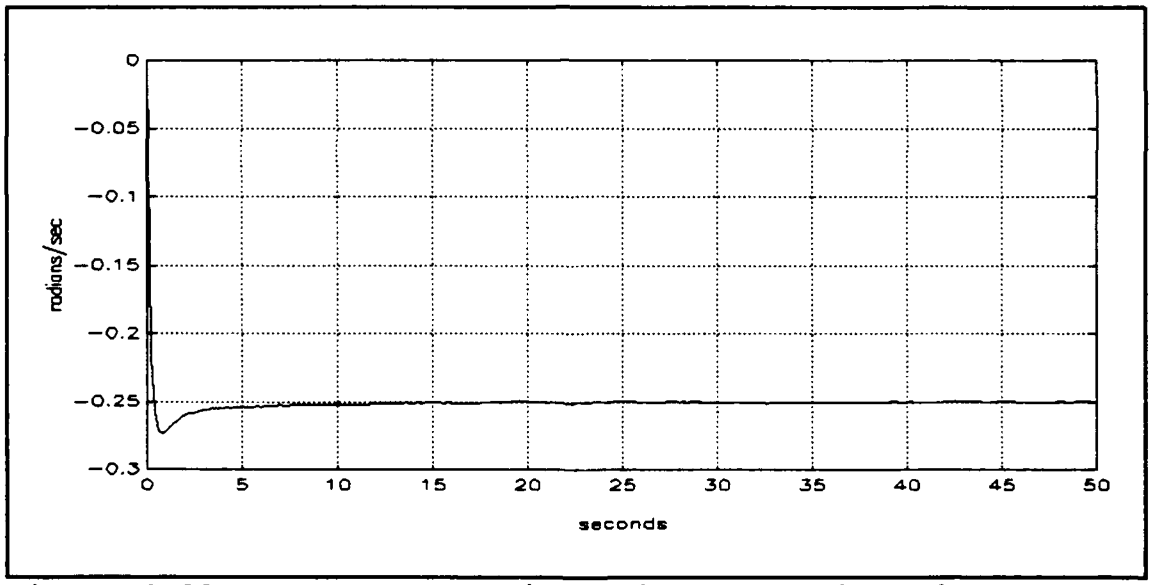

Figure 4.28 NPS AUV II Simulation - Estimated yaw rate gyroscope bias

2. Estimation of Forward Speed from Depth Rate

Figures 4.29, 4.30 and 4.31 show the results of speed estimation correction using depth-rate and pitch information. For this run the input has been changed from a steady turn to a straight line and the dive plane command has been changed to a square wave. More importantly, however, a 10 percent difference between the simulated vehicle's propeller speed and the Extended Kalman filter propeller speed input has been deliberately introduced. The filter has been given a slower propeller speed and thus it tends to generate a lower forward speed estimate.

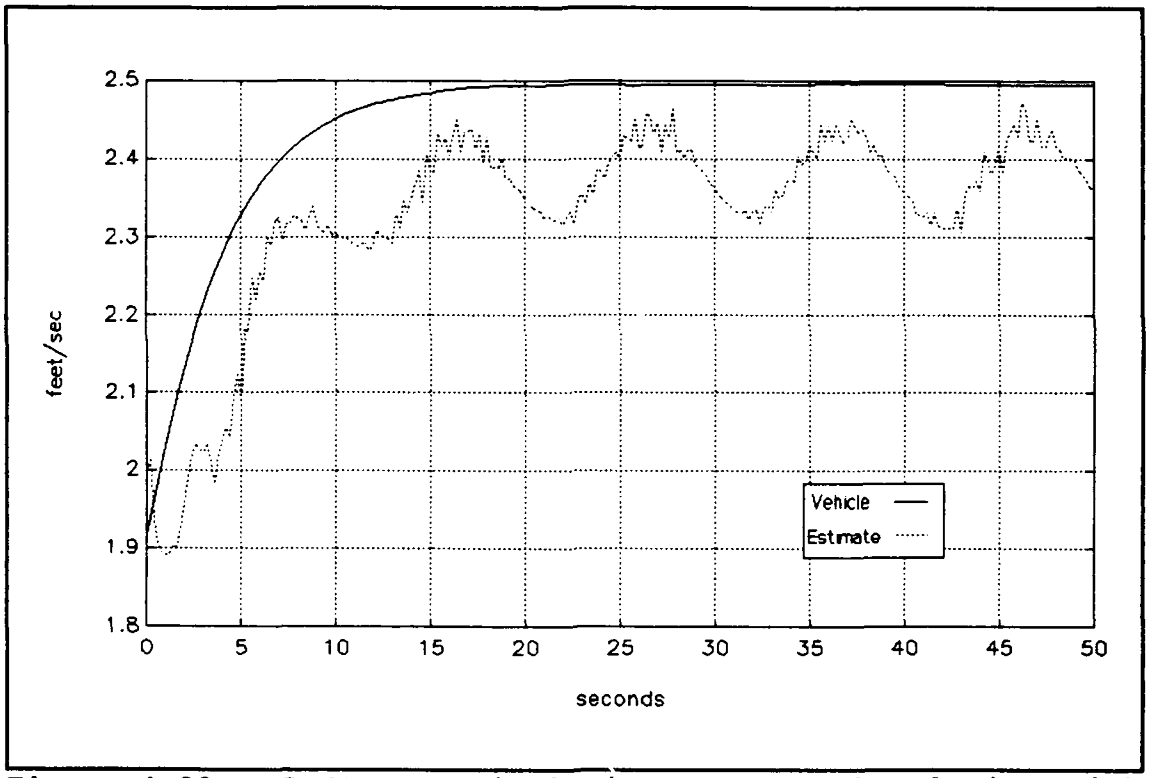

Figure 4.29 NPS AUV II Simulation - Forward velocity with propeller speed error

By changing the term in the *Q* matrix associated with the expected disturbance on the forward speed to 20 feet^2/second^2 from 0.05 feet^2/second^2, the estimate of forward speed has been made more sensitive to the correlation between depth rate and velocity. As shown by comparing Figures 4.29, 4.30 and 4.31, the estimate of *u* improves when the absolute magnitude of the pitch angle is high and when absolute magnitude of the depth rate is high. It is interesting to note that in this simulation the estimated speed actually increases when the dive planes are at their maximum deflection, a condition which induces drag and which would normally cause the vehicle speed and the estimated speed to drop.

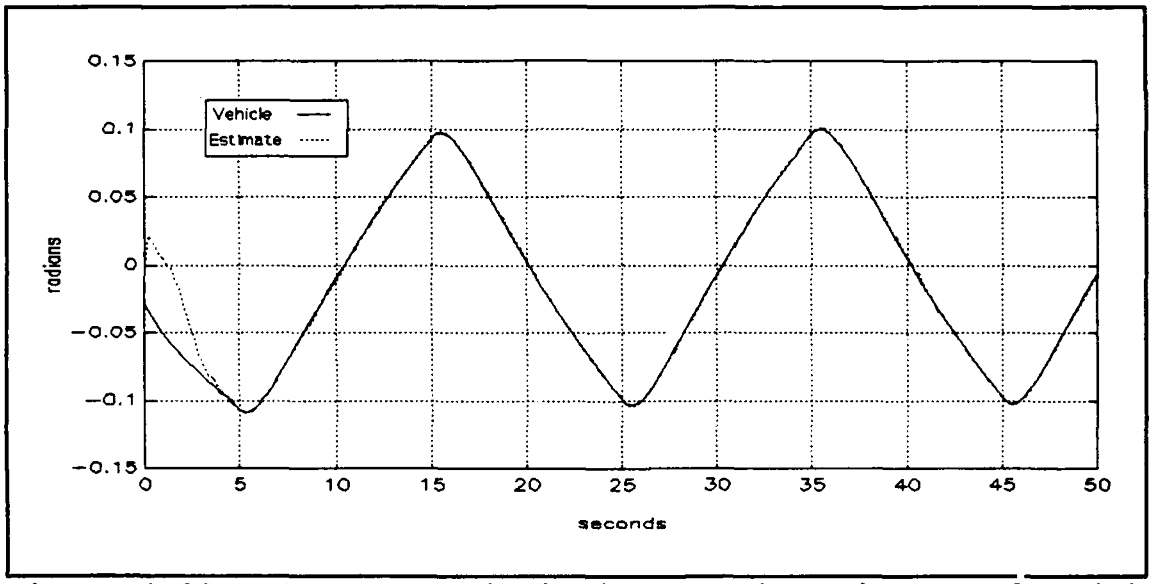

Figure 4.30 NPS AUV II Simulation - Euler pitch angle with propeller speed error

Because the Extended Kalman filter does not expect a steady-state error in *u*, the estimate oscillates. The frequency of the oscillation is at twice the frequency of the pitch and depth rate because the speed estimation is affected by the magnitude of these signals rather than their sign.

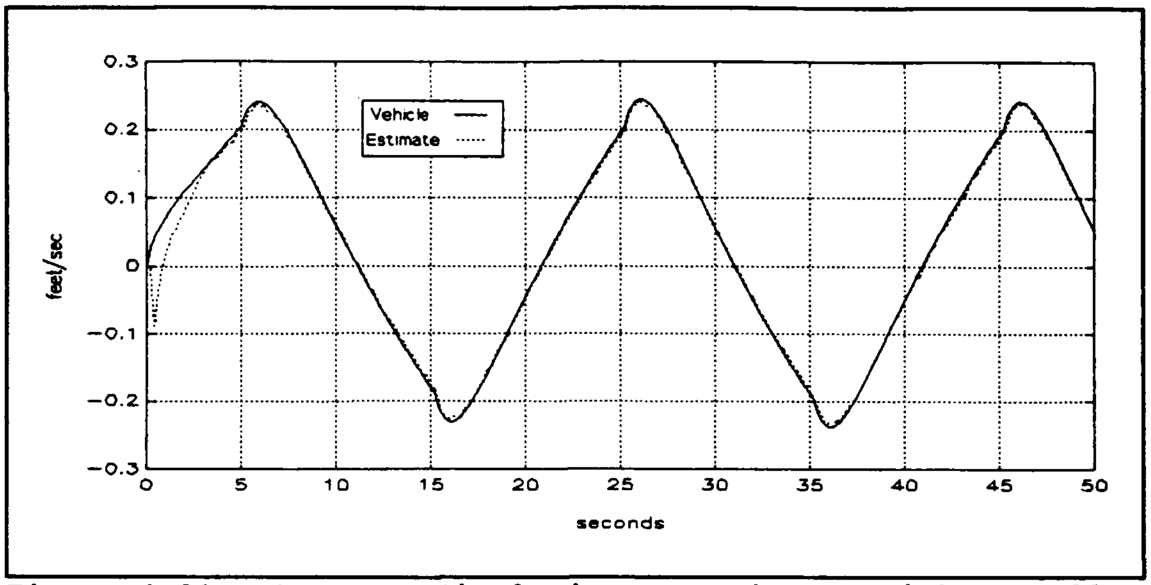

Figure 4.31 NPS AUV II Simulation - Depth rate with propeller speed error

To remove the variation in the forward speed estimate and to force the estimate to more closely track the actual vehicle speed in the presence of the induced error, i.e. when the forward speed prediction does not correlate with the estimated pitch and depth rate, the predictor was changed to include a constant propeller-speed bias term much like the rate gyroscope bias terms. This was done by appending the term from the linearized ***B*** matrix which relates propeller speed to forward speed to the linearized **A** matrix. The plant noise covariance matrix, ***Q***, and the initial filter error covariance matrix, ***P***, were modified to include this bias. The element of ***Q*** associated with forward speed was reduced to cause the filter to compensate for the prediction inconsistency by correcting the value of the propeller bias.

Some results of this modification are shown in Figures 4.32 and 4.33. As expected, the variation in the forward speed estimate is reduced; however, a steady-state

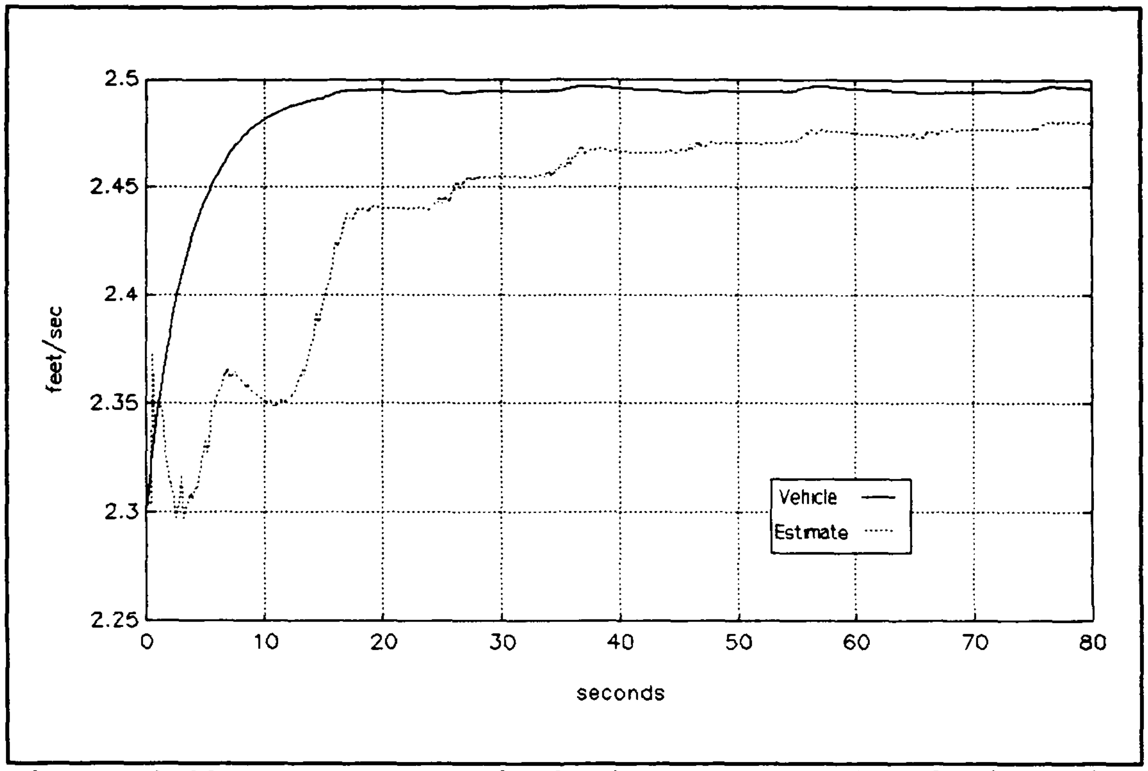

Figure 4.32 NPS AUV II Simulation - Forward velocity with propeller speed bias

error still exists between the estimated and the actual speeds. This error is a function the linearized model.

The actual propeller error introduced in this simulation was 55 rpm; however, as shown in Figure 4.33, the filter estimates the error at approximately 54 rpm. Additionally, the filter was given an initial condition for the bias of 50 rpm. While this is artificial, it did allow the filter to reach an apparent steady-state value in a

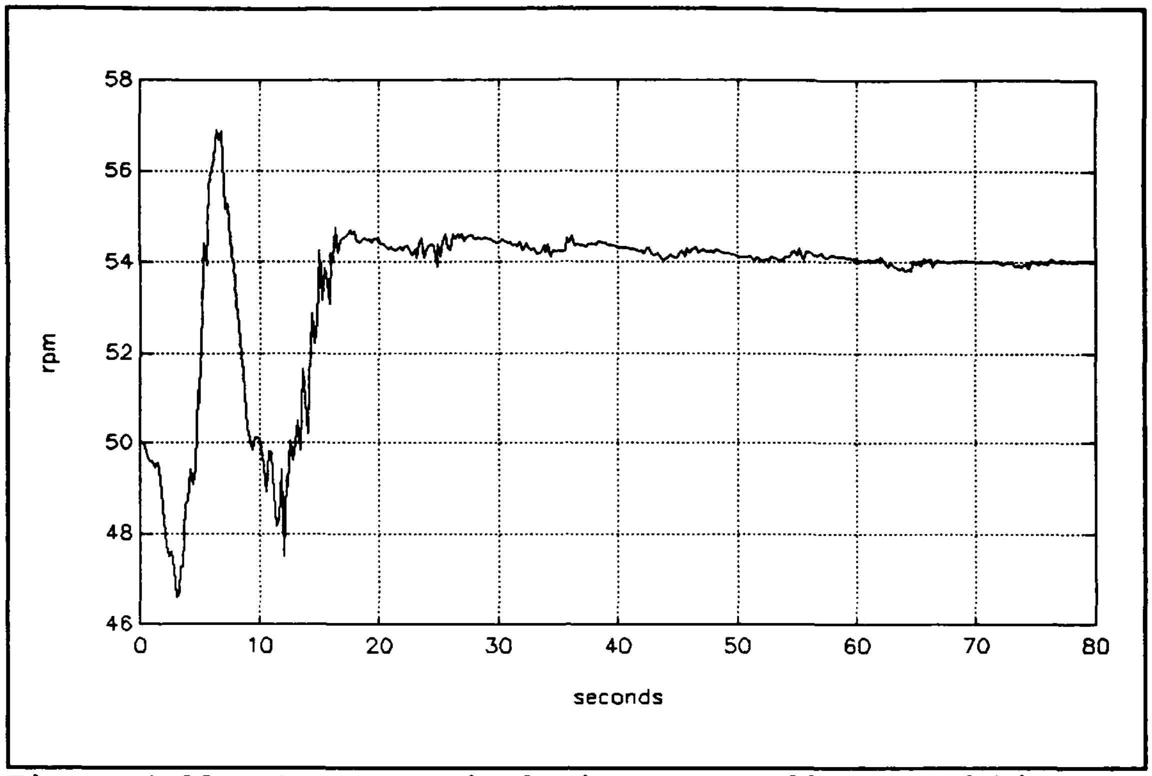

Figure 4.33 NPS AUV II Simulation - Propeller speed bias

short amount of time. If the bias were initialized at zero, then the filter would not have reached steady state in the simulated run time. It was found that the value of the bias state changed more slowly as the eigenvalues of the error covariance matrix became slower.

Because there is a steady-state error and because the length of time required by the filter to reach a steady-state value is approximately equal to the run time of a typical pool mission for the NPS AUV II, this approach to correcting a speed-estimation error is not used to filter the recorded experimental data.

V. EXPERIMENTAL RESULTS

A. GENERAL

This chapter presents the results of using actual data recorded from the instruments on board the NPS AUV II as input to the navigator rather than simulated measurements as in Chapter IV. The data was taken during a 100 second test-and-evaluation mission conducted in the Naval Postgraduate School's swimming pool on 26 August 1991. The data was recorded during the second mission run by the vehicle that day during which the vehicle made a single U-turn while porpoising at a depth of approximately two feet. A piecewise constant Extended Kalman filter gain has been used with a table look-up scheme to process this data. The navigator's output and the differences between the actual and expected results are discussed.

B. IMPLEMENTATION

The data recorded from the vehicle's instruments include a time record, a depth cell measurement, a paddle wheel speed log measurement, three rate gyroscope measurements, two measurements from a vertical gyroscope (roll and pitch), dive plane and rudder commands, and left and right propeller shaft speeds. There were no accelerometers in the vehicle at the time the data was recorded. The instruments on board the NPS AUV II have analog output which is converted to digital form by the on board computer.

The data sampling interval was 0.1 second. Although the instruments were sampled at this rate, the navigator uses data at every other sample time or at 0.2 second intervals. This reduces the navigator's computational requirements without sacrificing performance.

To further reduce the computational burden, the Extended Kalman filter has been implemented with piecewise constant gain matrices instead of the time varying matrices used in the simulations described in Chapter IV. Using this method obviates the calculation of the gain matrix on-line. Because the gain matrix is not calculated on-line, the relinearization and discretization of the vehicle model, required if the fully time-varying Extended Kalman filter were used, are not needed.

To use piecewise constant $\boldsymbol{K}$ matrices, several steady-state gain matrices were calculated for a given set of estimated states. Several values of forward speed, u, and Euler pitch angle Θ, were selected. For each combination, the corresponding steady-state $\boldsymbol{K}$ matrix was calculated and stored. The built-in MATLAB function DLQE, (discrete linear quadratic estimator) was used to calculate the gain matrices. The respective values of both states used to generate the results presented this chapter were

$$u \in \{1.5 \ 2.0\} \text{ feet/sec}$$
$$\theta \in \{-0.05 \ -0.025 \ 0.0 \ 0.025 \ 0.05\} \text{ radian.} \tag{5.1}$$

While other values have been used, these give satisfactory results because the measurements are sensitive to estimated pitch and these pitch intervals are small making the gain approximation better.

The net result of using a 0.2 second discrete time interval and using the piecewise constant K matrix is that the 100 second mission can be processed in approximately 40 seconds of processor time (MS-DOS PC with a 33 MHz Intel 80386 processor). If the entire algorithm were written in C, instead of MATLAB and C, it is expected that this time would be reduced significantly.

C. NAVIGATOR OUTPUT

Figures 5.1 through 5.16 show the estimated states generated by the navigator using the recorded data. For those states which correspond to an instrument output, the figure displays both the recorded measurement and the estimated state.

The remainder of this section describes the details of the Extended Kalman filter navigator responses to the recorded data. The pertinent characteristics are explained. This follows closely the format of the previous chapter.

Figure 5.1 compares the position estimate produced by the navigator with a dead-reckoning estimate. The dead reckoning plot was generated using the recorded vehicle

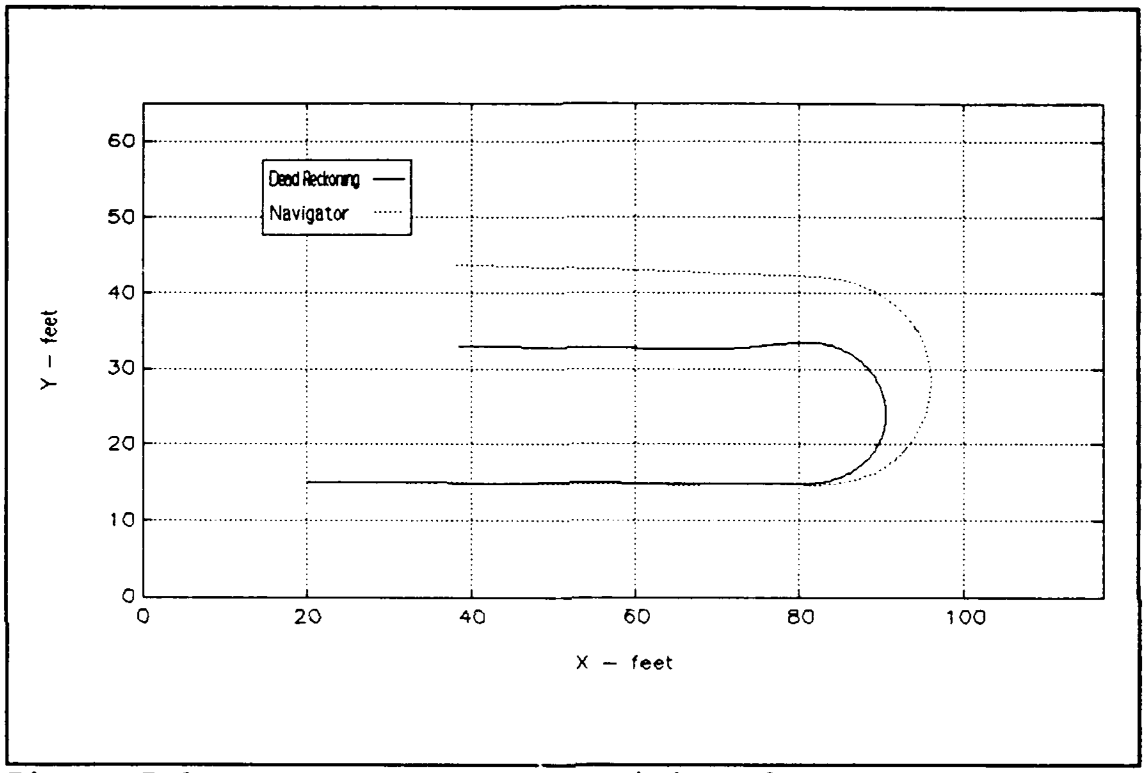

Figure 5.1 - NPS AUV II - X-Y position plot

speed and heading gyroscope measurements. No accurate record of the actual position of the vehicle during this mission is available. The borders of the graph correspond approximately to the dimensions of the pool. The initial position of the vehicle was approximately 20 feet from the short wall and 15 feet from the long wall. There is a significant difference between the dead reckoning plot and the navigator output which is attributed to the side slip, v, experienced by the vehicle in the turn, which is not taken into account in dead reckoning.

The estimated forward speed corresponds well to the speed log measurement as shown in Figure 5.2. The variations in the speed are caused by both the vehicle's speed controller which uses unfiltered speed log measurements in feedback and the dive plane commands, which induce drag. During the vehicle's 180 degree turn, which

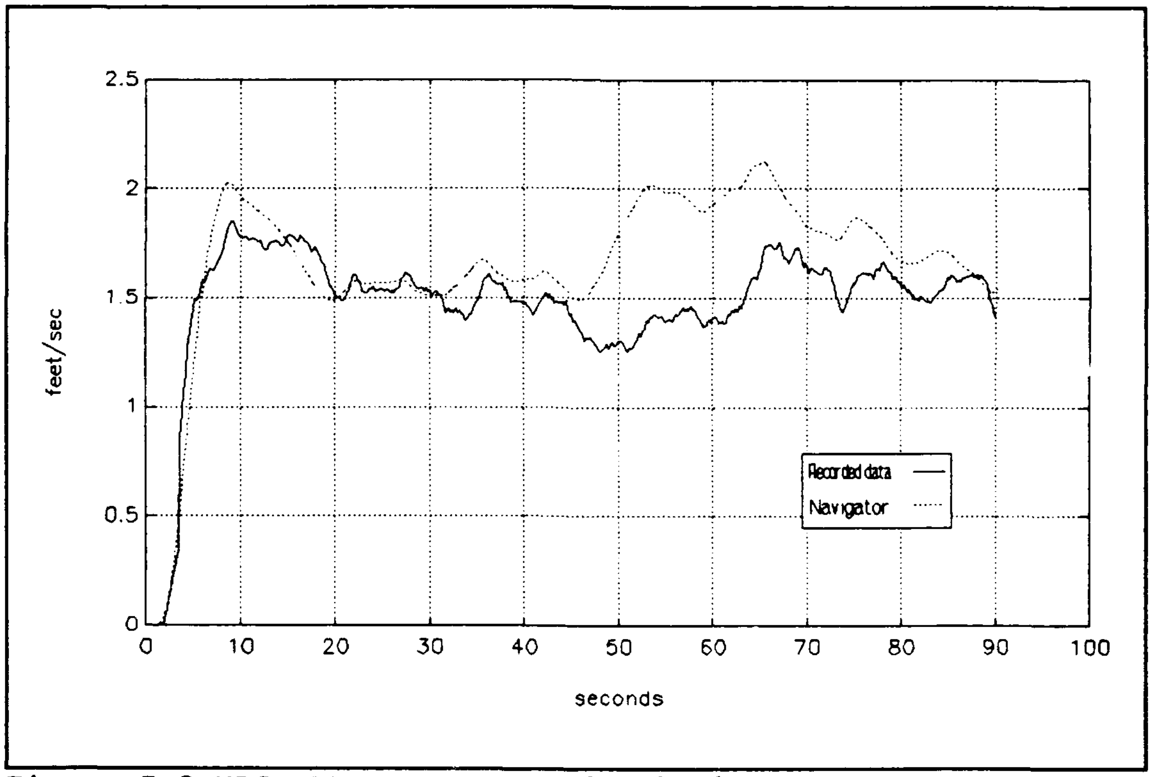

Figure 5.2 NPS AUV II - Forward velocity

started at approximately the 40 second mark, a significant difference between the measured speed and the estimated speed develops. The difference is due in part to errors in the model and in part to the side slip velocity, shown in

Figure 5.3. Side slip causes the paddle wheel to generate erroneous speed measurements.

The estimated side slip velocity, *v*, shown in Figure 5.3, approaches a minimum value of approximately -0.5 feet/second during the turn. There is no instrument on the vehicle to measure this quantity so there is no measurement for comparison. The oscillations present prior to the turn are due to the low initial vehicle speed, which makes the

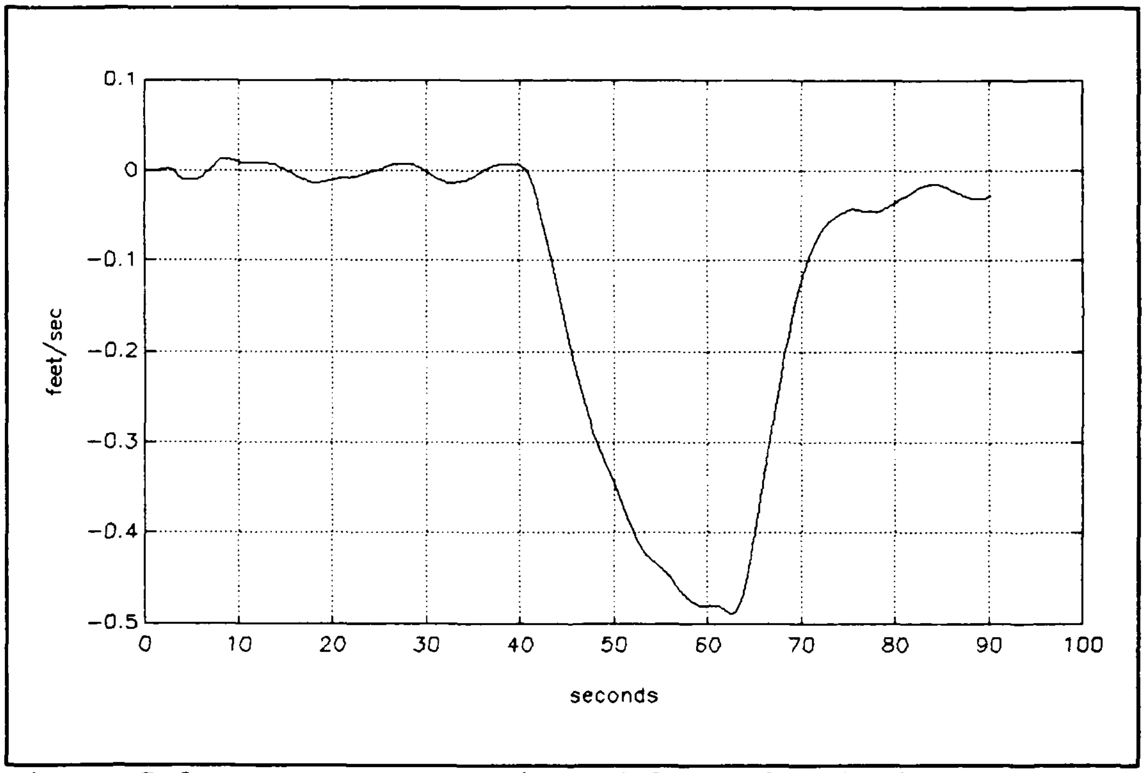

Figure 5.3 NPS AUV II - Estimated lateral velocity

vehicle itself underdamped in yaw, and the vehicle's heading controller which is also underdamped. During the turn, the estimated side slip approaches -0.5 feet/second and has not returned to a zero mean value by the end of the run. This latter fact results in the angled trajectory in the X-Y plane evident in the return-leg portion of Figure 5.1. While the estimated heading is very close to the measured heading used in the dead reckoning plot, the lateral velocity gives the vehicle an oblique path.

The body coordinate vertical velocity, *w*, does not have a significant effect on the position estimate because the pitch angles are relatively low; however, this state does

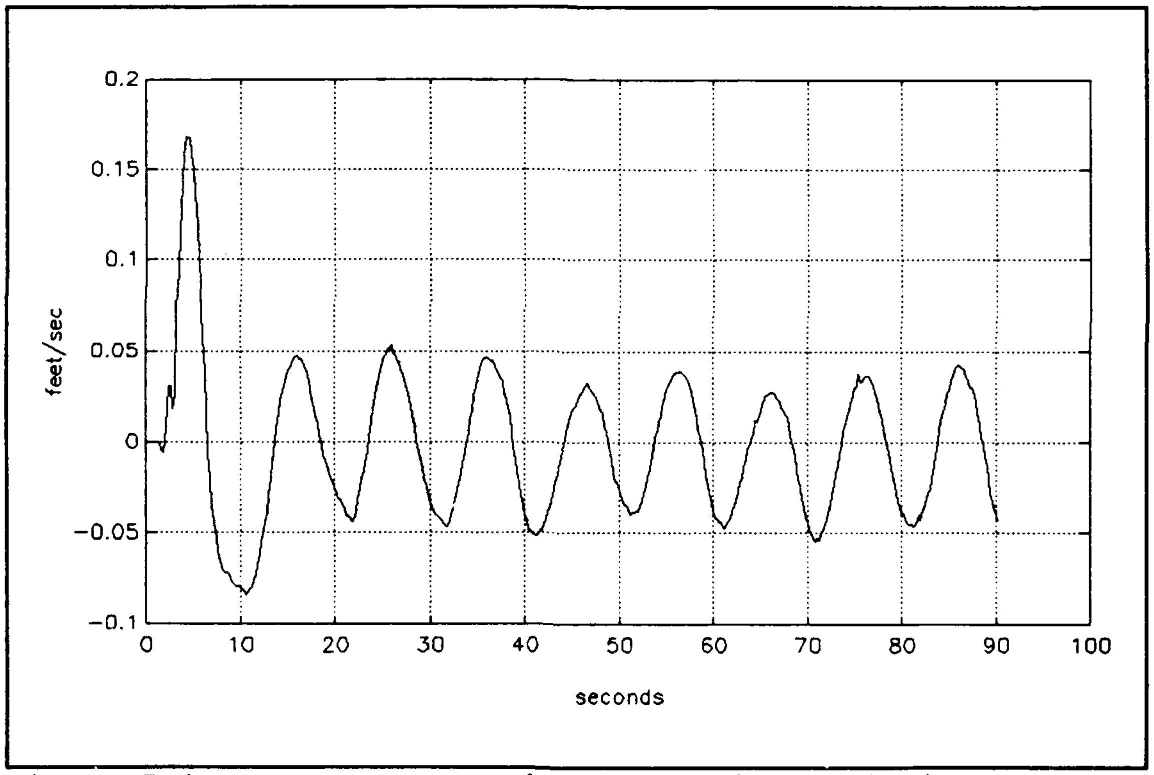

Figure 5.4 NPS AUV II - Estimated vertical velocity

affect the depth-rate measurement. As with side slip, there is no instrument to measure *w*. The relatively large spike occurring in the first few seconds is due to both the vehicle and the filter transient effects.

Comparison of the angular rate estimates shown in Figures 5.5 through 5.7 with those of the simulations indicates that the filter is not as sensitive to measurement noise as was indicated in simulation. This is largely because the data used in this study is relatively free of the noise present in other data sets.

As in the simulations, Figure 5.5 shows that the estimated and the measured roll rate, *p*, exhibit a high frequency oscillation when the vehicle is operating at low

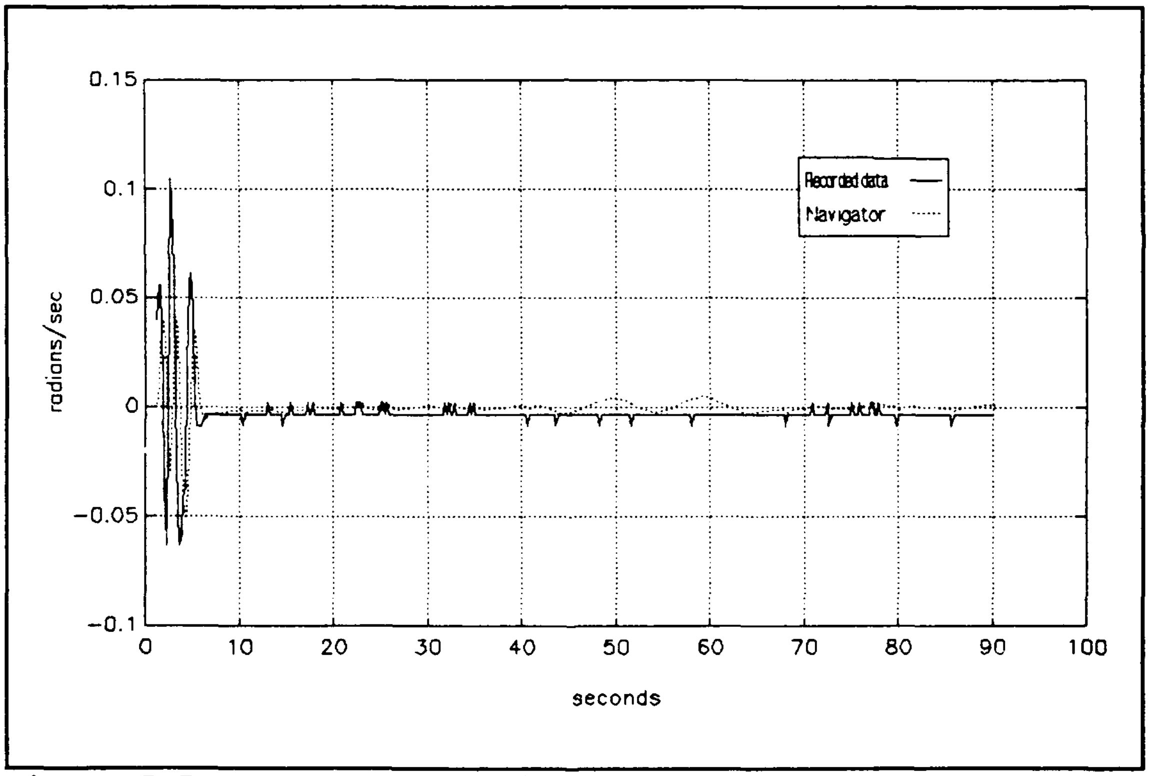

Figure 5.5 NPS AUV II - Roll rate

speed. As with the side slip, the vehicle's roll mode is underdamped at low speed but becomes more damped as the hydrodynamic forces on the vehicle increase with speed. The gyroscope measurement exhibits quantization noise to a significant degree. The obvious bias in the measurement is due largely to this effect. Additionally, the measurement gives no evidence of roll during the turn, although the navigator does estimate a rolling effect between approximately 45 and 65 seconds. This could be because the model is inaccurate or because the quantization noise is hiding the effect. Comparison with the roll angle measurement indicates that it is the latter.

Figure 5.6 shows the estimated and the measured pitch rate, *q*. As with the *w* estimate, both the vehicle and the filter transient effects produce the spike in the beginning of the plot. For all but the peak values, the navigator estimate corresponds closely to the measurement. The mismatch at the maximum values indicates that the model is in error. If it were just a bias in the measurement then the estimate would be low through the entire pitching motion.

The estimated yaw rate differs from the other angular rate estimates in that there is a period of time when the estimate differs significantly from the measurement. Figure 5.7 shows that this error occurs during the turn. Although quantization noise is evident in this figure, it is not the cause of the error as is hypothesized for the roll rate in Figure 5.5. The measured yaw rate reaches a maximum value in the turn while the estimate continues to increase. This

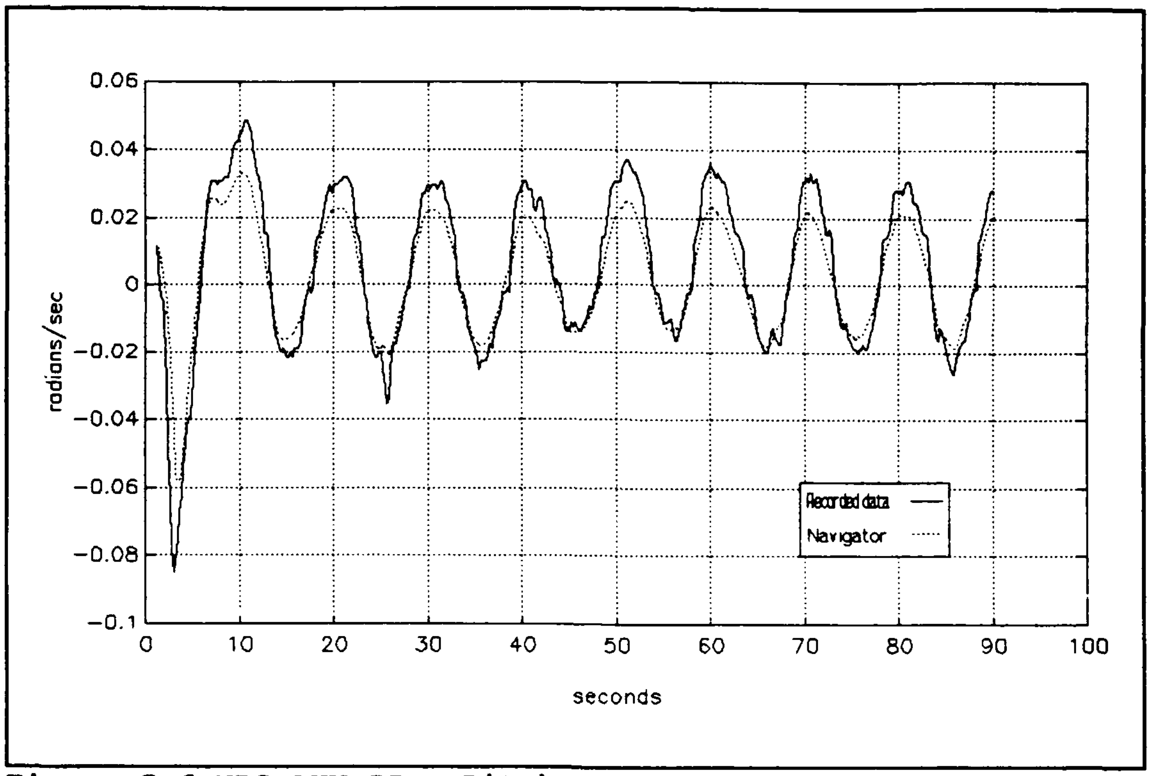

Figure 5.6 NPS AUV II - Pitch rate

effect can be attributed to physical phenomena occurring in the vehicle which is not included in the vehicle model, such as rudder stall. Because it uses a constant bias model for the rate gyroscope measurements, the navigator tends to use the bias to account for differences in the expected behavior and the measurements so the estimated bias varies more than was expected.

Figure 5.8 shows the estimated and the measured depth. There is excellent correlation between the measurement and the estimate. Also, because the depth signal is relatively free of noise, differentiating it to generate a depth-rate signal is entirely feasible.

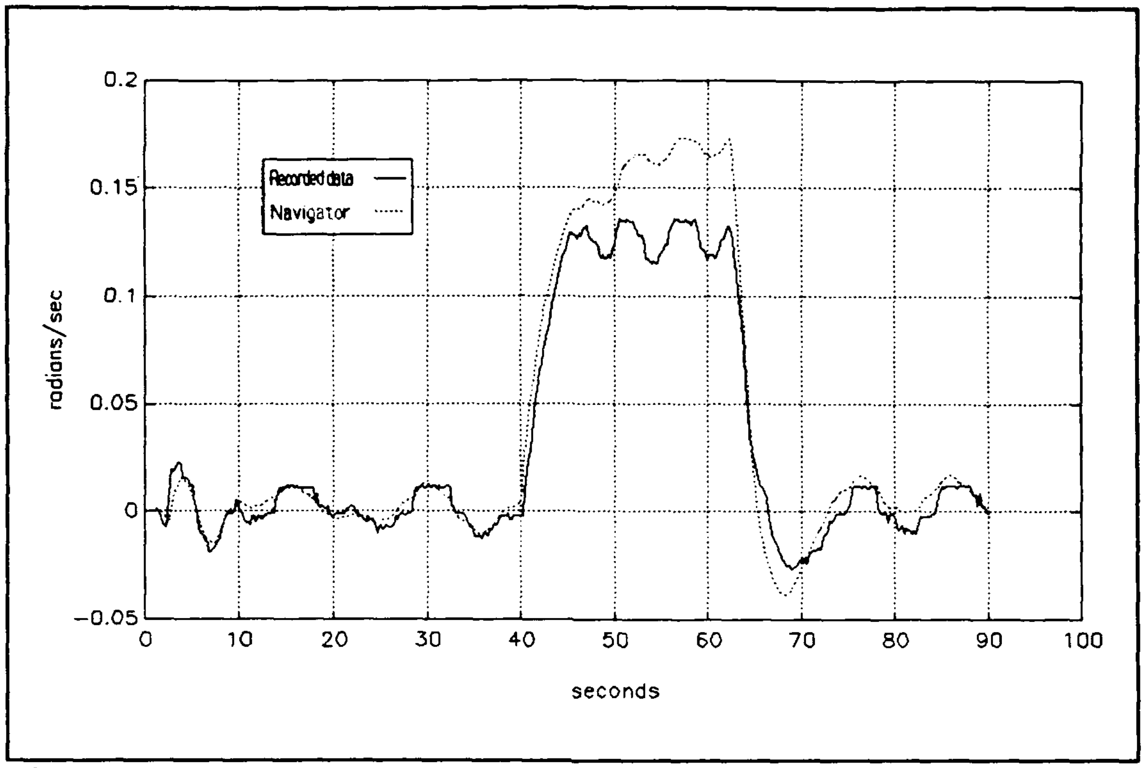

Figure 5.7 NPS AUV II - Yaw rate

The Eulerian attitude angles are the last vehicle states estimated by the filter. As previously mentioned, only the pitch angle measurement and the heading measurement are used in the navigator.

The roll measurement shows a significant, relatively constant bias as well as quantization noise. A comparison of the estimated and the measured roll angle, Φ in Figure 5.9 shows that the dynamics of the roll measurement do not appear to differ greatly from the estimated roll. The roll measurement (excluding the bias) corresponds to the estimated roll, especially in the turn. The absence of a corresponding roll rate measurement, from Figure 5.5,

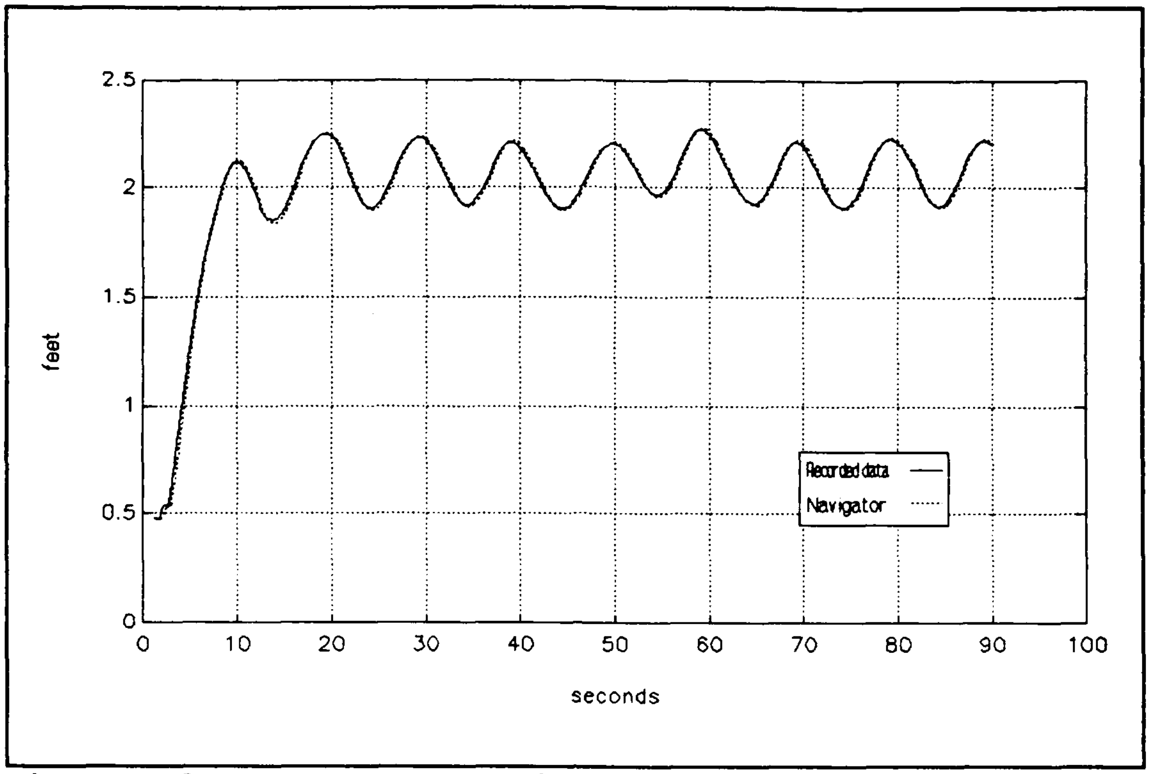

Figure 5.8 NPS AUV II - Depth

indicates that the roll motion experienced by the vehicle was not detected by the roll rate gyroscope and that measurement is in error.

Unlike the simulation studies, the estimated pitch converges quickly to the measured pitch, Θ. However, Figure 5.10 shows that there is a systematic estimation error in the pitch estimate. As opposed to the pitch rate measurement and estimate, the pitch angle estimate appears to underestimate the pitch throughout the cycle as though a bias error were present. As discussed above, this might be due to an error in the vehicle model or in the gyroscope model.

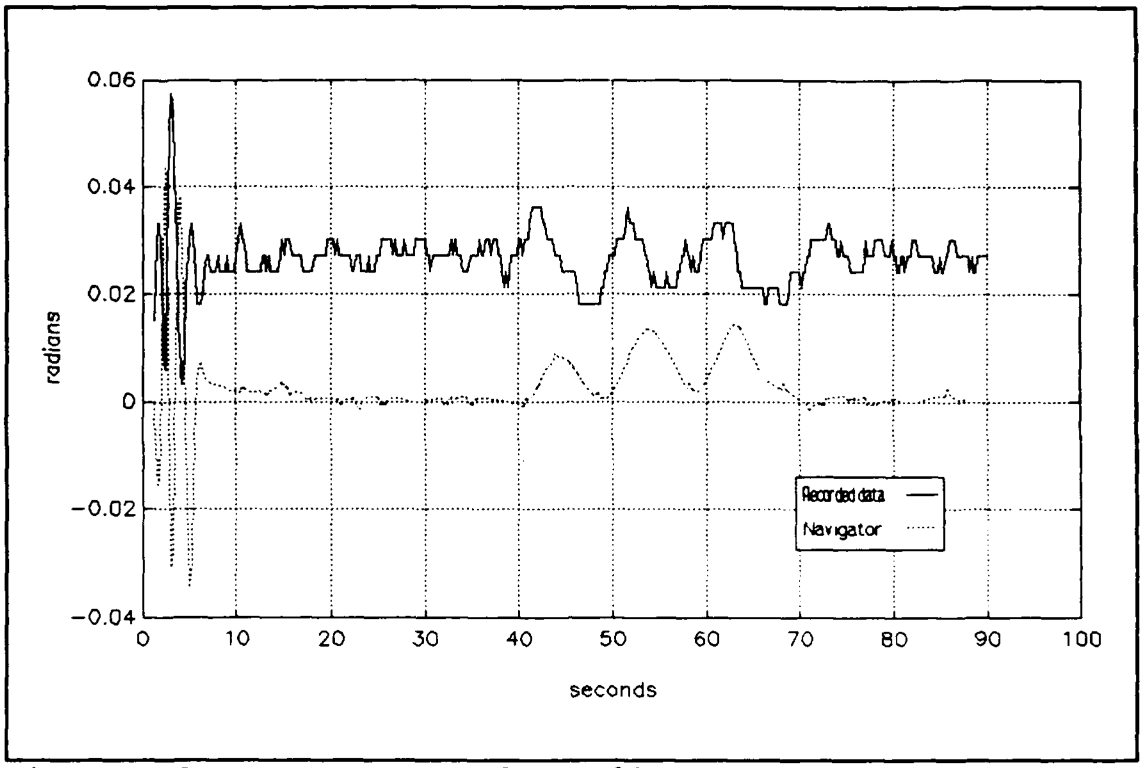

Figure 5.9 NPS AUV II - Euler roll angle

Figure 5.11 shows the estimated and the measured heading, Ψ. As previously mentioned, the heading gyroscope measurement is very clean and the Extended Kalman filter relies greatly on the accuracy of this measurement for heading estimation. While the scale of this figure prevents the heading prediction error from being seen, the mean square error is approximately of 0.015 radian. Additionally, the yaw rate error shown in Figure 5.7 is not evidenced in the heading estimate indicating that that error is insignificant for navigating a short-range mission.

Figures 5.12 through 5.14 show the three estimated rate gyroscope bias terms. As previously mentioned, the Extended

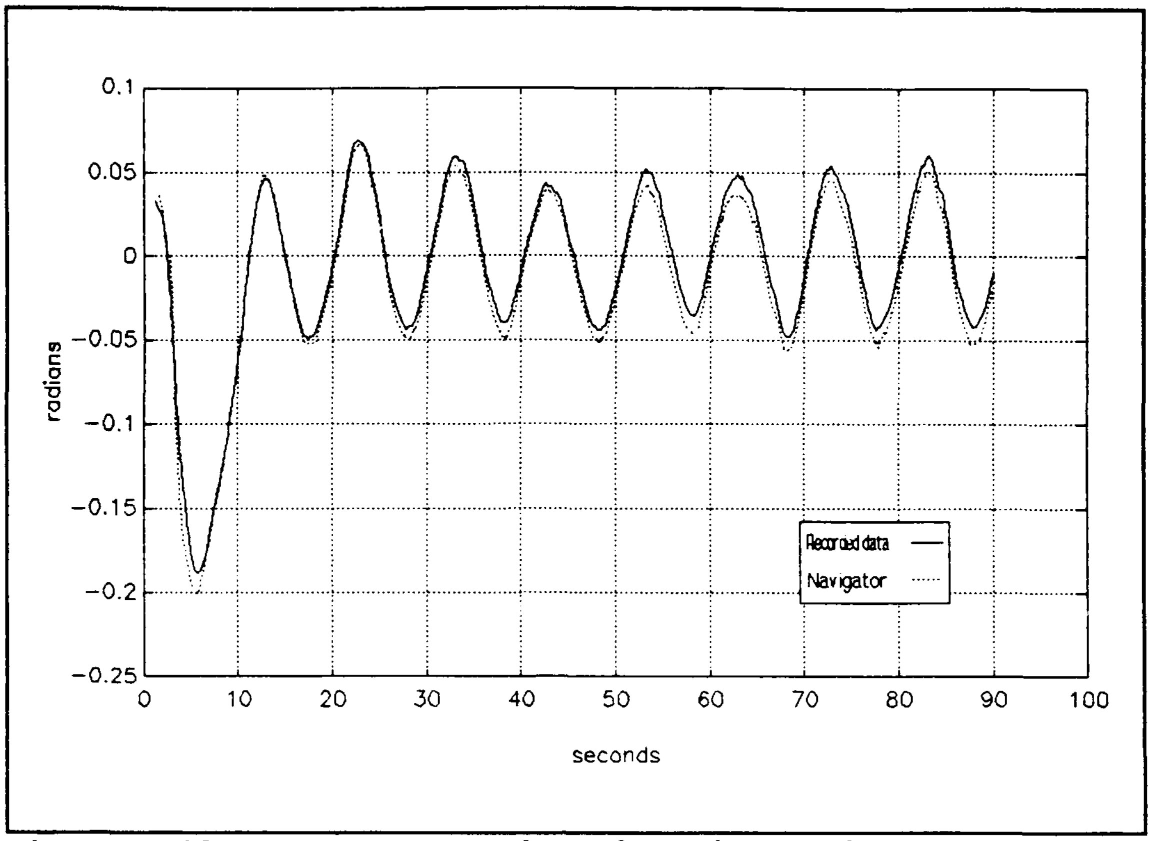

Figure 5.10 NPS AUV II - Euler elevation angle

Kalman filter tends to compensate for prediction inconsistencies by changing the bias terms. This is most apparent in Figure 5.15 which has a negative ramp-shaped section corresponding to the wedge shaped difference between the yaw rate measurement and estimate. It is also evident in the middle section of the roll rate bias estimate which corresponds to the vehicle's turn. Although none of these bias estimates is as invariant as in the simulation runs, they all appear to approach some non-zero mean value when the vehicle is running in a straight path.

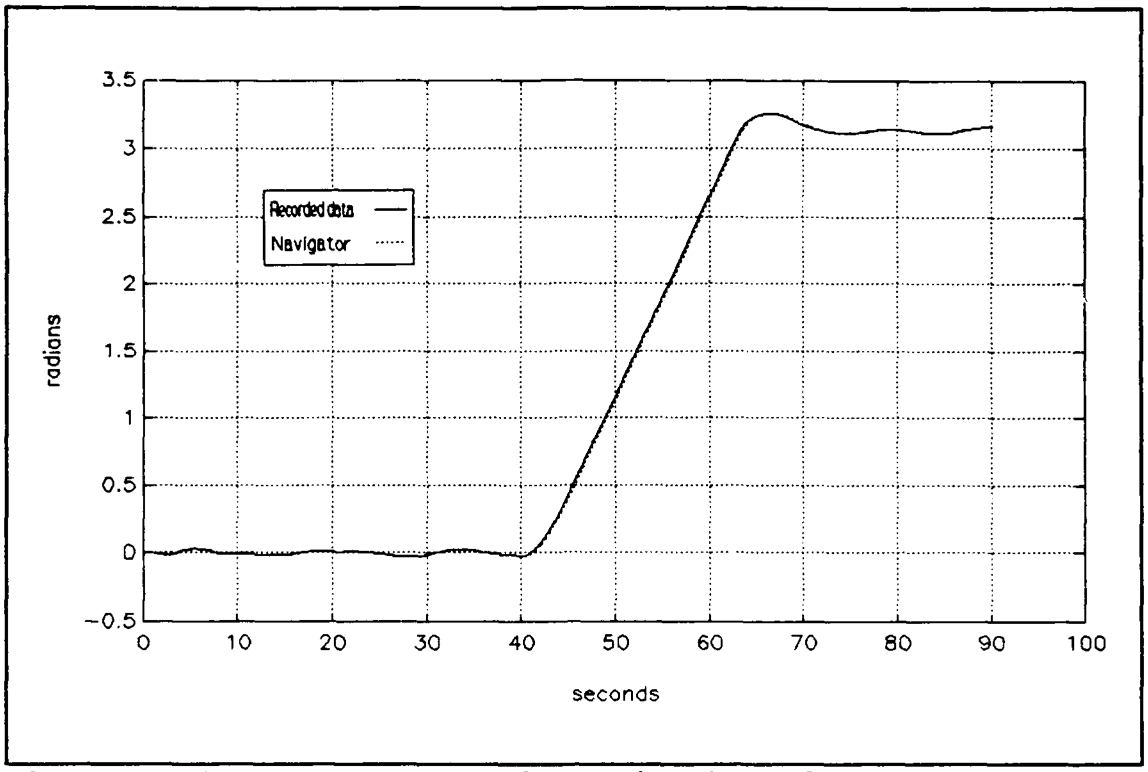

Figure 5.11 NPS AUV II - Euler azimuth angle

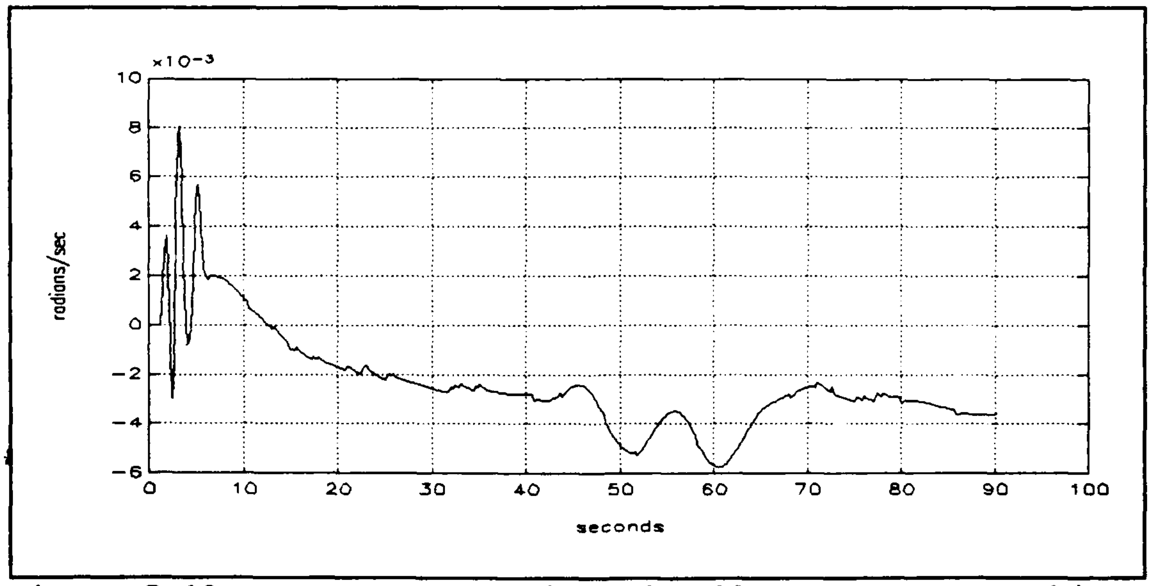

Figure 5.12 NPS AUV II - Estimated roll rate gyroscope bias

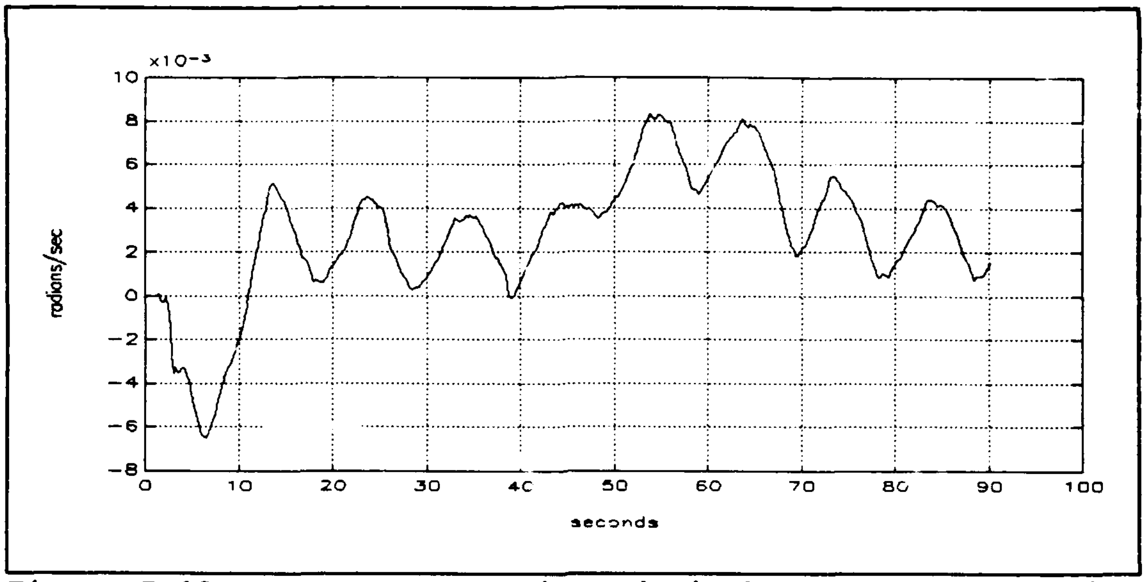

Figure 5.13 NPS AUV II - Estimated pitch rate gyroscope bias

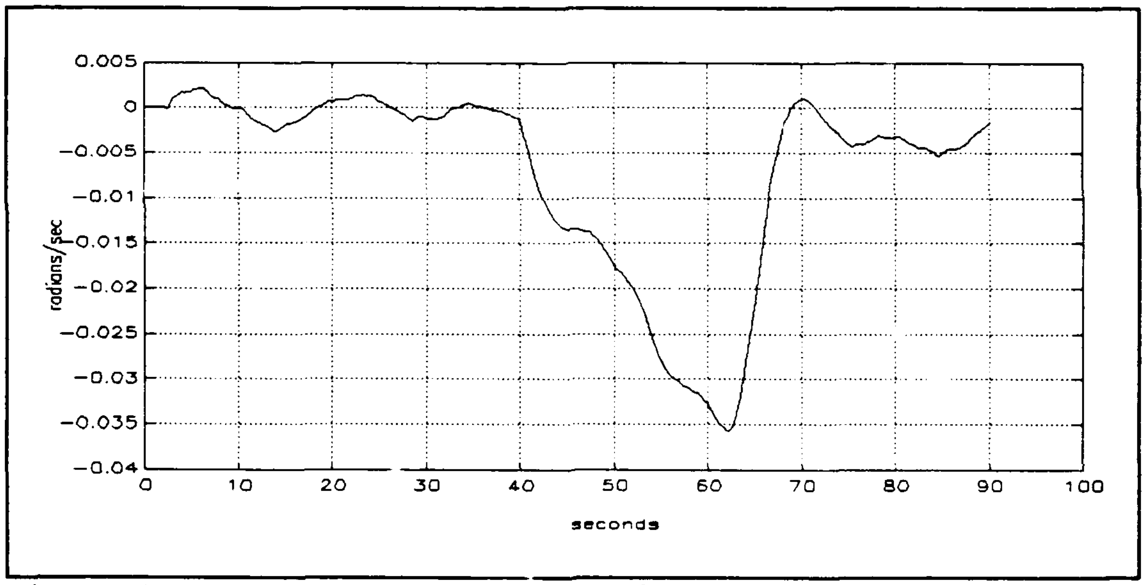

Figure 5.14 NPS AUV I - Estimated yaw rate gyroscope bias

In using the pitch gyroscope measurement, the filter was augmented to include a bias term for pitch. This is shown in Figure 5.15. As with the other bias terms, the estimate is not constant, but it does appear to evolve around some non-zero mean value. The variation in this bias is related

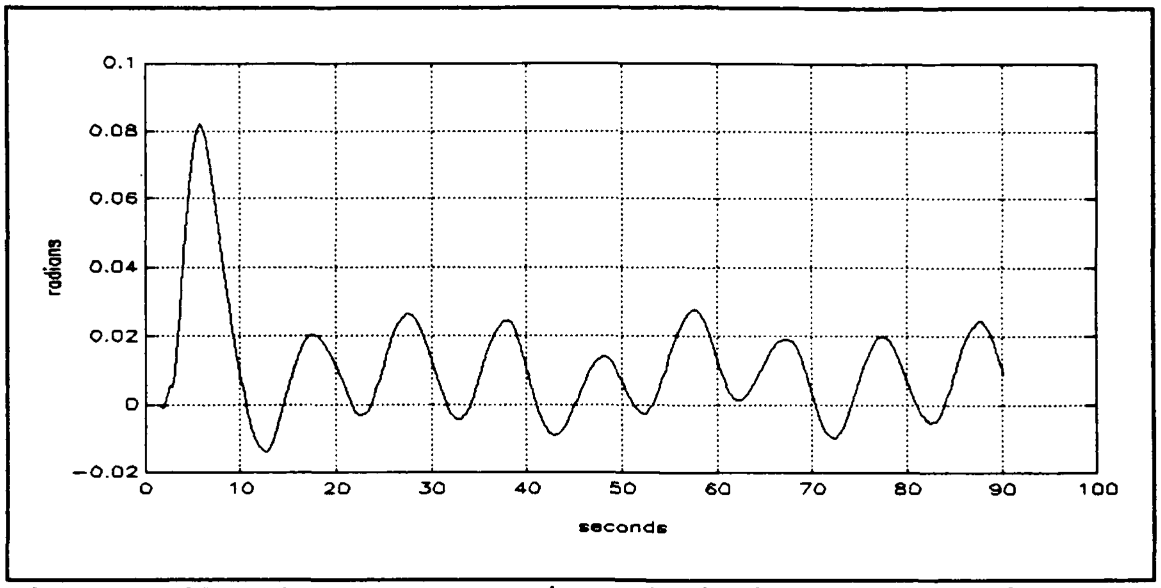

Figure 5.15 NPS AUV II - Estimated pitch gyroscope bias

to the time varying pitch angle prediction error made evident by the vehicle's porpoising.

The final measurement used by the filter is depth rate which is derived from the depth cell data by a simple difference equation. Figure 5.16 shows the estimated and the derived depth-rate measurement.

D. ALTERNATE SPEED ESTIMATION

To show that depth-rate information could be used to generate speed estimates without knowledge of vehicle dynamics, the depth cell data and the pitch gyroscope data have been processed by a simple second-order Extended Kalman filter which does not require a dynamic model of the AUV. For this filter, both the speed and the pitch are modeled as smoothly varying signals. The measurements used are pitch

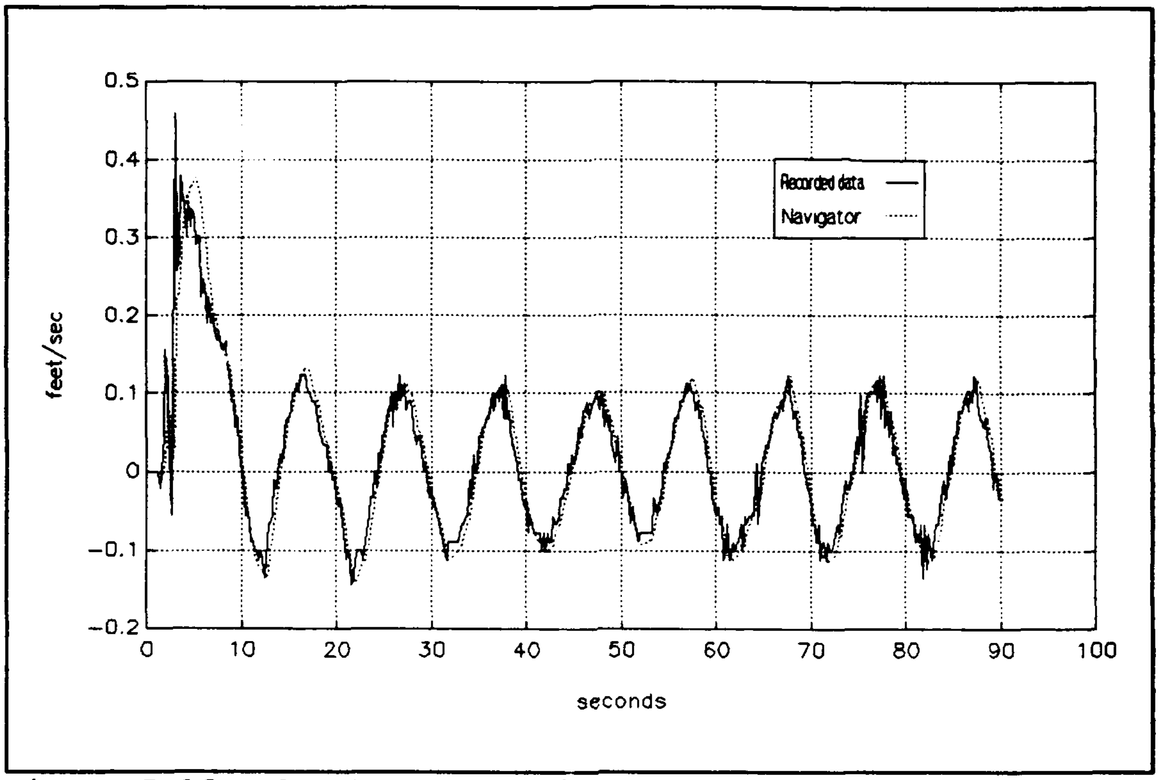

Figure 5.16 NPS AUV II - Depth rate

and depth rate. The measurement equation for depth rate is therefore nonlinear as explained in Chapter III. The system equations for this estimator are given by

$$\begin{bmatrix} \dot{u} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ \theta \end{bmatrix} + \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} \tag{5.2}$$

$$\underline{y} = \begin{bmatrix} \theta \\ \dot{z} \end{bmatrix} = \begin{bmatrix} \theta \\ -u \sin\theta \end{bmatrix} + \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \tag{5.3}$$

where υ and μ are white Gaussian noise signals. The linearized *C* matrix used by the Extended Kalman filter is therefore given by

$$\boldsymbol{C} = \begin{bmatrix} 0 & 1 \\ -\sin\theta & -\hat{u}\cos\theta \end{bmatrix}. \tag{5.4}$$

With this system of equations, Θ must be non-zero in order for the state to be observable.

With simulated data, these equations give perfect speed and pitch estimates even with a sinusoidal pitch measurement in spite of the linear approximation and the white noise assumption. A wide range of values could be used in $\boldsymbol{Q}$ and $\boldsymbol{R}$ without making the filter unstable. However, using actual data, in which other effects, including that of w, the body vertical velocity, are not taken into account, gives approximate u estimates while providing accurate Θ estimates. This is shown in Figures 5.17 and 5.18. Additionally, depth rate is not estimated as accurately as pitch as seen in Figure 5.19.

The stability of the estimator is very sensitive to the values chosen for $\boldsymbol{Q}$ and $\boldsymbol{R}$. These results were obtained with values of

$$\boldsymbol{Q} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{5.5}$$

and

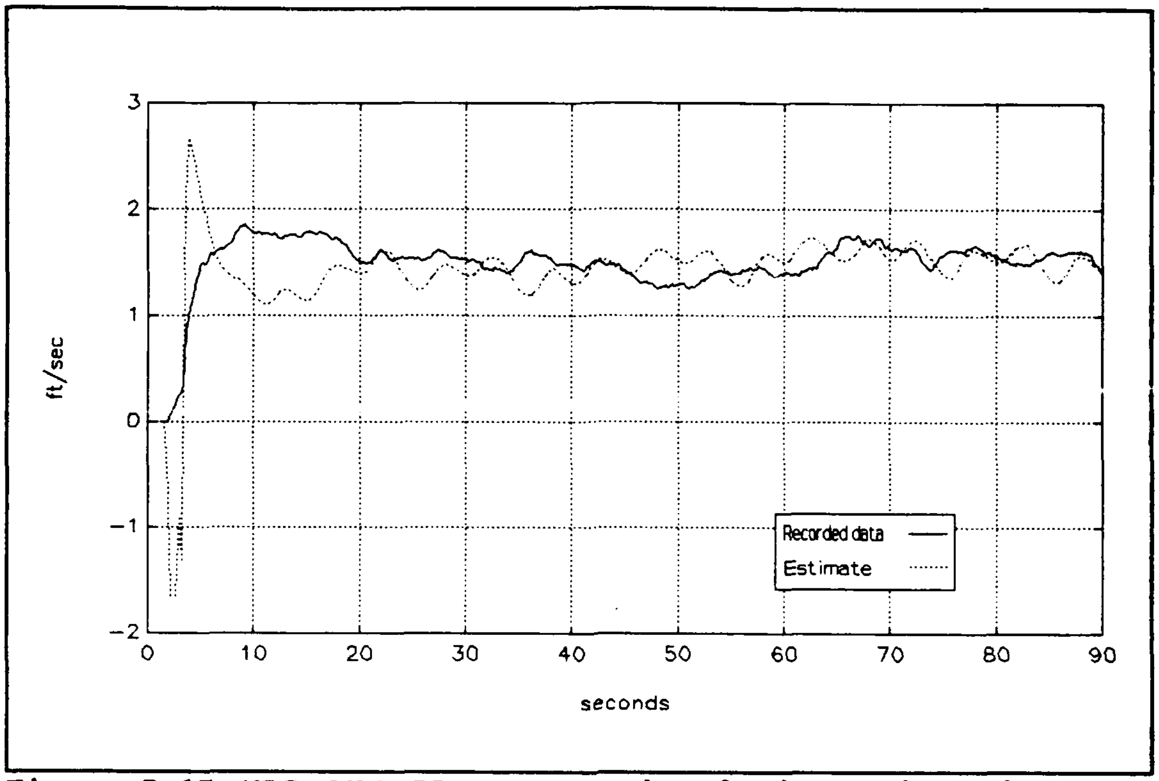

Figure 5.17 NPS AUV II - Forward velocity using alternate estimation model

$$\boldsymbol{R} = \begin{bmatrix} 0.001 & 0 \\ 0 & 0.01 \end{bmatrix}. \tag{5.6}$$

Other values yield a stable filter, but these values give the best compromise between accuracy of the three quantities estimated.

If $\boldsymbol{Q}(2,2)$ were made smaller, even if the other values were made larger to compensate, then the speed estimate would become unstable and the pitch estimate would not track the measured pitch; however, the depth-rate estimate would track even the noise in the depth-rate measurement.

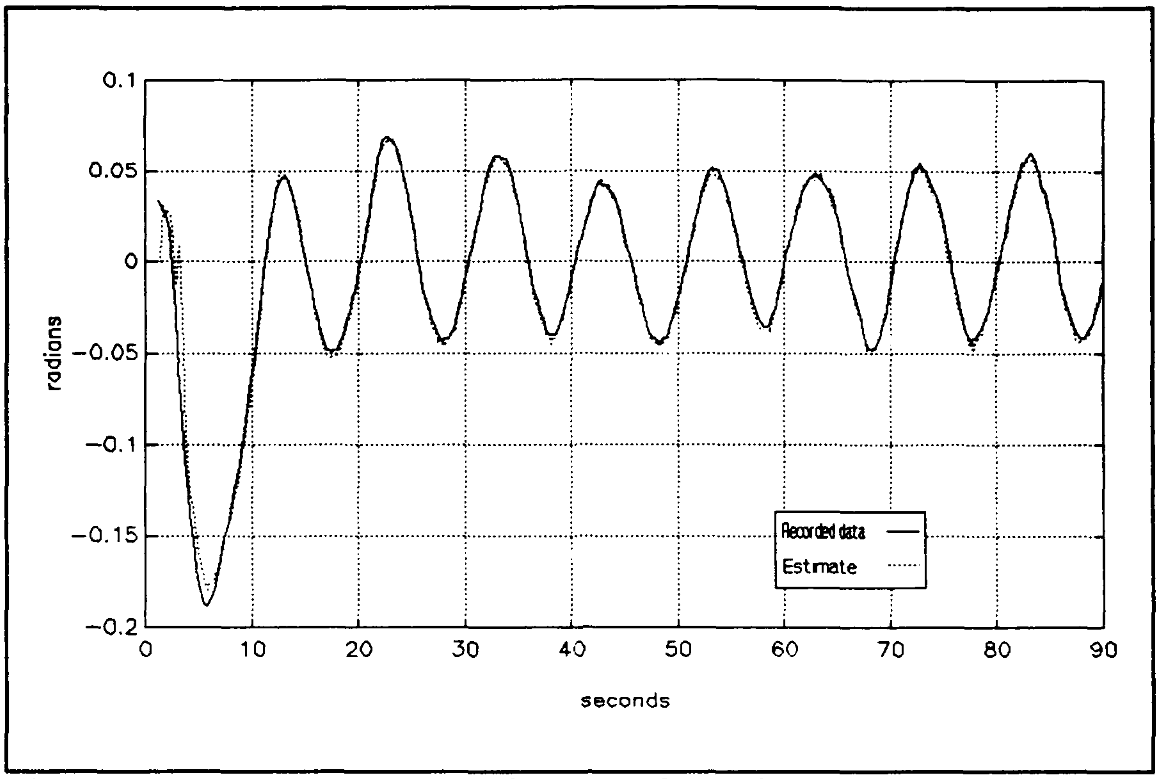

Figure 5.18 NPS AUV II - Pitch angle using alternate forward velocity estimation model

Although this second-order filter does give reasonable estimates, and is much simpler than the full navigator, its estimates do not have the accuracy that can be obtained by using information about the vehicle dynamics. However, if such information is not available, then this approach could be used to estimate speed without accelerometers or rate gyroscopes.

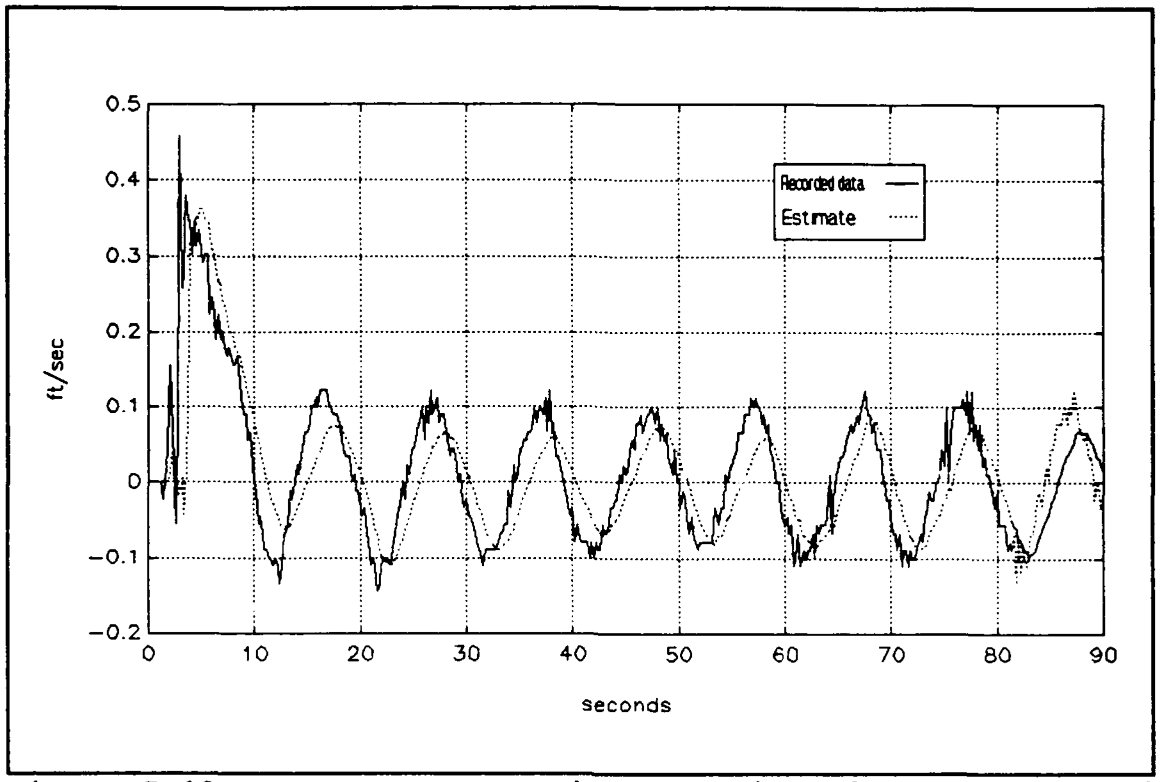

Figure 5.19 NPS AUV II - Depth rate using alternate forward velocity estimation model

VI. SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

A. SUMMARY

This thesis presents a study of model-based navigators for small autonomous underwater vehicles. The approach taken in the design and testing of the navigators included:

1. The development of linearized models for the SDV and the NPS AUV II based on nonlinear models which were already available.
2. The programming in C and MATLAB of both the nonlinear and linearized models for both vehicles.
3. The development and programming of nonlinear and linearized measurement equations for using accelerometers as attitude sensors.
4. The development and programming of nonlinear and linearized measurement equations for using depth rate to estimate forward speed.
5. Simulation studies for both vehicles using additive white Gaussian noise, an accelerometer dead zone model, and a gyroscope bias model.
6. Experimental studies with the NPS AUV II using recorded instrument data.

B. CONCLUSIONS

In this study, a navigator which uses knowledge of the vehicle dynamics is developed. In particular, speed estimation is obtained by combining the depth rate with inertial measurements.

The following conclusions can be drawn from the results of this study:

1. Position, attitude, velocity and angle rate estimation for both vehicles is possible.
2. Velocity and attitude estimation are possible through the use of nonlinear measurement equations.
3. Simulated instrument readings can be filtered and simulated gyroscope bias errors can be accurately predicted using the navigator.
4. Actual data can be filtered with simple instrument noise and error models.
5. A piecewise constant Extended Kalman gain is adequate for the NPS AUV II navigator.
6. The algorithm can be implemented in real time with a with a sampling rate of 5 Hz without overtaxing the processor.

C. RECOMMENDATIONS

Although manufacturers indicate that a small, accurate, laser-gyroscope-equipped, inertial measurement unit will be available in the near future, the algorithm explored in this thesis could be used in the interim for the NPS AUV II during pool missions. It is therefore recommended that the following be accomplished to facilitate implementation of the navigator:

1. Convert the navigator's main loop and other MATLAB functions to C and compile the entire program for use in the current NPS AUV II computer.

2. Accurately record the actual position of the vehicle in the pool for comparison with the navigator's position estimates, and update the hydrodynamic coefficients and filter parameters as required.
3. Use recorded gyroscope data to identify a better low-order model of the gyroscope measurement.
4. Use recorded data from each measurement source to identify a better low-order noise model.

If these can be accomplished then a study should be made to integrate differential Global Positioning System (GPS) data with the navigator, or to use differential GPS to reset the position estimate of the navigator periodically.

Finally, it is recommended that the current research in expert-system sonar processing be integrated with the navigator. As with GPS, the sonar could be used either to aid the navigator in real time, or to reset the position estimates periodically, depending on the sonar's accuracy and processing requirements. The sonar information could also be used to correct any unknown initial errors in the heading gyroscope which are not taken into account by the navigator.

APPENDIX A

Program Listing for SDV Simulation

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File    SIMUL9.M
%
%           This MATLAB Script file performs the SDV
%           navigator simulation.  It initializes the
%           variables and contains the main loop for the
%           navigator.
%           Velocities are in ft/sec; angles in radians
%
%   Calls   MKABMEX.MEX, C2D.M, KALM.M, MODEL.MEX
%           GETMEAS.M, MAKEMEAS.M
%
%   Modified 11 Oct 91
%
%   Ver.9   No speed log measurement
%           Previous versions used a different accelerometer
%           measurement concept and did not model
%           gyroscope errors
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Set initial state
Z         = 20;
rpm       = 500;
u0        = 0.30;
w0        = 0.0;
pitchrate = 0.0;
pitch     = 0.0;
heading   = pi/8;
phi0      = 0.0;
the0      = pitch;
psi0      = heading;
initial_state =
[u0;0;w0;0;pitchrate;0;0;0;Z;0;pitch;heading];

%  Set initial inputs
input0 = [0;0;0;rpm];

%  Set sample interval and run time
dt = 1.0;
Tf=100;
```

```
%  Set up plant and measurement noise covariance
%  matrices
%  (prepare to leave out X and Y terms from Kalman filter)
Q=zeros(13);
Q(1:10,1:10) = 0.01*eye(10);
Q(1,1) = .1;
Q(11:13,11:13) = 0.005*eye(3);

R=.05*eye(9);
R(6,6) = 10;

%  Set up initial estimate covariance matrices for 13 states
%  u,v,w,p,q,r,Z,phi,theta,psi plus
%  3 rate gyro bias states
Pold = eye(13);
Pold(8,8) = .1; Pold(9,9) = .1; Pold(10,10) = .1;

%  Initialize state vectors for simulation
%  simulate unknown vehicle state with random noise in IC
%  state = simulated vehicle
%  navig = Extended Kalman filter navigator

rand('normal')
g = 32.2;
kmax =Tf/dt;
state = zeros(12,kmax);
state(:,1) = initial_state + 0.03*rand(12,1);
navig = zeros(15,kmax);
navig(:,1) = [initial_state;0;0;0];

% shell for measurement matrix C:

C=[0 0 0 1 0 0 0 0 0 0 1 0 0; %          |p|
   0 0 0 0 1 0 0 0 0 0 0 1 0; %        ] =|q|+ biases,
   0 0 0 0 0 1 0 0 0 0 0 0 1; %          |r|
   0 0 0 0 0 0 1 0 0 0 0 0 0; %           depth
   0 0 0 0 0 0 0 0 0 0 0 0 0; %           depth dot
   0 0 0 0 0 0 0 0 0 0 0 0 0; %
   0 0 0 0 0 0 0 0 0 0 0 0 0; %        ] will be =-|R||g|
   0 0 0 0 0 0 0 0 0 0 0 0 0; %
   0 0 0 0 0 0 0 0 0 1 0 0 0];%          heading = Psi

%  disturbance input matrix (different from Gam)
Gam2=sign(Q);

meas(:,1)=zeros(9,1);

for (i=1:kmax)

    inputm=[0.2;0.20*cos(2*pi*i*dt/20);0;rpm];
    inputv=[0.2;0.20*cos(2*pi*i*dt/20);0;rpm];
```

```
% assign states required in C matrix relinearization
uh   = navig(1,i); wh = navig(3,i);
phih = navig(10,i);
theh = navig(11,i);
psih = navig(12,i);

% Recalculate C, Phi and Gam matrices around estimate of
% states.  For Zdot use abbreviated form of analytic
% equation

C(5,1) = -sin(theh);
C(5,3) =  cos(theh);
C(5,9) = -uh*cos(theh) - wh*sin(theh);


C(6,9)  = -g*cos(theh);
C(7,8)  = -g*cos(phih)*cos(theh);
C(7,9)  = -g*(-sin(theh)*sin(phih));
C(8,8)  = -g*(-sin(phih)*cos(theh));
C(8,9)  = -g*(-sin(theh)*cos(phih));

% in case accel is in dead zone
if (abs(meas(6,i))<.4)
    C(6,:)=zeros(C(6,:));
end;

% Setup the full 12 state re-linearized model
[a,b]=mkabmex(navig(1:12,i),inputm);

% delete from 'a' and 'b' elements having to do with
% X and Y to improve stablity of Extended Kalman filter
a = [a(1:6,1:6) , a(1:6,9:12);
      a(9:12,1:6), a(9:12,9:12)];
b = [b(1:6,:);
     b(9:12,:)];

% add gyro bias terms as constants
a = [a zeros(10,3);
    zeros(3,13)];
b = [b;zeros(3,4)];

% Dicretize the system
[Phi,Gam]=c2d(a,b,dt);

% Calculate Kalman gain
[K,Pnew] = kalm(Phi,Gam2,C,Pold,Q,R);
Pold=Pnew;

% Calculated next simulated vehicle state
state(:,i+1) = model(state(:,i),inputv,dt);
```

```
    % Generate simulated instrument measurements
    [rategyro,Z,Zdot,accel,hdg]=getmeas3(state(:,i+1),Z,dt);
    meas(:,i+1) = [rategyro;Z;Zdot;accel;hdg];

    % Calculate next navigator prediction
    navig(1:12,i+1) = model(navig(1:12,i),inputm,dt);
    navig(13:15,i+1) = navig(13:15,i);

    % Generate predicted measurements
    modmeas = makmeas2(navig(:,i+1),navig(:,i),dt);

    % Calculate correction using Kalman gain
    esterr(:,i) = K*(meas(:,i+1)-modmeas);

    % Correct prediction.  Zeros are to account for X and Y
    % not being included in the Extended Kalman filter
    navig(:,i+1)=...
         navig(:,i+1)+[esterr(1:6,i);0;0;esterr(7:13,i)];

end;

% Generate graphical output

t=0:dt:Tf;

!del sdvxy.met
plot(state(7,:),state(8,:),navig(7,:),navig(8,:));grid;
xlabel('X - feet');ylabel('Y - feet');pause
meta sdvxy

!del sdvu.met
plot(t,state(1,:),t,navig(1,:));grid
xlabel('seconds');ylabel('feet/sec');meta sdvu;pause

!del sdvv.met
plot(t,state(2,:),t,navig(2,:));grid
xlabel('seconds');ylabel('feet/sec');meta sdvv;pause

!del sdvw.met
plot(t,state(3,:),t,navig(3,:));grid
xlabel('seconds');ylabel('feet/sec');meta sdvw;pause

!del sdvp.met
plot(t,state(4,:),t,navig(4,:));grid
xlabel('seconds');ylabel('radians/sec');meta sdvp;pause

!del sdvq.met
plot(t,state(5,:),t,navig(5,:));grid
xlabel('seconds');ylabel('radians/sec');meta sdvq;pause

!del sdvr.met
```

```
plot(t,state(6,:),t,navig(6,:));grid
xlabel('seconds');ylabel('radians/sec');meta sdvr;pause

!del sdvz.met
plot(t,state(9,:),t,navig(9,:));grid
xlabel('seconds');ylabel('feet');meta sdvz;pause

!del sdvtheta.met
plot(t,state(11,:),t,navig(11,:));grid
xlabel('seconds');ylabel('radians');meta sdvtheta;pause

!del sdvpsi.met
plot(t,state(12,:),t,navig(12,:));grid
xlabel('seconds');ylabel('radians');meta sdvpsi;pause

!del sdvpbias.met
plot(t,navig(13,:));grid
xlabel('seconds');ylabel('radians/sec');meta sdvpbias;pause

!del sdvqbias.met
plot(t,navig(14,:));grid
xlabel('seconds');ylabel('radians/sec');meta sdvqbias;pause

!del sdvrbias.met
plot(t,navig(15,:));grid
xlabel('seconds');ylabel('radians/sec');meta sdvrbias;pause
```

```
/**********************************************
 *
 *  File   MODEL.C
 *
 *         C language source code for MODEL.MEX.  Must be
 *         compiled within MEX subdirectory of MATLAB.
 *         It is used like a MATLAB function:
 *
 *             function  state = model(oldstate,inputs,dt)
 *
 *         This code was translated from the DSL (FORTRAN
 *         based) code in R. Boncal's MS Thesis, 1987,
 *         NPS, Monterey, CA.  It is based on modified
 *         equations of motion from NSRDC Report 2510
 *         June, 1967.
 *
 *  Called by SIMUL9.M
 *
 **********************************************/


#include <math.h>
#include <stdio.h>
#include "modelprm.h"
#include "cmex.h"


int   model( double *state, double *oldstate, double
*inputs, double *dt );

/*
 *  This is the section of code recognized by MATLAB
 *  It calls function MODEL
 */
void  user_fcn( int nlhs, Matrix *plhs[], int nrhs, Matrix
*prhs[] )
{
   double  *oldstate, *inputs, *dt, *state;

   if (nrhs != 3)
      mex_error("Must be three input arguments.");
   if (nlhs != 1)
      mex_error("Must be one output argument.");
   if (ROWS_IN(0) != 12  ||  COLS_IN(0) != 1)
      mex_error("Previous state vector not correct size.");
   if (ROWS_IN(1) != 4  ||  COLS_IN(1) != 1)
      mex_error("Input vector not correct size.");
   if (ROWS_IN(2) != 1  ||  COLS_IN(2) != 1)
      mex_error("Time interval must be a scalar.");

   plhs[0] = create_matrix(12,1,REAL);
   state = OUT(0);
   oldstate = IN(0);
   inputs = IN(1);
   dt = IN(2);

   model(state,oldstate,inputs,dt);
}

/* This code is the nonlinear model of the SDV */

int  model( double *state, double *oldstate, double *inputs,
double *dt )
{
   int      j, k;
   double   u, v, w, p, q, r, phi, theta, psi;
   double   dr, ds, db, rpm, delt;
   double   mass, latyaw, norpit, re, term0;
   double   signu, signn, eta, cd0, ct, ct1, eps, xprop;
   double   ucf[4], fp[6], f[12];
   double   tmp1, tmp2, tmp3, tmp4;
   double   cos_theta, sin_theta, tan_theta;
   double   cos_phi, sin_phi, cos_psi, sin_psi;

   u     = oldstate[0];
   v     = oldstate[1];
   w     = oldstate[2];
   p     = oldstate[3];
   q     = oldstate[4];
   r     = oldstate[5];
   phi   = oldstate[9];
   theta = oldstate[10];
   psi   = oldstate[11];

   dr  = inputs[0];
   ds  = inputs[1];
   db  = inputs[2];
   rpm = inputs[3];

   delt = *dt;

   latyaw = norpit = 0.0;
   mass = weight/g;
   re = u*l/nu;

   signu = FSIGN(u);
   signn = FSIGN(rpm);
   if (fabs(u) < xltest)
      u = xltest;
   eta = 0.012*rpm/u;
   re = u*l/nu;
   cd0 = 0.00385 + 1.296e-17 * (re - 1.2e7)*(re - 1.2e7);
   ctl = 0.008*l*l/a0;
   ct  = ctl*eta*fabs(eta);
   eps = -1.0+signn/signu*(sqrt(ct+1.0)-1.0)
            /(sqrt(ctl+1.0)-1.0);
   xprop = cd0*(eta*fabs(eta) - 1.0);


/*
 *    calculate the drag force,
 *    integrate the drag over the vehicle
 *    integrate using 4 terms
 */

   for (k=0; k<4; ++k) {
      tmp1 = v+g4[k]*r*l;
      tmp2 = w-g4[k]*q*l;
      ucf[k] = sqrt(tmp1*tmp1 + tmp2*tmp2);
      if(1.0e-10 <= ucf[k]) {
         term0  = ((rho/2.0)*(cdy*hh[k]*tmp1*tmp1
                    + cdz*br[k]*tmp2*tmp2)) *gk4[4]*l/ucf[k];
         latyaw += term0*tmp1;
         norpit += term0*tmp2;
        }
    }

/*
 *    force equations
 */

/*
 *    common sub-expressions
 */

   tmp1 = (rho/2.0)*l*l;
   tmp2 = tmp1*l;
   tmp3 = tmp2*l;
   tmp4 = tmp3*l;
   cos_theta = cos(theta);
   sin_theta = sin(theta);
   tan_theta = sin_theta/cos_theta;
   cos_phi = cos(phi);
   sin_phi = sin(phi);
   cos_psi = cos(psi);
   sin_psi = sin(psi);

/*
 *    longitudinal force
 */

fp[0] = mass*v*r - mass*w*q + mass*xg*q*q
      + mass*xg*r*r - mass*yg*p*q - mass*zg*p*r
```

```
        + tmp3*(xpp*p*p+xqq*q*q + xrr*r*r+xpr*p*r)
        + tmp2*(xwq*w*q+xvp*v*p+xvr*v*r
        + u*q*(xqds*ds+xqdb*db)+ xrdr*u*r*dr)
        + tmp1*(xvv*v*v+xww*w*w + xvdr*u*v*dr
        + u*w*(xwds*ds+xwdb*db) + u*u*(xdsds*ds*ds+xdbdb*db*db
        + xdrdr*dr*dr)) - (weight -boy)*sin_theta
        + tmp2*xqdsn*u*q*ds*eps
        + tmp1*(xwdsn*u*w*ds+xdsdsn*u*u*ds*ds)*eps
        + tmp1*u*u*xprop;


/*
 *     lateral force
 */

fp[1] = -mass*u*r - mass*xg*p*q + mass*yg*r*r - mass*zg*q*r
        + tmp3*(ypq*p*q + yqr*q*r)+tmp2*(yp*u*p +
        yr*u*r + yvq*v*q + ywp*w*p + ywr*w*r) + tmp1*
        (yv*u*v + yvw*v*w +ydr*u*u*dr) -latyaw +(weight-boy)*
        cos_theta*sin_phi+mass*w*p+mass*yg*p*p;

/*
 *     normal force
 */

fp[2] = mass*u*q - mass*v*p - mass*xg*p*r - mass*yg*q*r +
        mass*zg*p*p + mass*zg*q*q + tmp3*
        (zpp*p*p+zpr*p*r + zrr*r*r) + tmp2*(zq*u
        *q+zvp*v*p + zvr*v*r) +tmp1*(zw*u*w
        + zvv*v*v+u*u*(zds*ds+zdb*db))-
        norpit+(weight-boy)*cos_theta*cos_phi
        +tmp2*zqn*u*q*eps +tmp1*(zwn*u*w +zdsn*
        u*u*ds)*eps;

/*
 *     roll force
 */

fp[3] = -iz*q*r +iy*q*r -ixy*p*r +iyz*q*q -iyz*r*r +ixz*p*q+
        mass*yg*u*q -mass*yg*v*p -mass*zg*w*p+tmp4*(kpq*
        p*q + kqr*q*r) +tmp3*(kp*u*p +kr*u*r + kvq*v*q +
        kwp*w*p + kwr*w*r) +tmp2*(kv*u*v + kvw*v*w) +
        (yg*weight - yb*boy)*cos_theta*cos_phi - (zg*weight -
        zb*boy)*cos_theta*sin_phi + tmp3*kpn*u*p*eps+
        tmp2*u*u*kprop +mass*zg*u*r;

/*
 *     pitch force
 */

fp[4] = -ix*p*r +iz*p*r +ixy*q*r -iyz*p*q -ixz*p*p +ixz*r*r-
```

```
        mass*xg*u*q + mass*xg*v*p + mass*zg*v*r - mass*zg*w*q
        + tmp4*(mpp*p*p +mpr*p*r +mrr*r*r) +
        tmp3*(mq*u*q + mvp*v*p + mvr*v*r) +
        tmp2*(mw*u*w+mvv*v*v+u*u*(mds*ds+mdb*db)) + norpit -
        (xg*weight-xb*boy)*cos_theta*cos_phi+tmp3*mqn*u*q*eps
        + tmp2*(mwn*u*w+mdsn*u*u*ds)*eps-
        (zg*weight-zb*boy)*sin_theta;

/*.P*/
/*
 *     yaw force
 */

fp[5] = -iy*p*q +ix*p*q +ixy*p*p -ixy*q*q +iyz*p*r -ixz*q*r-
        mass*xg*u*r + mass*xg*w*p - mass*yg*v*r + mass*yg*w*q
        + tmp4*(npq*p*q + nqr*q*r) +tmp3*(np*u*p+
        nr*u*r + nvq*v*q +nwp*w*p + nwr*w*r) +tmp2*(nv*
        u*v + nvw*v*w + ndr*u*u*dr) - latyaw + (xg*weight -
        xb*boy)*cos_theta*sin_phi+(yg*weight)*sin_theta
        +tmp2*u*u*nprop-yb*boy*sin_theta;

/*
 *     now compute the f(0-5) functions
 */

     for (j=0; j<6; ++j)
        for (f[j]=0.0,k=0; k<6; ++k)
           f[j] += xmminv[j][k]*fp[k];

/*
 *     the last six equations come from the kinematic
 *     relations
 */

/*
 *     inertial position rates f(6-8)
 */

f[6] =  u*cos_psi*cos_theta + v*(cos_psi*sin_theta*
        sin_phi - sin_psi*cos_phi) + w*(cos_psi*sin_theta*
        cos_phi + sin_psi*sin_phi);

f[7] =  u*sin_psi*cos_theta + v*(sin_psi*sin_theta*
        sin_phi + cos_psi*cos_phi) + w*(sin_psi*sin_theta*
        cos_phi - cos_psi*sin_phi);

f[8] = -u*sin_theta +v*cos_theta*sin_phi
         +w*cos_theta*cos_phi;

/*
 *     euler angle rates f(9-11)
 */

   f[9] = p + q*sin_phi*tan_theta + r*cos_phi*tan_theta;
   f[10] = q*cos_phi - r*sin_phi;
   f[11] = q*sin_phi/cos_theta + r*cos_phi/cos_theta;

/*
 *    Simpson's rule integration
 */

   for (j=0; j<12; j++)
      state[j] = oldstate[j] + delt * f[j];

   return  0;
}
```

```
/*********************************************
 *
 *  File   MKABMEX.C
 *
 *         C language source code for MKABMEX.MEX  Must be
 *         compiled within MEX subdirectory of MATLAB.
 *         It is used like a MATLAB function:
 *
 *             function  [A,B] = mkABmex(state,inputs)
 *
 *         This program makes a linearized model in MATLAB
 *         for the SDV around some [x0] and [u0] which are
 *         input parameters.  The vehicle mass matrix and
 *         hydrodynamic coefficients are in "modelprm.h"
 *
 *         This code is based on Taylor series linearization
 *         of the equations of motion in MODEL.C
 *
 *  Called by SIMUL9.M
 *
 *********************************************/


#include <math.h>
#include "tcmex.h"
#include "modelprm.h"

void  makeab( double *A, double *B, double *state, double
*inputs);

/*
 * Code recognized by MATLAB. It calls MAKEAB
 */

void user_fcn( int nlhs, Matrix *plhs[], int nrhs, Matrix
*prhs[] )
{
 double *state, *inputs, *a, *b;

 if (nrhs != 2)
  mex_error("Must be two input arguments");
 if (nlhs != 2)
  mex_error("Must be two output arguments");
 if (ROWS_IN(0) != 12 || COLS_IN(0) != 1)
  mex_error("Initial state vector must have 12 states");
 if (ROWS_IN(1) != 4  || COLS_IN(1) != 1)
  mex_error("Input vector must have 4 inputs");

 plhs[0] = create_matrix(12,12,REAL);
 plhs[1] = create_matrix(12,4,REAL);
 a = OUT(0);
 b = OUT(1);
 state = IN(0);
 inputs = IN(1);

 makeab(a,b,state,inputs);
}
/*
 * This code generates the linearized A and B matrices
 * from the partial derviatives of the equations of
 * motion for the vehicle.
 */

void makeab( double *A, double *B, double *state, double
*inputs )
{
 int    i, j, k;
 double a[12][12], b[6][4];
 double aa[12][12],bb[12][4];
 double u0, v0, w0, p0, q0, r0, phi0, theta0, psi0;
 double dr, ds, db, rpm;
 double mass, latyaw, norpit, re, term0;
 double eta, cdo, ct, ctl, eps, xprop;
 double  tmp1, tmp2, tmp3, tmp4, tmp5;
 double  cos_theta, sin_theta, tan_theta;
 double  cos_phi, sin_phi, cos_psi, sin_psi;


/* assign some common variable names */


u0 = state[0];
v0 = state[1];
w0 = state[2];
p0 = state[3];
q0 = state[4];
r0 = state[5];
phi0 = state[9];
theta0 = state[10];
psi0 = state[11];

dr = inputs[0];
ds = inputs[1];
db = inputs[2];
rpm = inputs[3];

cos_theta = cos(theta0);
sin_theta = sin(theta0);
tan_theta = tan(theta0);
cos_phi   = cos(phi0);
sin_phi   = sin(phi0);
cos_psi   = cos(psi0);
```

```
sin_psi    = sin(psi0);

if (theta0 == 0.0) { sin_theta = 0.0; tan_theta = 0;}
if (phi0 == 0.0) sin_phi = 0.0;
if (psi0 == 0.0) sin_psi = 0.0;

/* linear propulsion model */

eta = 0.012*rpm/u0;
re = u0*l/nu;
cdo = 0.00385+(1.296e-17)*(re-1.2e7)*(re-1.2e7);
ct = 0.008*l*l*eta*abs(eta)/a0;
ctl = 0.008*l*l/a0;
eps = -1+(sqrt(ct+1)-1)/(sqrt(ctl+1)-1);
xprop = cdo*(eta*abs(eta)-1);

/* assign commonly used values */

tmp1 = rho/2*l;
tmp2 = tmp1*l;
tmp3 = tmp2*l;
tmp4 = tmp3*l;
tmp5 = tmp4*l;
mass = weight/g;

/* initialize a, b, aa, and bb */

for (j=0; j<12; j++)
   for (k=0; k<12; k++)
      a[j][k] = 0.0;

for (j=0; j<6; j++)
   for (k=0; k<4; k++)
      b[j][k] = 0.0;

for (j=0; j<12; j++)
   for (k=0; k<12; k++)
      aa[j][k] = 0.0;

for (j=0; j<12; j++)
   for (k=0; k<4; k++)
      bb[j][k] = 0.0;


/*  Build the 12x12 a matrix  */


a[0][0]= tmp3*(xqds*ds*q0+xqdb/2*db*q0+xrdr*r0*dr)+
  tmp2*(xvdr*v0*dr+xwds*ds*w0+xwdb/2*w0*db+
  2*u0*(xdsds*ds*ds+xdbdb/2*db*db+xdrdr*dr*dr));
a[0][0]= a[0][0] + tmp3*xqdsn*q0*ds*eps+tmp2*(xwdsn*w0*ds+
  2*xdsdsn*u0*ds*ds)*eps+tmp2*u0*xprop+tmp3*
  xqdb/2*db*q0+tmp2*xwdb/2*db*w0+tmp2*u0*
  xdbdb/2*db*db;
a[0][1]= mass*r0+tmp3*(xvp*p0+xvr*r0)
  +tmp2*(2*xvv*v0+xvdr*u0*dr);
a[0][2]= -mass*q0+tmp3*xwq*q0+tmp2*(2*xww*w0+xwds*ds*u0+
  xwdb*db*u0+xwdsn*u0*ds*eps);
a[0][3]= -mass*yg*q0-mass*zg*r0+tmp4*(2*xpp*p0+xpr*r0)
  +tmp3*xvp*v0;
a[0][4]= -mass*w0+2*mass*xg*q0-mass*yg*p0+tmp4*2*xqq*q0+
  tmp3*(xwq*w0+xqds*ds*u0+xqdb/2*db*u0)+
  tmp3*xqdsn*u0*ds*eps+tmp3*xqdb/2*db*u0;
a[0][5]= mass*v0+2*mass*xg*r0-mass*zg*p0+tmp4*(2*xrr*r0+
  xpr*p0)+tmp3*(xvr*v0+xrdr*u0*dr);
a[0][10]=-(weight-boy)*cos_theta;


a[1][0]=-mass*r0+tmp3*(yp*p0+yr*r0)+tmp2*(yv*v0+
  2*ydr*u0*dr);
a[1][1]= tmp3*yvq*q0+tmp2*(yv*u0+yvw*w0);
a[1][2]= mass*p0+tmp3*(ywp*p0+ywr*r0)+tmp2*yvw*v0;
a[1][3]= mass*w0-mass*xg*q0+2*mass*yg*p0+tmp4*ypq*q0+
  tmp3*(yp*u0+ywp*w0);
a[1][4]=-mass*xg*p0-mass*zg*r0+tmp4*(ypq*p0+yqr*r0)+
  tmp3*yvq*v0;
a[1][5]=-mass*u0+2*mass*yg*r0-mass*zg*q0+tmp4*yqr*q0+
  tmp3*(yr*u0+ywr*w0);
a[1][9]= (weight-boy)*cos_theta*cos_phi;
a[1][10]=-(weight-boy)*sin_theta*sin_phi;


a[2][0]= mass*q0+tmp3*zq*q0+tmp2*(zw*w0+2*u0*zds*ds+
  2*u0*zdb/2*db+(zwn*w0+2*zdsn*u0*ds)*eps)+tmp3*
  zqn*q0*eps+tmp2*2*u0*zdb/2*db;
a[2][1]=-mass*p0+tmp3*(zvp*p0+zvr*r0)+tmp2*2*zvv*v0;
a[2][2]= tmp2*(zw*u0+zwn*u0*eps);
a[2][3]=-mass*v0-mass*xg*r0+2*mass*zg*p0+tmp4*(2*zpp*
  p0+zpr*r0)+tmp3*zvp*v0;
a[2][4]= mass*u0-mass*yg*r0+2*mass*zg*q0+tmp3*zq*u0+
  tmp3*zqn*u0*eps;
a[2][5]=-mass*xg*p0-mass*yg*q0+tmp4*(zpr*p0+2*zrr*r0)+
  tmp3*zvr*v0;
a[2][9]=-(weight-boy)*cos_theta*sin_phi;
a[2][10]=-(weight-boy)*sin_theta*cos_phi;


a[3][0]= mass*yg*q0+mass*zg*r0+tmp4*(kp*p0+
 kr*r0)+tmp3*(kv*v0+2*u0*(kdb/2*db-kdb/2*db))+
 tmp3*u0*kprop+tmp4*kpn*p0*eps;
a[3][1]=-mass*yg*p0+tmp4*kvq*q0+tmp2*(kv*u0+kvw*w0);
a[3][2]=-mass*zg*p0+tmp4*(kwp*p0+kwr*r0)+tmp3*kvw*v0;
```

```
a[3][3]=-ixy*r0+ixz*q0-mass*yg*v0-mass*zg*w0+
 tmp5*kpq*q0+tmp4*(kp*u0+kwp*w0);
a[3][4]=-iz*r0+iy*r0+2*iyz*q0+ixz*p0+mass*yg*u0+
 tmp5*(kpq*p0+kqr*r0)+tmp4*kvq*v0;
a[3][5]=-iz*q0+iy*q0-2*iyz*r0+mass*zg*u0+tmp5*kqr*q0+
 tmp4*(kr*u0+kwr*w0);
a[3][9]=-(yg*weight-yb*boy)*cos_theta*sin_phi-
 (zg*weight-zb*boy)*cos_theta*cos_phi;
a[3][10]=-(yg*weight-yb*boy)*sin_theta*cos_phi+
 (zg*weight-zb*boy)*sin_theta*sin_phi;


a[4][0]=-mass*xg*q0+tmp4*mq*q0+tmp3*mw*w0+tmp3*u0*
 (mds*ds+mdb/2*db)+tmp4*mqn*q0*eps+
 tmp3*(mwn*w0+2*mdsn*u0*ds)*eps+tmp3*u0*mdb/2*db;
a[4][1]= mass*xg*p0+mass*zg*r0+tmp4*(mvp*p0+mvr*r0)
  +tmp3*mvv*v0;
a[4][2]=-mass*zg*q0+tmp3*mw*u0+tmp3*mwn*u0*eps;
a[4][3]=-ix*r0+iz*r0-iyz*q0-2*ixz*p0+mass*xg*v0+
 tmp5*(2*mpp*p0+mpr*r0)+tmp4*mvp*v0;
a[4][4]= ixy*r0-iyz*p0-mass*xg*u0-mass*zg*w0+tmp4*mq*u0+
 tmp4*mqn*u0*eps;
a[4][5]=-ix*p0+iz*p0+ixy*q0+2*ixz*r0+mass*zg*v0+
 tmp5*(mpr*p0+2*mrr*r0)+tmp4*mvr*v0;
a[4][9]=(xg*weight-xb*boy)*cos_theta*sin_phi;
a[4][10]=(xg*weight-xb*boy)*sin_theta*cos_phi-
 (zg*weight-zb*boy)*cos_theta;


a[5][0]=-mass*xg*r0+tmp4*(np*p0+nr*r0)+tmp3*
 (nv*v0+2*ndr*u0*dr)+tmp3*u0*nprop;
a[5][1]=-mass*yg*r0+tmp4*nvq*q0+tmp3*(nv*u0+nvw*w0);
a[5][2]= mass*xg*p0+mass*yg*q0+tmp4*(nwp*p0+nwr*r0)+
  tmp3*nvw*v0;
a[5][3]=-iy*q0+ix*q0+2*ixy*p0+iyz*r0+mass*xg*w0+tmp5*npq*q0+
 tmp4*(np*u0+nwp*w0);
a[5][4]=-iy*p0+ix*p0-2*ixy*q0-ixz*r0+mass*yg*w0+
 tmp5*(npq*p0+nqr*r0)+tmp4*nvq*v0;
a[5][5]= iyz*p0-ixz*q0-mass*xg*u0-mass*yg*v0+
 tmp5*nqr*q0+tmp4*(nr*u0+nwr*w0);
a[5][9]=(xg*weight-xb*boy)*cos_theta*cos_phi;
a[5][10]=-(xg*weight-xb*boy)*sin_theta*sin_phi+
 (yg*weight-yb*boy)*cos_theta;


a[6][0]= cos_psi*cos_theta;
a[6][1]= cos_psi*sin_theta*sin_phi-sin_psi*cos_phi;
a[6][2]= cos_psi*sin_theta*cos_phi+sin_psi*sin_phi;
a[6][9]=v0*cos_psi*sin_theta*cos_phi+v0*sin_psi*
 sin_phi-w0*cos_psi*sin_theta*sin_phi+
 w0*sin_psi*cos_phi;
```

```
a[6][10]=-u0*cos_psi*sin_theta+v0*cos_psi*cos_theta*
 sin_phi+w0*cos_psi*cos_theta*cos_phi;
a[6][11]=-u0*sin_psi*cos_theta-v0*sin_psi*sin_theta*
 sin_phi-v0*cos_psi*cos_phi-w0*sin_psi*
 sin_theta*sin_phi+w0*cos_psi*sin_phi;


a[7][0]= sin_psi*cos_theta;
a[7][1]= sin_psi*sin_theta*sin_phi+cos_psi*cos_phi;
a[7][2]= sin_psi*sin_theta*cos_phi-cos_psi*sin_phi;
a[7][9]= v0*sin_psi*sin_theta*cos_phi-v0*cos_psi*
  sin_phi-w0*sin_psi*sin_theta*sin_phi-
  w0*cos_psi*cos_phi;
a[7][10]=-u0*sin_psi*sin_theta+v0*sin_psi*cos_theta*
  sin_phi+w0*sin_psi*cos_theta*cos_phi;
a[7][11]=u0*cos_psi*cos_theta+v0*cos_psi*sin_theta*
  sin_phi-v0*sin_psi*cos_phi+w0*cos_psi*
  sin_theta*cos_phi+w0*sin_psi*sin_phi;


a[8][0]=-sin_theta;
a[8][1]= cos_theta*sin_phi;
a[8][2]= cos_theta*cos_phi;
a[8][9]=v0*cos_theta*cos_phi-w0*cos_theta*sin_phi;
a[8][10]=-u0*cos_theta-v0*sin_theta*sin_phi-
  w0*sin_theta*cos_phi;


a[9][3]=1;
a[9][4]=sin_phi*tan_theta;
a[9][5]=cos_phi*tan_theta;
a[9][9]=q0*cos_phi*tan_theta-r0*sin_phi*tan_theta;
a[9][10]=q0*sin_phi/cos_theta*1/cos_theta+
  r0*cos_phi/cos_theta*1/cos_theta;


a[10][4]=cos_phi;
a[10][5]=-sin_phi;
a[10][9]=-q0*sin_phi-r0*cos_phi;


a[11][4]=sin_phi/cos_theta;
a[11][5]=cos_phi/cos_theta;
a[11][9]=q0*cos_phi/cos_theta-r0*sin_phi/cos_theta;
a[11][10]=q0*sin_phi/cos_theta*tan_theta+
   r0*cos_phi/cos_theta*tan_theta;


/*   Build the 12x4 b matrix  */
```

```
b[0][0]= tmp3*xrdr*u0*r0+tmp2*(xrdr*u0*v0+u0*u0*2*xdrdr*dr);
b[0][1]= u0*q0*xqds+u0*w0*xwds+u0*u0*2*xdsds*ds+
  tmp3*xqdsn*u0*q0*eps+
  tmp2*(xwdsn*u0*w0+2*xdsdsn*u0*u0*ds)*eps;
b[0][2]= u0*q0*xqdb+u0*w0*xwdb+2*u0*u0*xdbdb*db;
b[0][3]= tmp2*0.012*cdo*0.12*rpm;

b[1][0]= tmp2*ydr*u0*u0;

b[2][1]=u0*u0*zds*tmp2+tmp2*zdsn*u0*u0*eps;
b[2][2]=u0*u0*zdb*tmp2;

b[3][2]= 0;

b[4][1]= tmp3*(u0*u0*mds+mdsn*u0*u0*eps);
b[4][2]= tmp3*u0*u0*mdb;

b[5][0]= tmp3*ndr*u0*u0;


/* Multiply the appropriate parts of both by the inverted
 *  mass matrix
 *
 *  inv(mass matrix)*df/dx
 */

for (i=0; i<6; i++)
    for (j=0; j<6; j++)
        for (k=0; k<6; k++)
            aa[i][j] += xmminv[i][k]*a[k][j];
/*
 *  inv(mass matrix)*df/dz
 */

for (i=0; i<6; i++)
    for (j=6; j<12; j++)
        for (k=0; k<6; k++)
            aa[i][j] += xmminv[i][k]*a[k][j];

for (i=6; i<12; i++)
    for (j=0; j<12; j++)
        aa[i][j] = a[i][j];

/*
 *  inv(mass matrix)*df/du
 */

for (i=0; i<6; i++)
    for (j=0; j<4; j++)
        for (k=0; k<6; k++)
            bb[i][j] += xmminv[i][k]*b[k][j];
```

```
/*
 *  reorganize the matrices for use by Matlab
 *  which stores matrices columnwise vice rowwise
 */

for (j=0; j<12; j++)
    for (i=0; i<12; i++)
        A[i+12*j] = aa[i][j];
for (j=0; j<4; j++)
    for (i=0; i<12; i++)
        B[i+12*j] = bb[i][j];
}
```

```
/**********************************************
 *
 *  File   MODELPRM.H
 *
 *         This file contains all of the parameter
 *         coefficients used by the files MODEL.C.
 *         and MAKEAB.C.
 *         These coefficients were obtained from
 *         NCSC Technical Memorandum 231-78, June 78
 *
 *
 **********************************************/

/*
 *    longitudinal hydrodynamic coefficients
 */

const double
xpp    =  7.0e-3, xqq   =  -1.5e-2, xrr   =  4.0e-3,
xpr    =  7.5e-4, xudot=  -7.6e-3, xwq   =  -2.0e-1,
xvp    = -3.0e-3, xvr   =  2.0e-2,  xqds =  2.5e-2,
xqdb   = -2.6e-3, xrdr = -1.0e-3,  xvv   =  5.3e-2,
xww    =  1.7e-1, xvdr =   1.7e-3, xwds =  4.6e-2,
xwdb   =  1.0e-2, xdsds= -1.0e-2,  xdbdb= -8.0e-3,
xdrdr = -1.0e-2, xqdsn= 2.0e-3,   xwdsn =  3.5e-3,
xdsdsn= -1.6e-3;


/*
 *    lateral hydrodynamic coefficients
 */

const double
ypdot =  1.2e-4, yrdot =  1.2e-3, ypq =  4.0e-3,
yqr   = -6.5e-3, yvdot = -5.5e-2, yp  =  3.0e-3,
yr    =  3.0e-2, yvq   =  2.4e-2, ywp =  2.3e-1,
ywr   = -1.9e-2, yv    = -1.0e-1, yvw =  6.8e-2,
ydr   = '2.7e-2, cdy   =  3.5e-1;

/*
 *    normal hydrodynamic coefficients
 */

const double
zqdot = -6.8e-3, zpp  =  1.3e-4, zpr  =  6.7e-3,
zrr   = -7.4e-3, zwdot= -2.4e-1, zq   = -1.4e-1,
zvp   = -4.8e-2, zvr  =  4.5e-2, zw   = -3.0e-1,
zvv   = -6.8e-2, zds  = -7.3e-2, zdb  = -2.6e-2,
zqn   = -2.9e-3, zwn  = -5.1e-3, zdsn = -1.0e-2,
cdz   =    1.0;
```

```
/*
 *    roll hydrodynamic coefficients
 */

const double
kpdot = -1.0e-3, krdot = -3.4e-5, kpq = -6.9e-5,
kqr   =  1.7e-2, kvdot =  1.3e-4, kp  = -1.1e-2,
kr    = -8.4e-4, kvq   = -5.1e-3, kwp = -1.3e-4,
kwr   =  1.4e-2, kv    =  3.1e-3, kvw = -1.9e-1,
kpn   = -5.7e-4, kdb   =  0.0;


/*
 *    pitch hydrodynamic coefficients
 */

const double
mqdot = -1.7e-2, mpp   =  5.3e-5,  mpr  =  5.0e-3,
mrr   = -2.9e-3, mwdot = -6.8e-3,  mq   = -6.8e-2,
mvp   =  1.2e-3, mvr   =  1.7e-2,  mw   =  1.0e-1,
mvv   = -2.6e-2, mds   = -4.1e-2,  mdb  =  6.9e-3,
mqn   = -1.6e-3, mwn   = -2.9e-3,  mdsn = -5.2e-3;


/*
 *    yaw hydrodynamic coefficients
 */

const double
npdot = -3.4e-5, nrdot = -3.4e-3, npq = -2.1e-2,
nqr   =  2.7e-3, nvdot =  1.2e-3, np  = -8.4e-4,
nr    = -1.6e-2, nvq   = -1.0e-2, nwp = -1.7e-2,
nwr   =  7.4e-3, nv    = -7.4e-3, nvw = -2.7e-2,
ndr   = -1.3e-2;


/*
 *    mass characteristics of the flooded vehicle
 */

const double
weight = 12000.0, boy = 12000.0, vol =  200.0,
xg  =     0.0,     yg =     0.0, zg  =   0.20,
xb  =     0.0,     zb =     0.0, ix  = 1500.0,
iy  =    10000.0, iz = 10000.0, ixz =  -10.0,
iyz =   -10.0,     ixy =   -10.0, yb  =    0.0,
l =      17.4,     rho =     1.94,g   =   32.2,
nu = 8.47e-4,a0 =      2.0,      kprop =    0.0,
nprop =  0.0,      xltest = 0.1,
degrud = 0.0,      degstn = 0.0;
```

```
/*
 *    define length fractions for gauss quadrature terms
 */

const double
   g4[]  = { 0.069431844, 0.330009478, 0.669990521,
                 0.930568155 },
   gk4[] = { 0.1739274225687, 0.3260725774312,
                 0.3260725774312,  0.1739274225687 };


/*
 *    define the breadth bb and height hh terms for the
 *    integration
 */

const double
   br[] = {75.7/12.0, 75.7/12.0, 75.7/12.0, 55.08/12.0},
   hh[] = {16.38/12.0, 31.85/12.0, 31.85/12.0, 23.76/12.0};


/*
 *    assemble inverted mass matrix
 */

const double
xmminv[6][6] = {
     { 0.2431e-2, 0.2701e-8, 0.1899e-5, 0.1649e-7,
                 -0.5023e-5, 0.3243e-8 },
     { 0.2679e-8, 0.1537e-2, 0.5593e-8, 0.4276e-4,
                 -0.1479e-7,  0.1057e-4 },
     { 0.1899e-5, 0.5639e-8, 0.6293e-3, 0.3443e-7,
                 -0.1049e-4,  0.6770e-8 },
     { 0.1649e-7, 0.4321e-4, 0.3443e-7, 0.3294e-3,
                 -0.9106e-7, -0.1049e-5 },
  { -.5023e-5, -.1491e-7, -.1049e-4, -.9106e-7,
                  0.2773e-4, -0.1790e-7 },
  { 0.3243e-8, 0.1057e-4, 0.6769e-8, -.1052e-5,
                 -0.1790e-7,  0.6561e-4 }
   };
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%    File     GETMEAS2.M
%
%             simulates accelerometer and rate gyro
%             measurements using the nonlinear model as the
%             vehicle.  Additive white Gaussian noise included
%
%    Called by SIMUL9.M
%
%    Ver.3  Includes gyro biases, and depth rate meas.
%           no speed log measurement
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [anglerate,Z,Zdot,accel,hdg]=getmeas3(new,Zold,dt)

rand('normal');

% depth measurement
Z = new(9)+1e-3*rand;

% time rate of change of depth
Zdot = (Z - Zold)/dt;

%  simulated noise and bias on rate gyro
Vp=0.003*rand;
Vq=0.003*rand;
Vr=0.003*rand;
pBias = 0.10; qBias =0.200; rBias = -0.25;

%  rate gyro readings
anglerate(1,1)=new(4) + pBias + Vp;
anglerate(2,1)=new(5) + qBias + Vq;
anglerate(3,1)=new(6) + rBias + Vr;

%  accelerometer measurement of gravity (down is positive)
g=[0;0;32.2];

%  transform gravity vector to body coordinates to subtract
phi=new(10);                        % Euler roll angle
the=new(11);                        % Euler elevation/dive angle
psi=new(12);                        % Euler azimuth/heading


% calculate the tranformation matrix
R(1,1) = cos(psi)*cos(the);
R(2,1) = cos(psi)*sin(the)*sin(phi)-sin(psi)*cos(phi);
R(3,1) = cos(psi)*sin(the)*cos(phi)+sin(psi)*sin(phi);
R(1,2) = sin(psi)*cos(the);
R(2,2) = cos(psi)*cos(phi)+sin(psi)*sin(the)*sin(phi);
```

```
R(3,2) = -cos(psi)*sin(phi)+sin(psi)*sin(the)*cos(phi);
R(1,3) = -sin(the);
R(2,3) = cos(the)*sin(phi);
R(3,3) = cos(the)*cos(phi);


accel = -R*g + 0.00025*rand(3,1);

%  simulated accelerometer dead zone
if (abs(accel(1)) < 0.4)
        accel(1) = 0.0;
end;
if (abs(accel(2)) < 0.4)
        accel(2) = 0.0;
end;

hdg = psi;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File   MAKMEAS3.M
%
%          transforms the observer states into measurements
%          using nonlinear measurement equation.
%
%   Called by SIMUL9.M
%
%   Ver.3  corresponds with getmeas2
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [modmeas]=makmeas2(new,old,dt)

modmeas = zeros(9,1);

modmeas(1) = new(4) + new(13);          % rate gyro
modmeas(2) = new(5) + new(14);          % and bias estimate
modmeas(3) = new(6) + new(15);
modmeas(4) = new(9);                    % depth
modmeas(5) = (new(9)-old(9))/dt;        % depth dot


%  gravity (down is positive)
g=[0;0;32.2];

%  transform gravity vector to body coordinates to subtract
phi=new(10);                        % Euler roll angle
the=new(11);                        % Euler elevation/dive angle
psi=new(12);                        % Euler azimuth/heading


% calculate the tranformation matrix
R(1,1)=cos(psi)*cos(the);
R(2,1)=cos(psi)*sin(the)*sin(phi)-sin(psi)*cos(phi);
R(3,1)=cos(psi)*sin(the)*cos(phi)+sin(psi)*sin(phi);
R(1,2)=sin(psi)*cos(the);
R(2,2)=cos(psi)*cos(phi)+sin(psi)*sin(the)*sin(phi);
R(3,2)=-cos(psi)*sin(phi)+sin(psi)*sin(the)*cos(phi);
R(1,3)=-sin(the);
R(2,3)=cos(the)*sin(phi);
R(3,3)=cos(the)*cos(phi);

modmeas(6:8)=-R*g;                  % accelerometer meas

modmeas(9)= psi;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File   KALM.M
%
%          Time varying Kalman Filter subroutine
%          for predictor-corrector formulation of filter
%
%   Called by SIMUL9.M, AUVSIM.M, MAKEK.M
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [K,Pnew]= kalm(Phi,Del,C,Pold,W,V)

% update Pold
P=Phi*Pold*Phi'+Del*W*Del';

% calculate new gain matrix
K=P*C'*inv([C*P*C'+V]);

% complete update of P with new K matrix
Pnew=(eye(Phi)-K*C)*P;
```

APPENDIX B

Program Listing for Experimental NPS AUV II Study

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File    AUVSIM7.M
%
%           Structured as SIMUL.M but this uses
%           recorded data from the NPS AUV 2 vehicle.
%
%   Calls   AUV2.M, GETMEAS.M, MAKMEAS.M
%           MAKEK.M, GETK.M
%
%   Modified 18 Nov 91
%
%   Ver.2   added rate gyro bias terms to observer
%
%   Ver.3   removed speed log measurement
%           and added accelerometer dead zone
%
%   Ver.4   modified to use recorded vehicle data
%           and simulating accelerometer data from
%           vertical gyro data
%
%   Ver.5   using gyro data directly and added bias term for
%           pitch gyro
%
%   Ver.6   augment state to estimate
%           error in u using an input rpm bias as a state
%
%   Ver.7   using piecewise constant Extended Kalman filter
%           gain matrix. (and drop rpm bias)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%  dt = sample interval

%  load recorded data into memory. File obtained from
%  NPS AUV Group
load dat8262.d
data = dat8262;
clear dat8262;

%  Set initial state  based on data and estimated conditions
X         = data(1,2);
```

```
Y         = data(1,3);
Z         = data(1,4);
rpm       = 550;
u0        = 0.0;
w0        = 0.0;
pitchrate = data(1,9);
pitch     = data(1,6);
heading   = data(1,7);
phi0      = 0.0;
the0      = pitch;
psi0      = heading;
initial_state =
[u0;0;w0;0;pitchrate;0;X;Y;Z;0;pitch;heading];


%  Set sample interval and run time based on data set
dt = data(2,1)-data(1,1);
Tf = data(length(data),1)-data(1,1);

%  Set up disturbance and measurement noise covariance
%   matrices
%  (prepare to leave out X and Y terms from Kalman filter)
 Q = zeros(14);
 Q(1:10,1:10) = 0.05*eye(10);
 Q(1,1) = 1.0;
 Q(9,9) = .05;
 Q(11:14,11:14) = 0.0005*eye(4);      % gyro bias rand walk

% Measurement noise covariance matrix
% off diagonal terms for correlation between forward
% speed, pitch and depth rate.
 R=.01*eye(7);
 R(1,5) = 0.001;
 R(5,1) = R(1,5);
 R(6,6) = .005;
 R(5,5) = 0.05;
 R(5,6) = 0.001;
 R(6,5) = R(5,6);
 R(7,7) = 0.0001;


%  Initialize vectors for simulation
imax =Tf/dt + 1;
navig = zeros(16,imax);
navig(:,1) = [initial_state;0;0;0;0];

% Make block row matrix of steady state
% Extended Kalman filter gain matrices for table lookup
% the gain will be a function of u and theta

uRange = [1.5;2.0];
```

```
thtRange = [-0.05;-0.025;0.0;0.025;0.05];

allK = makeK(uRange,thtRange,Q,R,dt);

meas = zeros(7,1);

disp('Navigation Begins...')
t0=clock;

for (i=1:imax)

   rpm   = ((data(i,17)+data(i,18))/2);
   inputm =...
      [data(i,11);-data(i,11);data(i,12);-data(i,12);rpm];

   uhat = navig(1,i);
   thetahat = navig(11,i);

   % Look up proper steady state K matrix for the current
   % estimated state
   K = getK(allK,uhat,thetahat,uRange,thtRange);

   %  Format measurement from data file
   [rategyro,Z,Zdot,pitch,hdg]=getmeas5(data(i,:),Z,dt);
   meas = [rategyro;Z;Zdot;pitch;hdg];

   % Calculate next navigator prediction
   navig(1:12,i+1) = auv2(navig(1:12,i),inputm,dt);
   navig(13:16,i+1) = navig(13:16,i);

   % Generate predicted measurements
   modmeas(:,i+1) = makmeas3(navig(:,i+1),navig(:,i),dt);

   % Calculate correction using Kalman gain
   esterr(:,i)  = K*(meas-modmeas(:,i+1));

   % Correct navigator prediction
   navig(:,i+1) =...
         navig(:,i+1)+[esterr(1:6,i);0;0;esterr(7:14,i)];


end;

etime(clock,t0)
pause
axis([0 117 0 65]);

% Generate graphical output

!del expxy.met;
plot(data(:,2)+20,data(:,3)+15,navig(7,:)+20,navig(8,:)+15);
```

```
grid;xlabel('X - feet');ylabel('Y - feet');grid;pause;
meta expxy;

axis;
t = data(1,1):dt:data(length(data),1)+dt;

!del expu.met;
plot(data(:,1),data(:,15),t,navig(1,:));grid;
ylabel('feet/sec');xlabel('seconds');pause;
meta expu;

!del expv.met;
plot(t,navig(2,:));grid;
xlabel('seconds');ylabel('feet/sec');pause;
meta expv;

!del expw.met;
plot(t,navig(3,:));grid;
xlabel('seconds');ylabel('feet/sec');pause;
meta expw;

!del expp.met;
plot(data(:,1),data(:,8),t,navig(4,:));grid;
xlabel('seconds');ylabel('radians/sec');pause;
meta expp;

!del expq.met;
plot(data(:,1),data(:,9),t,navig(5,:));grid;
xlabel('seconds');ylabel('radians/sec');pause;
meta expq;

!del expr.met;
plot(data(:,1),data(:,10),t,navig(6,:));grid;
xlabel('seconds');ylabel('radians/sec');pause;
meta expr;

!del expz.met;
plot(data(:,1),data(:,4),t,navig(9,:));grid;
xlabel('seconds');ylabel('feet');pause;
meta expz;

!del expphi.met;
plot(data(:,1),data(:,5),t,navig(10,:));grid;
xlabel('seconds');ylabel('radians');pause;
meta expphi;

!del exptheta.met;
plot(data(:,1),data(:,6),t,navig(11,:));grid;
xlabel('seconds');ylabel('radians');pause;
meta exptheta;
```

```
!del exppsi.met;
plot(data(:,1),data(:,7),t,navig(12,:));grid;
xlabel('seconds');ylabel('radians');pause;
meta exppsi;

!del exppbias.met;
plot(t,navig(13,:));grid;
xlabel('seconds');ylabel('radians/sec');pause;
meta exppbias;

!del expqbias.met;
plot(t,navig(14,:));grid;
xlabel('seconds');ylabel('radians/sec');pause;
meta expqbias;

!del exprbias.met;
plot(t,navig(15,:));grid;
xlabel('seconds');ylabel('radians/sec');pause;
meta exprbias;

!del expthbia.met;
plot(t,navig(16,:));grid;
xlabel('seconds');ylabel('radians');pause;
meta expthbia;
```

```
/**********************************************
 *
 *  File   AUV2.C
 *
 *         C language source code for AUV2.MEX.  Must be
 *         compiled within MEX subdirectory of MATLAB.
 *         It is used like a MATLAB function:
 *
 *             function  state = auv2(oldstate,inputs,dt)
 *
 *         Translated to C for MEX file use from SIM3D,
 *         a FORTRAN based AUV2 model written
 *         by Prof. Fotis Papoulias and CDR David Warner,
 *         NPS, Monterey, CA.  It is based on modified
 *         equations of motion from NSRDC Report 2510
 *         June, 1967.
 *
 *  Called by AUVSIM7.M
 *
 **********************************************/


#include <math.h>
#include "auv2prm.h"
#include "tcmex.h"

double trapz( int n, double *a, const double *b );

int   auv2( double *state, double *oldstate, double *inputs,
double *dt );

/*
 *  This is the code recognized by MATLAB
 *  it calls function AUV2
 */

void  user_fcn( int nlhs, Matrix *plhs[], int nrhs, Matrix
*prhs[] )
{
   double  *oldstate, *inputs, *dt, *state;

   if (nrhs != 3)
      mex_error("Must be three input arguments.");
   if (nlhs != 1)
      mex_error("Must be one output argument.");
   if (ROWS_IN(0) != 12  ||  COLS_IN(0) != 1)
      mex_error("Previous state vector not correct size.");
   if (ROWS_IN(1) != 5  ||  COLS_IN(1) != 1)
      mex_error("Input vector not correct size.");
   if (ROWS_IN(2) != 1  ||  COLS_IN(2) != 1)
      mex_error("Time interval must be a scalar.");
   plhs[0] = create_matrix(12,1,REAL);
   state = OUT(0);
   oldstate = IN(0);
   inputs = IN(1);
   dt = IN(2);

   auv2(state,oldstate,inputs,dt);
}

/*
 * This code is the nonlinear model for NPS AUV II
 */

int  auv2( double *state, double *oldstate, double *inputs,
double *dt )
{
   int      j, k;
   int      flag = 0;
   double   u, v, w, p, q, r, phi, theta, psi;
   double   drs, drb, ds, db, rpm, delt;
   double   uv, ssas, ssab, drss, drbs;
   double   mass, heave, pitch, sway, yaw;
   double   ucf, cflow;
   double   tmp1, tmp2;
   double   vech1[15], vech2[15], vecv1[15], vecv2[15];
   double   fp[6], f[12];
   double   cos_theta, sin_theta, tan_theta;
   double   cos_phi, sin_phi, cos_psi, sin_psi;

   u     = oldstate[0];
   v     = oldstate[1];
   w     = oldstate[2];
   p     = oldstate[3];
   q     = oldstate[4];
   r     = oldstate[5];
   phi   = oldstate[9];
   theta = oldstate[10];
   psi   = oldstate[11];

   drs = inputs[0];
   drb = inputs[1];
   ds  = inputs[2];
   db  = inputs[3];
   rpm = inputs[4];

   delt = *dt;

   mass = weight/g;


/*.P*/
```

```
/*
 * calculate the drag force, integrate the drag over
 * the vehicle integrate using a 15 term trapezoidal
 * numerical integration
 */

   for (k=0; k<15; ++k) {
      tmp1 = (v+xx[k]*r);
      tmp2 = (w-xx[k]*q);
      ucf  = sqrt(fabs(tmp1*tmp1+tmp2*tmp2));
      if(ucf < 1.0e-6) {
         flag = 1;
         break;
         }
      cflow   = fabs((cdy*hh[k]*tmp1*tmp1 +
                     cdz*br[k]*tmp2*tmp2)/ucf);
      vech1[k] = cflow*tmp1;
      vech2[k] = vech1[k]*xx[k];
      vecv1[k] = cflow*tmp2;
      vecv2[k] = vecv1[k]*xx[k];

    }
    if (flag == 1) {
      heave = 0.0;
      pitch = 0.0;
      sway  = 0.0;
      yaw   = 0.0;
      }
    else {
      heave = trapz(15,vecv1,xx);
      pitch = trapz(15,vecv2,xx);
      sway  = trapz(15,vech1,xx);
      yaw   = trapz(15,vech2,xx);
      heave = -rho*heave/2.0;
      pitch = +rho*pitch/2.0;
      sway  = -rho*sway/2.0;
      yaw   = -rho*yaw/2.0;
      }

/*
 *    force equations
 */

/*
 *    common sub-expressions
 */

   ssas = 0.0;
   ssab = 0.0;
   uv = u;
   drss = drs - ssas;
   drbs = drb -ssab;
   cos_theta = cos(theta);
   sin_theta = sin(theta);
   tan_theta = tan(theta);
   cos_phi = cos(phi);
   sin_phi = sin(phi);
   cos_psi = cos(psi);
   sin_psi = sin(psi);

   if (phi == 0.0 )  {sin_phi =  0.0;}
   if (theta == 0.0) {sin_theta = 0.0; tan_theta = 0.0; }
   if (psi == 0.0 )  {sin_psi = 0.0; }

/*
 *    longitudinal force
 */

   fp[0] = mass*v*r - mass*w*q + mass*xg*q*q
      + mass*xg*r*r - mass*yg*p*q - mass*zg*p*r
      + xpp*p*p+xqq*q*q + xrr*r*r+xpr*p*r
      + xwq*w*q+xvp*v*p+xvr*v*r+u*q*(xqds*ds+xqdb*db)
      + u*r*(xrdrs*drss + xrdrb*drbs)
      + xvv*v*v+xww*w*w + u*v*(xvdrs*drss+xvdrb*drbs)
      + u*w*(xwds*ds+xwdb*db)
      + uv*uv*(xdsds*ds*ds+xdbdb*db*db
      + xdrdr*(drss*drss+drbs*drbs))
      - (weight-boy)*sin_theta + rpm*rpm*xprop - u*u*xres;

/*.P*/
/*
 *    lateral force
 */

   fp[1] = -mass*u*r - mass*xg*p*q + mass*yg*r*r
      - mass*zg*q*r + ypq*p*q + yqr*q*r + yp*u*p + yr*u*r
      + yvq*v*q + ywp*w*p
      + ywr*w*r + yv*u*v + yvw*v*w + ydrs*uv*uv*drss
      + ydrb*uv*uv*drbs +  (weight-boy)*cos_theta*sin_phi
      + mass*w*p + mass*yg*p*p + sway;

/*
 *    normal force
 */

   fp[2] = mass*u*q - mass*v*p - mass*xg*p*r - mass*yg*q*r
      + mass*zg*p*p + mass*zg*q*q + zpp*p*p + zpr*p*r
      + zrr*r*r + zq*u*q + zvp*v*p + zvr*v*r + zw*u*w
      + zvv*v*v + u*u*(zds*ds+zdb*db)
      + (weight-boy)*cos_theta*cos_phi
      + heave;
```

```
/*
 *    roll moment
 */

   fp[3] = -iz*q*r + iy*q*r - ixy*p*r + iyz*q*q
      - iyz*r*r + ixz*p*q + mass*yg*u*q - mass*yg*v*p
      - mass*zg*w*p + kpq*p*q + kqr*q*r + kp*u*p + kr*u*r
      + kvq*v*q + kwp*w*p + kwr*w*r + kv*u*v + kvw*v*w
      + (yg*weight - yb*boy)*cos_theta*cos_phi
      - (zg*weight - zb*boy)*cos_theta*sin_phi +
mass*zg*u*r;

/*
 *    pitch moment
 */

   fp[4] = -ix*p*r + iz*p*r + ixy*q*r - iyz*p*q - ixz*p*p
      + ixz*r*r - mass*xg*u*q + mass*xg*v*p + mass*zg*v*r
      - mass*zg*w*q + mpp*p*p +mpr*p*r +mrr*r*r + mq*u*q
      + mvp*v*p + mvr*v*r + mw*u*w + mvv*v*v
      + u*u*(mds*ds+mdb*db)
      - (xg*weight-xb*boy)*cos_theta*cos_phi
      - (zg*weight-zb*boy)*sin_theta + pitch;

/*.P*/
/*
 *    yaw moment
 */

   fp[5] = -iy*p*q + ix*p*q + ixy*p*p - ixy*q*q + iyz*p*r
      - ixz*q*r - mass*xg*u*r + mass*xg*w*p - mass*yg*v*r
      + mass*yg*w*q + npq*p*q + nqr*q*r + np*u*p + nr*u*r
      + nvq*v*q + nwp*w*p + nwr*w*r + nv*u*v + nvw*v*w
      + ndrs*uv*uv*drss + ndrb*uv*uv*drbs + (xg*weight
      - xb*boy)*cos_theta*sin_phi
      + (yg*weight-yb*boy)*sin_theta + yaw;

/*
 *    now compute the f(0-5) functions
 */

   for (j=0; j<6; ++j)
      for (f[j]=0.0,k=0; k<6; ++k)
         f[j] += xmminv[j][k]*fp[k];

/*
 *    the last six equations come from the kinematic
 *     relations
 */

/*
```

```
 *    inertial position rates f(6-8)
 */

   f[6] =  u*cos_psi*cos_theta + v*(cos_psi*sin_theta*
      sin_phi - sin_psi*cos_phi) + w*(cos_psi*sin_theta*
      cos_phi + sin_psi*sin_phi);

   f[7] =  u*sin_psi*cos_theta + v*(sin_psi*sin_theta*
      sin_phi + cos_psi*cos_phi) + w*(sin_psi*sin_theta*
      cos_phi - cos_psi*sin_phi);

   f[8] = -u*sin_theta + v*cos_theta*sin_phi +
w*cos_theta*cos_phi;

/*
 *    euler angle rates f(9-11)
 */

   f[9] = p + q*sin_phi*tan_theta + r*cos_phi*tan_theta;
   f[10] = q*cos_phi - r*sin_phi;
   f[11] = q*sin_phi/cos_theta + r*cos_phi/cos_theta;

/*
 *    first order integration
 */

   for (j=0; j<12; j++)
      state[j] = oldstate[j] + delt * f[j];

   return  0;
}

/*
 * This function performs a trapezoidal integration
 */

double trapz( int n, double *a, const double *b )
  {
   int n1, k;
   double y;
   n1 = n-1;
   for (y = 0.0, k=0; k<n1; k++) {
      y += (a[k] + a[k+1])*(b[k+1]-b[k])/2;
   }
   return y;
}
```

```
/*********************************************
 *
 *  File   AUV2AB.C
 *
 *         C language scurce code for AUV2AB.MEX.  Must be
 *         compiled within MEX subdirectory of MATLAB.
 *         It is used like a MATLAB function:
 *
 *             function  [A,B] = auv2AB(state,inputs)
 *
 *         Based on Taylor series expansion of the equations
 *         of motion in AUV2.C
 *
 *  Called by AUVSIM7.M
 *
 *********************************************/


#include <math.h>
#include "tcmex.h"
#include "auv2prm.h"

void  makeab( double *A, double *B, double *state, double
*inputs);

/*
 *  This section of code is recognized by MATLAB
 *  It calls function MAKEAB
 */
void user_fcn( int nlhs, Matrix *plhs[], int nrhs, Matrix
*prhs[] )
{
 double *state, *inputs, *a, *b;

 if (nrhs != 2)
  mex_error("Must be two input arguments");
 if (nlhs != 2)
  mex_error("Must be two output arguments");
 if (ROWS_IN(0) != 12 || COLS_IN(0) != 1)
  mex_error("Initial state vector must have 12 states");
 if (ROWS_IN(1) != 5  || COLS_IN(1) != 1)
  mex_error("Input vector must have 5 elements");

 plhs[0] = create_matrix(12,12,REAL);
 plhs[1] = create_matrix(12,5,REAL);
 a = OUT(0);
 b = OUT(1);
 state = IN(0);
 inputs = IN(1);

 makeab(a,b,state,inputs);
```

```
}

/*
 * This code makes the linearized model.
 */

void makeab( double *A, double *B, double *state, double
*inputs )
{
 int    i, j, k;
 double a[12][12], b[6][5];
 double aa[12][12],bb[12][5];
 double u0, v0, w0, p0, q0, r0, phi0, theta0, psi0;
 double mass;
 double drs, drb, ds, db, rpm;
 double cos_theta, sin_theta, tan_theta;
 double cos_phi, sin_phi, cos_psi, sin_psi;


/* assign some common variable names */


u0 = state[0];
v0 = state[1];
w0 = state[2];
p0 = state[3];
q0 = state[4];
r0 = state[5];
phi0 = state[9];
theta0 = state[10];
psi0 = state[11];

drs = inputs[0];
drb = inputs[1];
ds  = inputs[2];
db  = inputs[3];
rpm = inputs[4];

cos_theta = cos(theta0);
sin_theta = sin(theta0);
tan_theta = tan(theta0);
cos_phi   = cos(phi0);
sin_phi   = sin(phi0);
cos_psi   = cos(psi0);
sin_psi   = sin(psi0);

if (theta0 == 0.0) { sin_theta = 0.0; tan_theta = 0;}
if (phi0 == 0.0) sin_phi = 0.0;
if (psi0 == 0.0) sin_psi = 0.0;

mass = weight/g;
```

```
/* initialize a, b, aa, and bb */

for (j=0; j<12; j++)
   for (k=0; k<12; k++)
      a[j][k] = 0.0;

for (j=0; j<6; j++)
   for (k=0; k<5; k++)
      b[j][k] = 0.0;

for (j=0; j<12; j++)
   for (k=0; k<12; k++)
      aa[j][k] = 0.0;

for (j=0; j<12; j++)
   for (k=0; k<5; k++)
      bb[j][k] = 0.0;


/*  Build the 12x12 a matrix  */

/* d(long. force)/dx */

a[0][0]= q0*(xqds*ds+xqdb*db)+r0*(xrdrs*drs+xrdrb*drb)+
         v0*(xvdrs*drs+xvdrb*drb)+w0*(xwds*ds+xwdb*db)+
         2*u0*(xdsds*ds*ds+xdbdb*db*db+
         xdrdr*(drs*drs+drb*drb))-2*xres*u0;
a[0][1]= mass*r0+xvp*p0+xvr*r0+2*xvv*v0+
         u0*(xvdrs*drs+xvdrb*drb);
a[0][2]= -mass*q0+xwq*q0+2*xww*w0+u0*(xwds*ds+xwdb*db);
a[0][3]= -mass*yg*q0-mass*zg*r0+2*xpp*p0+xpr*r0+xvp*v0;
a[0][4]= -mass*w0+2*mass*xg*q0-mass*yg*p0+2*xqq*q0+
          xwq*w0+u0*(xqds*ds+xqdb*db);
a[0][5]= mass*v0+2*mass*xg*r0-mass*zg*p0+2*xrr*r0+xpr*p0+
         xvr*v0+u0*(xrdrs*drs+xrdrb*drb);
a[0][10]=-(weight-boy)*cos_theta;


/*  d(lat. force)/dx  */

a[1][0]=-mass*r0+yp*p0+yr*r0+yv*v0+2*u0*(ydrs*drs+ydrb*drb);
a[1][1]= yvq*q0+yv*u0+yvw*w0;
a[1][2]= mass*p0+ywp*p0+ywr*r0+yvw*v0;
a[1][3]=
mass*w0-mass*xg*q0+2*mass*yg*p0+ypq*q0+yp*u0+ywp*w0;
a[1][4]=-mass*xg*p0-mass*zg*r0+ypq*p0+yqr*r0+yvq*v0;
a[1][5]=
-mass*u0+2*mass*yg*r0-mass*zg*q0+yqr*q0+yr*u0+ywr*w0;
a[1][9]= (weight-boy)*cos_theta*cos_phi;
a[1][10]=-(weight-boy)*sin_theta*sin_phi;
```

```
/*   d(norm. force)/dx   */

a[2][0]= mass*q0+zq*q0+zw*w0+2*u0*(zds*ds+zdb*db);
a[2][1]=-mass*p0+zvp*p0+zvr*r0+2*zvv*v0;
a[2][2]= zw*u0;
a[2][3]=-mass*v0-mass*xg*r0+2*mass*zg*p0+2*zpp*p0+
         zpr*r0+zvp*v0;
a[2][4]= mass*u0-mass*yg*r0+2*mass*zg*q0+zq*u0;
a[2][5]=-mass*xg*p0-mass*yg*q0+zpr*p0+2*zrr*r0+zvr*v0;
a[2][9]=-(weight-boy)*cos_theta*sin_phi;
a[2][10]=-(weight-boy)*sin_theta*cos_phi;


/*  d(roll moment)/dx   */

a[3][0]= mass*yg*q0+mass*zg*r0+kp*p0+kr*r0+kv*v0;
a[3][1]=-mass*yg*p0+kvq*q0+kv*u0+kvw*w0;
a[3][2]=-mass*zg*p0+kwp*p0+kwr*r0+kvw*v0;
a[3][3]=-ixy*r0+ixz*q0-mass*yg*v0-mass*zg*w0+
         kpq*q0+kp*u0+kwp*w0;
a[3][4]=-iz*r0+iy*r0+2*iyz*q0+ixz*p0+mass*yg*u0+
         kpq*p0+kqr*r0+kvq*v0;
a[3][5]=-iz*q0+iy*q0-ixy*p0-2*iyz*r0+mass*zg*u0+
         kqr*q0+kr*u0+kwr*w0;
a[3][9]=-(yg*weight-yb*boy)*cos_theta*sin_phi-
         (zg*weight-zb*boy)*cos_theta*cos_phi;
a[3][10]=-(yg*weight-yb*boy)*sin_theta*cos_phi+
          (zg*weight-zb*boy)*sin_theta*sin_phi;


/*  d(pitch moment)/dx    */

a[4][0]=-mass*xg*q0+mq*q0+mw*w0+2*u0*(mds*ds+mdb*db);
a[4][1]= mass*xg*p0+mass*zg*r0+mvp*p0+mvr*r0+2*mvv*v0;
a[4][2]=-mass*zg*q0+mw*u0;
a[4][3]=-ix*r0+iz*r0-iyz*q0-2*ixz*p0+mass*xg*v0+
         2*mpp*p0+mpr*r0+mvp*v0;
a[4][4]= ixy*r0-iyz*p0-mass*xg*u0-mass*zg*w0+mq*u0;
a[4][5]=-ix*p0+iz*p0+ixy*q0+2*ixz*r0+mass*zg*v0+
         mpr*p0+2*mrr*r0+mvr*v0;
a[4][9]= (xg*weight-xb*boy)*cos_theta*sin_phi;
a[4][10]=(xg*weight-xb*boy)*sin_theta*cos_phi-
         (zg*weight-zb*boy)*cos_theta;


/*   d(yaw moment)/dx   */

a[5][0]=-mass*xg*r0+np*p0+nr*r0+nv*v0+
         2*u0*(ndrs*drs+ndrb*drb);
a[5][1]=-mass*yg*r0+nvq*q0+nv*u0+nvw*w0;
```

```
a[5][2]= mass*xg*p0+mass*yg*q0+nwp*p0+nwr*r0+nvw*v0;
a[5][3]=-iy*q0+ix*q0+2*ixy*p0+iyz*r0+mass*xg*w0+
        npq*q0+np*u0+nwp*w0;
a[5][4]=-iy*p0+ix*p0-2*ixy*q0-ixz*r0+mass*yg*w0+
        npq*p0+nqr*r0*nvq*v0;
a[5][5]= iyz*p0-ixz*q0-mass*xg*u0-mass*yg*v0+
        nqr*q0+nr*u0+nwr*w0;
a[5][9]= (xg*weight-xb*boy)*cos_theta*cos_phi;
a[5][10]=-(xg*weight-xb*boy)*sin_theta*sin_phi+
         (yg*weight-yb*boy)*cos_theta;


/*  d(inertial position X)/dx    */

a[6][0]= cos_psi*cos_theta;
a[6][1]= cos_psi*sin_theta*sin_phi-sin_psi*cos_phi;
a[6][2]= cos_psi*sin_theta*cos_phi+sin_psi*sin_phi;
a[6][9]= v0*(cos_psi*sin_theta*cos_phi+sin_psi*sin_phi)+
        w0*(-cos_psi*sin_theta*sin_phi+sin_psi*cos_phi);
a[6][10]=-u0*cos_psi*sin_theta+v0*cos_psi*cos_theta*
         sin_phi+w0*cos_psi*cos_theta*cos_phi;
a[6][11]=-u0*sin_psi*cos_theta-v0*(sin_psi*sin_theta*sin_phi
          +cos_psi*cos_phi)+
         w0*(-sin_psi*sin_theta*cos_phi+cos_psi*sin_phi);


/* d(inertial postion Y)/dx  */

a[7][0]= sin_psi*cos_theta;
a[7][1]= sin_psi*sin_theta*sin_phi+cos_psi*cos_phi;
a[7][2]= sin_psi*sin_theta*cos_phi-cos_psi*sin_phi;
a[7][9]= v0*(sin_psi*sin_theta*cos_phi-cos_psi*sin_phi)-
        w0*(sin_psi*sin_theta*sin_phi+cos_psi*cos_phi);
a[7][10]=-u0*sin_psi*sin_theta+v0*sin_psi*cos_theta*sin_phi+
         w0*sin_psi*cos_theta*cos_phi;
a[7][11]= u0*cos_psi*cos_theta+
         v0*(cos_psi*sin_theta*sin_phi-
         sin_psi*cos_phi)+
         w0*(cos_psi*sin_theta*cos_phi+sin_psi*sin_phi);


/*  d(depth)/dx   */

a[8][0]=-sin_theta;
a[8][1]= cos_theta*sin_phi;
a[8][2]= cos_theta*cos_phi;
a[8][9]= v0*cos_theta*cos_phi-w0*cos_theta*sin_phi;
a[8][10]=-u0*cos_theta-v0*sin_theta*sin_phi-
         w0*sin_theta*cos_phi;
```

```
/* d(roll angle)/dx    */

a[9][3]=1;
a[9][4]=sin_phi*tan_theta;
a[9][5]=cos_phi*tan_theta;
a[9][9]=q0*cos_phi*tan_theta-r0*sin_phi*tan_theta;
a[9][10]=(q0*sin_phi+r0*cos_phi)/cos_theta*1/cos_theta;


/* d(pitch angle)/dx   */

a[10][4]=cos_phi;
a[10][5]=-sin_phi;
a[10][9]=-q0*sin_phi-r0*cos_phi;


/*   d(yaw angle)/dx   */

a[11][4]=sin_phi/cos_theta;
a[11][5]=cos_phi/cos_theta;
a[11][9]=q0*cos_phi/cos_theta-r0*sin_phi/cos_theta;
a[11][10]=(q0*sin_phi+r0*cos_phi)/cos_theta*tan_theta;


/*   Build the 12x5 b matrix  */

/* d(long force)/d(inputs)  */

b[0][0]= xrdrs*u0*r0+xrdrs*u0*v0+2*u0*u0*xdrdr*drs;
b[0][1]= xrdrb*u0*r0+xrdrb*u0*v0+2*u0*u0*xdrdr*drb;
b[0][2]= u0*q0*xqds+u0*w0*xwds+2*u0*u0*xdsds*ds;
b[0][3]= u0*q0*xqdb+u0*w0*xwdb+2*u0*u0*xdbdb*db;
b[0][4]= 2*xprop*rpm;


/*  d(lat force)/d(inputs)   */

b[1][0]= ydrs*u0*u0;
b[1][1]= ydrb*u0*u0;


/*  d(normal force)/d(inputs)   */

b[2][2]= u0*u0*zds;
b[2][3]= u0*u0*zdb;


/*  d(roll moment)/d(inputs)   */

b[3][3]= 0;
```

```
/*   d(pitch moment)/d(inputs)   */

b[4][2]= u0*u0*mds;
b[4][3]= u0*u0*mdb;


/*  d(yaw moment)/d(inputs)   */

b[5][0]= ndrs*u0*u0;
b[5][1]= ndrb*u0*u0;


/* Multiply the appropriate parts of both by the inverted
 *    mass matrix
 *
 *  inv(mass matrix)*df/dx
 */

for (i=0; i<6; i++)
    for (j=0; j<6; j++)
        for (k=0; k<6; k++)
            aa[i][j] += xmminv[i][k]*a[k][j];
/*
 *  inv(mass matrix)*df/dz
 */

for (i=0; i<6; i++)
    for (j=6; j<12; j++)
        for (k=0; k<6; k++)
            aa[i][j] += xmminv[i][k]*a[k][j];

for (i=6; i<12; i++)
    for (j=0; j<12; j++)
        aa[i][j] = a[i][j];

/*
 *  inv(mass matrix)*df/du
 */

for (i=0; i<6; i++)
    for (j=0; j<5; j++)
        for (k=0; k<6; k++)
            bb[i][j] += xmminv[i][k]*b[k][j];

/*
 *  reorganize the matrices for use by Matlab
 *  which stores matrices columnwise vice rowwise
 */

for (j=0; j<12; j++)
    for (i=0; i<12; i++)
```

```
            A[i+12*j] = aa[i][j];

    for (j=0; j<5; j++)
        for (i=0; i<12; i++)
            B[i+12*j] = bb[i][j];

    }
```

```
/************************************************
 *
 *  File   AUV2PRM.H
 *
 *         This file contains all of the parameter
 *         coefficients used by the file AUV2.C
 *         and AUV2AB.C.
 *         The format is identical to that in MODELPRM.H
 *         which contains parameters for the Swimmer
 *         Delivery Vehicle
 *
 *         These parameters were determined by
 *         Prof. Fotis Papoulias and CDR David Warner, NPS
 *         Monterey, CA.
 *
 ************************************************/

/*
 * define some constants for use in expressions
 */

#define trm2 1.94/2*87.625/12.0*87.625/12.0
#define trm3 1.94/2*87.625/12.0*87.625/12.0*87.625/12.0
#define trm4
    1.94/2*87.625/12.0*87.625/12.0*87.625/12.0*87.625/12.0
#define trm5
    1.94/2*87.625/12.0*87.625/12.0*87.625/12.0*87.625/
     12.0*87.625/12.0
#define lngth 87.625/12.0
#define urpm 2.5/550.0

/*
 *   mass characteristics of the vehicle
 */

const double
   weight = 435.0,      boy =  435.0,  vol = 200.0,
   xg     = 0.125/12.0, yg  =    0.0,  zg  =   0.05,
   xb     = 0.125/12.0, zb  =    0.0,  ix  =   2.7,
   iy     =  42.0,      iz  =   45.0,  ixz = 0.0,
   iyz =    0.0,        ixy =    0.0,   yb = 0.0,
   l =      87.625/12.0,rho =   1.94,   g =  32.2,
   cd0 = 0.015,
   cdy =    0.5, cdz =   0.6,    rpm0= 550.0,    u0 =  2.5,
   xrs =    -0.377*lngth,xrb =    0.283*lngth;


/*
 *   longitudinal hydrodynamic coefficients
 */
```

```
const double
   xpp =   0.0,  xqq =   0.0,   xrr =   -0.01735*trm4,
   xpr =   0.0,  xudot = -2.82e-3*trm3,  xwq =   0.0,
   xvp =   0.0,  xvr =    0.0,           xqds =   0.0,
   xqdb =  0.0,  xrdrs =  0.0,           xrdrb = 0.0,
   xvv =  -4.019e-2*trm2,  xww =    0.0, xvdrs =  0.0,
   xvdrb = 0.0,  xwds =  0.0,            xwdb =   0.0,
   xdsds = -2.345e-3*0.417*trm2,
   xdbdb = -2.345e-3*0.417*trm2,
   xdrdr = -2.345e-3*0.417*trm2,
   xres =  0.015*trm2,     xprop = 0.015*trm2*(urpm)*(urpm);


/*
 *    lateral hydrodynamic coefficients
 */

const double
   ypdot = 0.0, yrdot = -1.78e-3*trm4,  ypq =    0.0,
   yqr =   0.0, yvdot = -3.43e-2*trm3,  yp  =    0.0,
   yr  =   1.187e-2*trm3, yvq =    0.0, ywp =    0.0,
   ywr =   0.0,           yv  =   -3.896e-2*trm2,
   yvw =   0.0,           ydrs =  2.345e-2*trm2,
   ydrb =  2.345e-2*trm2;


/*
 *    normal hydrodynamic coefficients
 */

const double
   zqdot = -2.53e-3*trm4, zpp = 0.0,  zpr =    0.0,
   zrr =  0.0, zwdot = -9.34e-2*trm3, zq =  -7.013e-2*trm3,
   zvp =    0.0,  zvr =   0.0,        zw = -1.5678e-1*trm2,
   zvv =    0.0, zds = -2.345e-2*trm2,  zdb =-2.345e-2*trm2;


/*
 *    roll hydrodynamic coefficients
 */

const double
   kpdot = -2.4e-4*trm5, krdot = 0.0,  kpq =  0.0,
   kqr =    0.0,         kvdot = 0.0,  kp =  -5.4e-3*trm4,
   kr =   0.0,           kvq =   0.0,  kwp =   0.0,
   kwr =    0.0,         kv =    0.0,  kvw =   0.0;


/*
 *    pitch hydrodynamic coefficients
 */
```

```
const double
   mqdot = -6.25e-3*trm5, mpp =    0.0,     mpr = 0.0,
   mrr =   0.0,           mwdot = -2.53e-3*trm4,
   mq = -3.565e-2*trm4,   mvp =    0.0,     mvr =    0.0,
   mw =  5.122e-2*trm3,   mvv =   0.0,
   mds =   -0.377*lngth*2.345e-2*trm2,
   mdb = 0.283*lngth*2.345e-2*trm2;


/*
 *    yaw hydrodynamic coefficients
 */

const double
   npdot = 0.0, nrdot = -4.7e-4*trm5,  npq =   0.0,
   nqr   = 0.0, nvdot = -1.78e-3*trm4, np =    0.0,
   nr   = -1.022e-2*trm4, nvq =   0.0, nwp =   0.0,
   nwr  =  0.0, nv =    -7.69e-3*trm3, nvw =   0.0,
   ndrs = -0.377*lngth*2.345e-2*trm2,
   ndrb = 0.283*lngth*2.345e-2*trm2;


/*
 *    define length fractions terms for gauss quadrature
 */

const double
   x[]  = {-43.9/12.0, -39.2/12.0, -35.2/12.0, -31.2/12.0,
          -27.2/12.0, -10.0/12.0,  0.0, 10.0/12.0,
           26.8/12.0,  32.0/12.0,  37.8/12.0,  40.8/12.0,
           42.3/12.0,  43.3/12.0,  43.7/12.0};


/*
 *    define the breadth br and height hh terms
 */

const double
   hh[] = { 0.0/12.0, 2.7/12.0, 5.2/12.0, 7.6/12.0,
           10.1/12.0, 10.1/12.0, 10.1/12.0, 10.1/12.0,
           10.1/12.0, 9.6/12.0, 7.6/12.0, 5.6/12.0,
           4.2/12.0, 2.3/12.0, 0.0 },
   br[] = { 16.5/12.0, 16.5/12.0, 16.5/12.0, 16.5/12.0,
           16.5/12.0, 16.5/12.0, 16.5/12.0, 16.5/12.0,
           16.5/12.0, 15.5/12.0, 12.45/12.0, 9.5/12.0,
            7.0/12.0, 4.0/12.0, 0.0 };


/*
 *    assemble inverted mass matrix
 *    this matrix was calculated previously using the above
 *     coefficients and the equations of motion
 */

const double
   xmminv[6][6] = {
      { 6.8626e-2, 0.0, 3.8922e-5, 0.0, -2.7774e-4, 0.0 },
      { 0.0, 3.8558e-2, 0.0, 3.4574e-3, 0.0, -3.5748e-3 },
      { 3.8922e-5, 0.0, 2.0616e-2, 0.0, -8.3980e-4,  0.0 },
      { 0.0, 3.4574e-3, 0.0, 1.3306e-1, 0.0, -3.2055e-4 },
      {-2.7774e-4, 0.0,-8.3980e-4, 0.0,  5.9927e-3,  0.0 },
      { 0.0,-3.5748e-3, 0.0,-3.2054e-4,  0.0,  1.8692e-2 }
   };
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File   MAKEK.M
%
%
%          Function which makes the piecewise constant
%          Extended Kalman filter gain matrices for various
%          values of forward speed, u, and pitch angles,
%          theta.  Assumes that other states are zero to
%          keep the number of different K matrices to a
%          minimum.  Returns a block row matrix of steady
%          state K matrices
%
%   Calls  DLQE.M, AUV2AB.MEX,
%
%   Called by AUVSIM7.M
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function allK = makeK(uRange,thtRange,Q,R,dt)

allK = [];

u = uRange;
theta = thtRange;

% calculate rpm range corresponding to the u range using
% known rpm to thrust relationship and drag to speed
% relationships:
%
%       thrust = drag  (in steady state)
%       thrust = const1*rpm^2
%       drag   = const2*u^2
%
% and
%       550 rpm gives 2.5 feet/sec
%

rpm = u*550/2.5;

% shell for measurement matrix C:

C=[0 0 0 1 0 0 0 0 0 1 0 0 0; %   ┐ =|p|
   0 0 0 0 1 0 0 0 0 0 1 0 0; %   ] =|q|+ biases,
   0 0 0 0 0 1 0 0 0 0 0 1 0; %   ┘  |r|
   0 0 0 0 0 0 1 0 0 0 0 0 0; %      depth
   0 0 0 0 0 0 0 0 0 0 0 0 0; %      depth dot
   0 0 0 0 0 0 0 0 1 0 0 0 0 1; %    Theta (pitch + bias
   0 0 0 0 0 0 0 0 0 1 0 0 0 0];%    Psi (heading gyro)

%  loop through both u and theta to calculate K,
%  the Extended Kalman filter gain matrices

for uIndx = 1:length(u)

   input =[0;0;0;0;rpm(uIndx)];

   for thIndx = 1:length(theta)

      % Setup the full 12 state linearized model

[a,b]=auv2ab([u(uIndx);zeros(9,1);theta(thIndx);0],input);

      % delete from 'a' and 'b' elements having to do
      % with X and Y to remove poles at origin to
      % help stabilize
      % the Extended Kalman filter
      a = [a(1:6,1:6) , a(1:6,9:12);
           a(9:12,1:6), a(9:12,9:12)];
      b = [b(1:6,:);
           b(9:12,:)];

      % add gyro bias terms as constant states
      % with random walk
      a = [a zeros(10,4);
           zeros(4,14)];
      b = [b;zeros(4,5)];

      %  disturbance input matrix
      b2=sign(Q);

      % convert to discrete time
      [Phi,Gam2]=c2d(a,b2,dt);

      % Recalculate the depth dot row of the C  matrix
      % around u and theta only

      C(5,1) = -sin(theta(thIndx));
      C(5,3) =  cos(theta(thIndx));
      C(5,9) = -u(uIndx)*cos(theta(thIndx));

      % calculate the Extended Kalman filter gain matrix for
      % u & theta values, & append it to form a matrix made
      % of gain matrices
      % DLQE solves the discrete time Riccati equation
      % to find the steady state Kalman filter gain for a
      % given linear system
      allK = [allK,real(dlqe(Phi,Gam2,C,Q,R))];
    end;
end;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File   GETK.M
%
%          Function which compares estimated speed and
%          estimated pitch to the nominal speed and pitch
%          vectors to determine which of the steady state
%          Kalman filter gain matrices to use in the
%          piecewise constant approximation for the Extended
%          Kalman filter
%
%   Called by AUVSIM7.M
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function K = getK(allK,u,theta,uRange,thRange)

%  determine index of the nominal speed and pitch which
%  is closest to the actual speed and pitch

[error,uIndx] = min(abs(u-uRange));
[error,thIndx] = min(abs(theta-thRange));

%  allK was made by MAKEK.M and therefore it is
%  a block row matrix of K matrices calculated over the
%  the range of u values in uRange and the range of theta
%  values in thRange.  Theta is the most rapidly changing
%  index so allK has the following format:
%
%  allK =...
%   [K(uRange(1),thRange(1)),K(uRange(1),thRange(2)),...
%      K(uRange(last),thRange(last-1)),...
%            K(uRange(last),thRange(last))]
%
%  so allK has:
%  (number of states in Extended Kalman filter) rows &
%  (length(uRange)*length(thRange)*(number of measurements))
%   columns


[m,n] = size(allK);
Kwidth = n/length(uRange)/length(thRange);

% so the proper K to use is

Indx = 1 + Kwidth*((uIndx-1)*length(thRange) + (thIndx-1));

K = allK(:,Indx:(Indx+Kwidth-1));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File   GETMEAS5.M
%
%          formats actual recorded measurement data for
%          navigator
%
%   Ver.4  using recorded data vice simulated data
%
%   Ver.5  use pitch gyro directly rather than through
 %         simulated accelerometers
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [anglerate,Z,Zdot,pitch,hdg]=getmeas5(data,Zold,dt)

% depth, Z, measurement
Z = data(4);

% time rate of change of depth
Zdot = (Z - Zold)/dt;

%  rate gyro readings
anglerate(1,1)=data(8);
anglerate(2,1)=data(9);
anglerate(3,1)=data(10);

pitch = data(6);
hdg = data(7);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   File   MAKMEAS3.M
%
%       subroutine to transform the observer states
%       into measurements
%
%   Called by AUVSIM7.M
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [modmeas]=makmeas3(new,old,dt)

modmeas = zeros(7,1);

modmeas(1) = new(4) + new(13);          % rate gyro
modmeas(2) = new(5) + new(14);          % with bias added)
modmeas(3) = new(6) + new(15);
modmeas(4) = new(9);                    % depth
modmeas(5) = (new(9)-old(9))/dt;        % depth dot


modmeas(6) = new(11) + new(16);
modmeas(7) = new(12);
```

LIST OF REFERENCES

1. Bane, G. L., Ferguson, J., "The Evolutionary Development of the Military Autonomous Vehicle," *5th International Symposium on Unmanned Untethered Submersible Technology*, Vol. 1, University of New Hampshire, June 22-24, 1987.

2. Blidberg, D. R., "Time-Ordered Architecture for Knowledge-Based Guidance of an Unmanned Untethered Submersible," paper presented at the Oceans Engineering Conference, Washington, D.C., 10-12 September 1984.

3. Boncal, Richard. J., A *Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1987.

4. O'Donnell, C. F., and others, *Inertial Navigation Analysis and Design*, McGraw-Hill Book Company, 1964.

5. Putnam, Rex G., A *Conceptual Design of an Inertial Navigation System for an Autonomous Submersible Testbed Vehicle*, Masters's Thesis, Naval Postgraduate School, Monterey, California, September 1987.

6. Friedland, Bernard, *Control System Design*, McGraw-Hill Book Company, 1986.

7. Burl, Jeff B., "Derivation of the Kalman Filter Equations," notes for EC3310 (Linear Optimal Estimation and Control), Naval Postgraduate School, 1990 (unpublished).

8. Gelb, A., and others, *Applied Optimal Estimation*, The Massachusetts Institute of Technology Press, 1974.

9. Thaler, George J., *Automatic Control Systems*, West Publishing Company, 1989.

10. Burl, Jeff B., "Linearization of Nonlinear Systems," notes for EC3310 (Linear Optimal Estimation and Control), Naval Postgraduate School, 1990 (unpublished).

11. NSRDC Report 2510, *Standard Equations of Motion for Submarine Simulation*, by M. Gertler and G. R. Hagen, June 1967.

12. NCSC Technical Memorandum 231-78, *SDV Simulator Hydrodynamic Coefficients*, by N. S. Smith, J. W. Crane, and D. C. Sumney, June 1978.

13. Snyder, Wesley E., *Industrial Robots: Computer Interfacing and Control*, Prentice-Hall, Incorporated, 1985.

14. McGhee, R. B., "Transformation of Body Angular Rates to Euler Angle Rates," notes for CS4314 (Computers for Artificial Intelligence), Naval Postgraduate School, 1986 (unpublished).

15. Warner, David C., *Design, Simulation, and Experimental Verification of a Computer Model and Enhanced Position Estimator for the NPS AUV II*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1991.

16. The MathWorks Corporation, *MATLAB for MS-DOS Personal Computers User's Guide*, 1990.