NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ANALYSIS AND TUNING OF A LOW COST INERTIAL NAVIGATION SYSTEM IN THE ARIES AUV

by

Steven R. Vonheeder

December 2006

Thesis Advisor: Anthony J. Healey

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Analysis and Tuning of a Low Cost Inertial Navigation System in the ARIES AUV			**5. FUNDING NUMBERS**
6. AUTHOR: Vonheeder, Steven R.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research, Code 32			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES 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 U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT

Autonomous underwater vehicle navigation is a complex problem of state estimation. Accurate navigation is made difficult due to a lack of reference navigation aids or use of the Global Positioning System (GPS) that could establish the vehicles position. Accurate navigation is critical due to the level of autonomy and range of missions and environments into which an underwater vehicle may be deployed. Navigational accuracy depends not only on the initialization and drift errors of the low cost Inertial Motion Unit (IMU) gyros and the speed over ground sensor, but also on the performance of the sensor fusion filter used.

This thesis will present the method by which an Extended Kalman Filter (EKF) was tuned after installation of an IMU in the ARIES Autonomous Underwater Vehicle. The goal of installing the IMU, analyzing the navigational results and tuning the EKF was to achieve navigational accuracy in the horizontal plane with a position error of less than one percent of distance traveled when compared with GPS. The research consisted of IMU installation and software modifications within the vehicle to fully realize the design goal. Data collection and analysis was conducted through field experiments and computer simulation. A significant result of this work was development of a pseudo-adaptive algorithm to vary the measurement noise values in selected channels to force a desired response in the filter and improve accuracy and precision in the state estimates.

14. SUBJECT TERMS Navigation, Underwater vehicle, AUV, ARIES, Kalman Filter, Extended Kalman Filter, IMU, Inertial Navigation System			**15. NUMBER OF PAGES:** 140
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ANALYSIS AND TUNING OF A LOW COST INERTIAL NAVIGATION SYSTEM IN THE ARIES AUV

Steven R. Vonheeder
Lieutenant Commander, United States Navy
B.S. Nuclear Engineering, Oregon State University, 1994
M.S. Environmental Management, University of Maryland University College, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2006

Author: Steven R. Vonheeder

Approved by: Anthony J. Healey
Thesis Advisor

Anthony J. Healey
Chairman, Department of Mechanical and
Astronautical Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Autonomous underwater vehicle navigation is a complex problem of state estimation. Accurate navigation is made difficult due to a lack of reference navigation aids or use of the Global Positioning System (GPS) that could establish the vehicles position. Accurate navigation is critical due to the level of autonomy and range of missions and environments into which an underwater vehicle may be deployed. Navigational accuracy depends not only on the initialization and drift errors of the low cost Inertial Motion Unit (IMU) gyros and the speed over ground sensor, but also on the performance of the sensor fusion filter used.

This thesis will present the method by which an Extended Kalman Filter (EKF) was tuned after installation of an IMU in the ARIES Autonomous Underwater Vehicle. The goal of installing the IMU, analyzing the navigational results and tuning the EKF was to achieve navigational accuracy in the horizontal plane with a position error of less than one percent of distance traveled when compared with GPS. The research consisted of IMU installation and software modifications within the vehicle to fully realize the design goal. Data collection and analysis was conducted through field experiments and computer simulation. A significant result of this work was development of a pseudo-adaptive algorithm to vary the measurement noise values in selected channels to force a desired response in the filter and improve accuracy and precision in the state estimates.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ARIES	Acoustic Radio Interactive Exploratory Server
AUV	Autonomous Underwater Vehicle
EKF	Extended Kalman Filter
GPS	Global Positioning System
IMU	Inertial Motion Unit
INS	Inertial Navigation System
RDI	RD Instruments
UKF	Unscented Kalman Filter
UUV	Unmanned Underwater Vehicle

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I thank God first and foremost for the blessings and opportunities He has presented in my life. I would like to thank my family. To my wife, Dawn, I want you to know how much I appreciate the support and encouragement you have provided. To my sons, Travis and Nathan, I appreciate your patience with me when I always seemed to be too busy to spend time with you.

I would like to thank my thesis advisor, Professor Anthony Healey, for his expert insight, direction, and assistance during this work. His knowledge and guidance were instrumental in the conduct of this work.

I would like to thank Sean Kragelund for his technical help throughout this endeavor and for always managing to set me straight when I would get lost in the fields of data and code. His programming skills and patience for all of my seemingly crazy ideas were invaluable.

Finally, to all the members of the NPS AUV Research Group, particularly Dr. Hag Seong Park of South Korea, I express my gratitude for listening, providing input, and offering thoughts and opinions over the course of this work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. BACKGROUND

Autonomous underwater vehicle (AUV) navigation is a complex problem of state estimation. Accurate navigation is further made difficult due to a lack of reference navigation aids or use of the Global Positioning System (GPS) that could be used to establish the vehicles position. Potential mission requirements necessitate accurate navigation due to the level of autonomy and the potential range of missions and environments into which an underwater vehicle may be deployed. Navigational accuracy depends not only on the initialization and drift errors of the low cost Inertial Motion Unit (IMU) gyros and the speed over ground sensor, but also on the performance of the sensor fusion filter used.

An IMU helps the problem of state estimation by providing accurate sensory inertial inputs that can be used along with a model of the vehicle dynamics. The outputs of the IMU take the form of linear accelerations and angular rates that can then be input into a system model that will allow for estimation of the vehicles state, in particular the vehicle position that is necessary for conduct of various missions.

This thesis will present the method by which an Extended Kalman Filter (EKF) was tuned after installation of an IMU in the ARIES AUV. The goal of installing the IMU, analyzing the navigational results and tuning the EKF was to achieve navigational accuracy in the horizontal plane with a position error of less than one percent of distance traveled when compared with GPS. The research consisted of IMU installation and software modifications within the vehicle to fully realize the design goal. Data collection and analysis was conducted through field experiments and computer simulation. A significant result of this work was development of a pseudo-adaptive algorithm to vary the measurement noise values in selected channels to force a desired response in the filter and improve accuracy and precision in the state estimates.

B. MOTIVATION

Unmanned Underwater Vehicles (UUVs) are being actively pursued in the United States Navy as a means to enhance war fighting in the underwater realm. UUVs refer to both remotely controlled vehicles and AUVs. These vehicles can be used in a wide range of mission functions from Intelligence, Surveillance, and Reconnaissance to Mine Warfare to Salvage and Recovery operations as detailed in the Navy's Master UUV Plan (2004). They are a force multiplier and allow the Navy the accomplish missions that may be too dangerous or impractical for current practices. "The long-term UUV vision is to have the capability to: (1) deploy or retrieve devices, (2) gather, transmit, or act on all types of information, and (3) engage bottom, volume, surface, air or land targets (UUV Master Plan, 2004)." Figure 1 shows UUVs in use during recent military operations. These missions require accurate navigation to perform their tasks.

Figure 1. Tactical Application of AUV (From: UUV Master Plan, 2004)

The motivation for this thesis research was to obtain navigational estimates of position in the horizontal plane that were within one percent error of the distance traveled. This design goal would need to be addressed through both software and hardware configuration changes to the navigation architecture of the ARIES vehicle. The most significant physical change was in the hardware configuration which would utilize a

relatively low cost IMU currently used in production of military missile technology. Once the gains in positional accuracy from the hardware were realized the navigation filter would be tuned in order to achieve the design goal.

C. THE ARIES VEHICLE

"The ARIES is used for development of computer architecture, software, sensors and navigational hardware for small to medium sized autonomous systems (Healey, 2006)." It is approximately three meters in length and is fitted with various sensors and electronics in order to carry out the development and research noted above. Figure 2 shows the ARIES being loaded onto the research support vessel Cyprus Sea in Monterey Harbor.

Figure 2. ARIES Operations in Monterey Bay (From: Healey, 2006)

The vehicles nominal operating speed is 1.2 to 1.5 meters per second developed from twin thrusters mounted in the rear. The vehicle operates off of a bank of batteries that provide a nominal bus voltage of 60 volts for the vehicle propulsion and hotel loads.

Vertical maneuvering control for the vehicle is provided by two forward and two rear plane surfaces. Steering control is provided by a top mounted twin rudder configuration, one forward and one aft.

Communications with the vessel occur through several antennas on top of the vehicle which include 802.11 type wireless digital communication as well as standard radio free wave communication. A GPS receiver is mounted on top of the aft rudder to allow for receipt of GPS signals when on the surface. Navigation in the ARIES is performed without the use of any radio beacons and relies solely on GPS, inertial navigation system, and the Doppler speed sensor.

There are three main compartments in the vehicle that house the equipment necessary to run the vehicle. The forward compartment consists of a PC-104 computer that operates the sonar imaging obtained from ARIES forward looking blazed array sonar. The forward compartment also houses the servos for the forward control surfaces. The mid compartment is the largest and houses the vehicle relays for ancillary sensors, the battery banks, and the two main computers used for operating the vehicle. There is one computer dedicated to executive level process management and mission execution and one computer dedicated to tactical execution of commands used for vehicle motion. Finally, the aft compartment houses the motors for the thrusters and servos for the aft control surfaces, the wireless router for communications and on the forward bulkhead of the aft compartment, the IMU.

D. THESIS STRUCTURE

The research conducted was an attempt to achieve navigational errors within one percent of the distance traveled. This was an iterative approach conducted over an approximately one year period that involved data analysis from field experiments coupled with computer simulation. The analyses of the errors and modifications made to the vehicle from each significant evolution are presented.

Chapter II will present the theory of the inertial navigation model used, covering specifically the operation of inertial motion units and the theory of the EKF as used in the

ARIES vehicle. Chapter III will provide the details of the IMU installation into the vehicle. Chapter IV will detail the software changes that were implemented over the course of this work. Chapter V will present the field experiments and the geometries utilized, and the results obtained, presenting in detail the navigational errors. Finally, Chapter VI provides thesis conclusions and recommendations for future work. The supporting code utilized in this work will be retained in the appendices to this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. THEORY OF INERTIAL NAVIGATION

A. INTRODUCTION

Inertial navigation systems (INS) have become increasingly widespread in their use over the last few decades. This is due in large part to technological gains made in computing power and in increasing sensitivity of inertial sensor units made from quality manufacturing. The backbone of an inertial navigation system is the IMU, which has gained increasing use due to manufacturing techniques allowing the production of small and accurate light weight systems as well as larger extremely precise units.

Coupled with the several order of magnitude increase in computing power that has been observed in the last few decades is the development of sophisticated algorithms that can process and manipulate the large data streams from an inertial sensor. These algorithms allow for increased accuracy in state estimation by using more information and increasingly smaller time steps in computing dynamic information. The latter is a result of extremely fast processing speeds now capable in computer systems. These algorithms allow for inertial navigation of vehicles relying on inputs from inertial sensors for navigation such as underwater vehicles and submarines without continuous external reference navigation inputs.

B. INERTIAL MOTION UNIT

Inertial motion units (also called inertial measurement units) have become increasingly popular as measurement devices of vehicle motion for use in inertial navigation systems. These units come in a wide variety of forms from simple strap-down systems to extremely accurate complex gimbaled or stabilized platform systems. These instruments work by sensing the vehicle's inertial linear accelerations and angular rotation rates which can then be sent to a computer for processing with a filter that fuses data from different sensors. The intricate details of inertial motion units are well published and the following provides a basic understanding of the strap-down style system that was used for this research.

A strap-down style IMU is ideal for a wide range of applications; particularly in vehicles where space is limited due to its relatively small size and low weight. These units are robust and their use of solid state electronics make them more reliable than mechanical gimbaled systems (Yakimenko, 2006). For the details of mathematics and mechanization of a strap-down IMU the reader is referred to Siouris (1993). The strap-down system uses accelerometers to measure linear accelerations and a set of gyroscopes to measure the angular rates of the body. From these accelerations velocity and position information may be obtained through integration. The gyroscopes measure the rate at which the vehicles attitude changes. From this data the vehicles yaw, pitch, and roll may be obtained. A simple illustration of a strap-down IMU is shown in Figure 3.

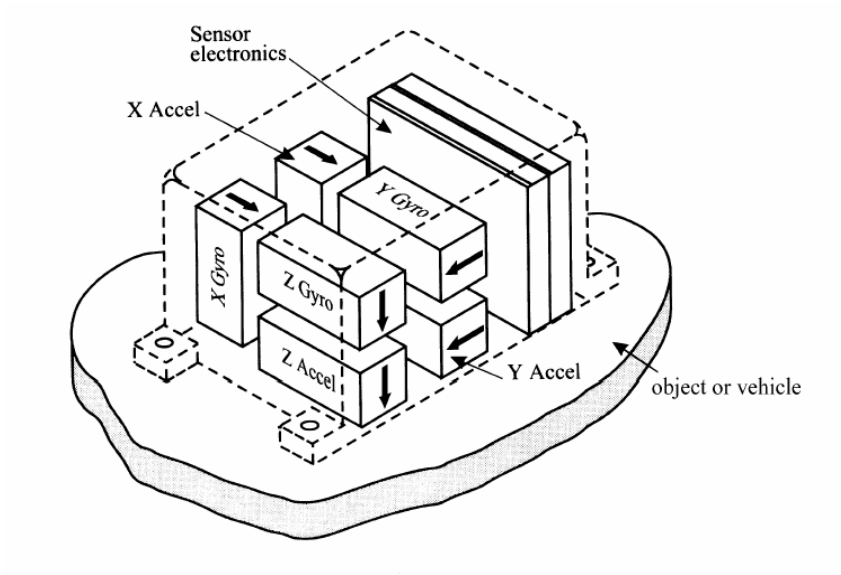

Figure 3. Basic Strap-down IMU (From: Roth, 1999)

C. EXTENDED KALMAN FILTER

The EKF was developed to address the problem of utilizing a standard Kalman filter, a linear estimator, for problems involving non-linear system dynamics and measurements. Motion of vehicles with non-linear dynamics can be made locally linear through the use of the Jacobian of the dynamics matrix from the vehicle equations of

motion. By taking a first-order series expansion of the non-linear equations, the system is linearized about a point in time (Bar-Shalom, 2001). The algorithm can then employ the estimation techniques of a Kalman filter to produce an estimate of the vehicle state at an instant in time.

1. Assumptions of the EKF

The EKF utilizes some well known assumptions that are an extension from the original Kalman Filter. These assumptions are relevant when comparing filters of various types, e.g. the Unscented Kalman Filter, in order to understand the strengths and weaknesses of different estimation methods. The EKF assumes that the process noise (**q(t)**) and the measurement noise (**v(t)**) are white, additive and zero mean. Additionally, the process and measurement noises are uncorrelated, i.e. independent of each other. Finally, the initial state and its covariance are independent of either the process or measurement noise. In summary the process and measurement noise are given as (Bar-Shalom, 2001):

$$\begin{aligned} E(\mathbf{q(t)}) &= 0 \\ E(\mathbf{qq}') &= \mathbf{Q} \end{aligned} \tag{1}$$

$$\begin{aligned} E(\mathbf{v(t)}) &= 0 \\ E(\mathbf{vv}') &= \mathbf{R} \end{aligned} \tag{2}$$

2. System Model

The literature on EKFs is well developed and extensive; therefore, the following discussion will focus on the specifics of the EKF as used in the ARIES AUV. For more information on EKFs the reader is referred to Bar-Shalom (2001).

Navigational state estimates in the ARIES vehicle are made by utilizing an EKF to predict and correct these states every 0.125 seconds. The state of the vehicle for the horizontal plane is contained in an eight state vector as follows (Healey, 1995):

$$\mathbf{x}(t) = [X, Y, \psi, r, u_g, v_g, b_r, b_\psi]' \tag{3}$$

Where:

X = North-South position in Local Navigation Plane

Y = East-West position in Local Navigation Plane

Ψ = Heading

r = Yaw (Heading) rate

u_g = Forward speed over ground

v_g = Lateral speed over ground

b_r = Yaw rate bias

b_ψ = Heading bias

The system dynamics and measurement model are given by (Healey, 1995):

$$\begin{aligned} \dot{\mathbf{x}}(\mathbf{t}) &= \mathbf{f(x(t))} + \mathbf{q(t)} \\ \mathbf{y}(\mathbf{t}) &= \mathbf{h(x(t))} + \mathbf{v(t)} \end{aligned} \tag{4}$$

For ARIES the vector valued function **h(x(t))** is constant and can be represented as:

$$\mathbf{y(t)} = \mathbf{Cx(t)} \tag{5}$$

The EKF may be formulated in either continuous or discrete time; the discrete time is necessary for computer application (Healey, 1995). The vehicle states are linked through the following vehicle equations of motion and are contained in the dynamics matrix.

$$
\begin{aligned}
\dot{X} &= u_g \cos(\psi) - v_g \sin(\psi) \\
\dot{Y} &= u_g \sin(\psi) + v_g \cos(\psi) \\
\dot{\psi} &= r \\
\dot{r} &= 0 \\
\dot{u}_g &= 0 \\
\dot{v}_g &= 0 \\
\dot{b}_r &= 0 \\
\dot{b}_\psi &= 0
\end{aligned}
\tag{6}
$$

Linearizing these equations yields the transition matrix, $\mathbf{\Phi}$, used to propagate the state and covariance matrix between time steps in the discrete model.

The measurements are associated with the state vector as follows:

$$
\begin{aligned}
& y_1 = u_g; \\
& y_2 = v_g; \\
& y_3 = \psi + b_\psi; \\
& y_4 = r + b_r; \\
& y_5 = X; \\
& y_6 = Y; \\
& \text{or} \\
& \boldsymbol{y} = \boldsymbol{h(x(t))} = \boldsymbol{C}\boldsymbol{x}(t)
\end{aligned}
\tag{7}
$$

The discrete-time filter can then be represented as:

$$
\begin{aligned}
\hat{\mathbf{x}}_{\mathbf{i+1/i}} &= \mathbf{\Phi}\hat{\mathbf{x}}_{\mathbf{i|i}} \\
\mathbf{P}_{\mathbf{i+1/i}} &= \mathbf{\Phi}\mathbf{P}_{\mathbf{i/i}}\mathbf{\Phi}' + \mathbf{Q} \\
\mathbf{L}_{\mathbf{i+1/i}} &= \mathbf{P}_{\mathbf{i+1/i}}\mathbf{C}'(\mathbf{C}\mathbf{P}_{\mathbf{i+1|i}}\mathbf{C}' + \mathbf{R})^{-1} \\
\hat{\mathbf{x}}_{\mathbf{i+1|i+1}} &= \hat{\mathbf{x}}_{\mathbf{i+1/i}} + \mathbf{L}_{\mathbf{i+1}}(\mathbf{y}_{\mathbf{i+1}} - \mathbf{C}\hat{\mathbf{x}}_{\mathbf{i+1/i}}) \\
\mathbf{P}_{\mathbf{i+1|i+1}} &= (\mathbf{I} - \mathbf{L}_{\mathbf{i+1}}\mathbf{C})\mathbf{P}_{\mathbf{i+1|i}}
\end{aligned}
\tag{8}
$$

Where:

$\hat{\mathbf{x}}$ = Estimate of state

P = Covariance of states

Q = Process noise matrix

R = Measurement noise matrix

L = Kalman Gain

The resulting algorithm provides discrete state updates at each time step and predicts the successive time step and then corrects the estimate based on the measurements received. When properly tuned and with good quality sensors the EKF provides very good estimates of the vehicle state. Tuning the filter is done through the choice of the values for the process and measurement noise matrices, based on known information about the sensors and/or experience through application of the filter.

D. NAVIGATION ERROR DEFINED

The navigational accuracy must be defined to provide a standard quantitative measure by which to evaluate the changes made to the ARIES inertial navigation system. For this work, only motion in the horizontal plane was analyzed. The absolute error ($\tilde{E}$) in meters is defined as norm of the error vector taken as the difference between the initial GPS position at a time step (*i*) and the navigation filter estimate at the (*i-1*) time step.

$$\tilde{E} = \sqrt{\left(X_{GPS(i)} - X_{NavFilter(i-1)}\right)^2 + \left(Y_{GPS(i)} - Y_{NavFilter(i-1)}\right)^2} \tag{9}$$

The method in which the GPS information is used when obtained in the navigation process required the use of the *(i-1)* position. This is because at time step *(i)*, if GPS position information is available this information is inserted into the EKF for the X and Y measurements and the estimates are updated, providing a new position estimate based on this information. It was necessary to see where the filter thought it was located prior to obtaining updated position information compared to the actual vehicle position determined by GPS. The assumption was made that GPS information was absolute truth for position in order to compare filter performance. The processing of GPS information had to be modified slightly in order to obtain increased accuracy by rejecting fixes for which the number of satellites visible to the receiver were below a set threshold. This modification is discussed further in Chapter IV. The error induced from using the *(i-1)*

position is at most vehicle speed times the time interval of 0.125 seconds, or 0.15 meters, which can be considered negligible in this analysis.

E. THEORETICAL ERROR ANALYSIS

The ideal position error can be estimated from the manufacturers published gyro drift rate for the IMU and some basic assumptions for vehicle dynamics. The Honeywell HG1700 has a published drift rate of one degree per hour. Given the dynamics as:

$$\dot{\tilde{Y}} = U\tilde{\Psi} \tag{10}$$

$$\tilde{\Psi} = kt + \tilde{\Psi}_0 \tag{11}$$

Where:

$\dot{\tilde{Y}} =$ Cross Track Error Rate

$\tilde{\Psi} =$ Heading Error

$\tilde{\Psi}_0 =$ Heading Initialization Error $= 1$ deg

$k =$ Gyro Drift Rate $= 1$ deg/ hr

$U =$ Forward Vehicle Speed $= 1.2 m/s$

$$\tilde{Y} = \int_0^T \dot{\tilde{Y}}\, dt = \int_0^T U\left(kt + \tilde{\Psi}_0\right) dt$$

$$\tilde{Y} = U\left(k\frac{T^2}{2} + \tilde{\Psi}_0 T\right) \tag{12}$$

The theoretical cross track error for a thirty minute run is as follows:

$$\tilde{Y} = 1.2\frac{\text{m}}{\text{sec}}\left(1\frac{\text{deg}}{\text{hr}}\cdot\frac{1\,\text{rad}}{57.3\,\text{deg}}\cdot\frac{0.5^2\ \text{hr}^2}{2} + 1\text{deg}\cdot\frac{1\,\text{rad}}{57.3\,\text{deg}}\cdot 0.5\ \text{hr}\right)\cdot\left(\frac{3600\,\text{sec}}{1\,\text{hr}}\right) = 47.1\ \text{m}$$

$$Y \equiv \text{Distance Traveled} = 1.2\frac{\text{m}}{\text{sec}}\cdot 1800\text{sec} = 2160\ \text{m}$$

The resulting error as a percentage of distance traveled is 2.18%. Based on this a design goal of one percent of the distance traveled was set for this research. Figure 4

illustrates the theoretical cross track error that develops over time based on the above formulations. Figure 5 illustrates the percentage error growth over time. Both figures are parameterized by the initialization error from using the original compass to align the IMU. The effects of the initialization error can be seen on the resulting cross track error and error per unit distance traveled.

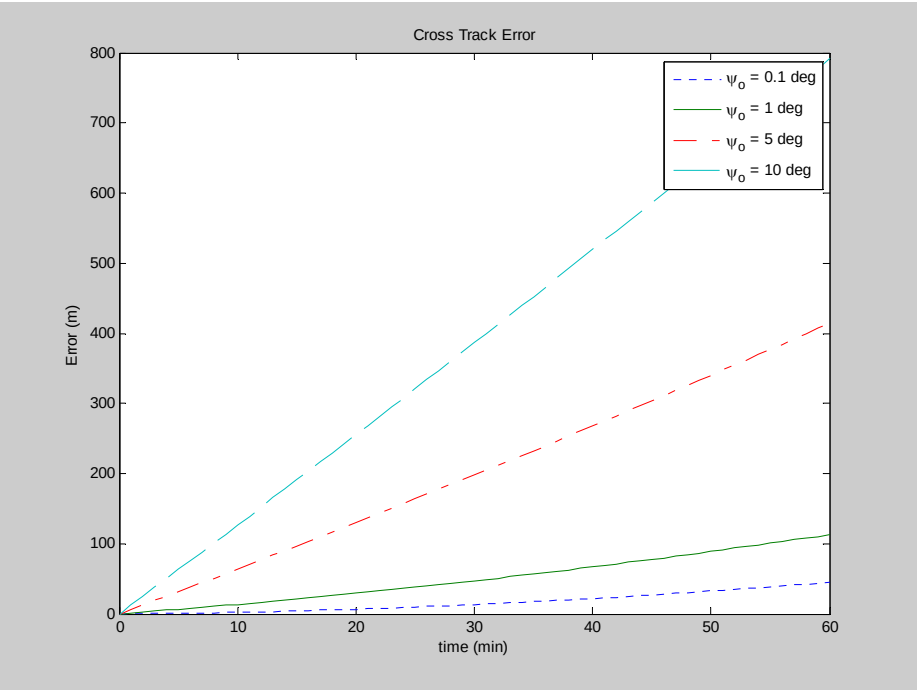

Figure 4. Theoretical Cross Track Error

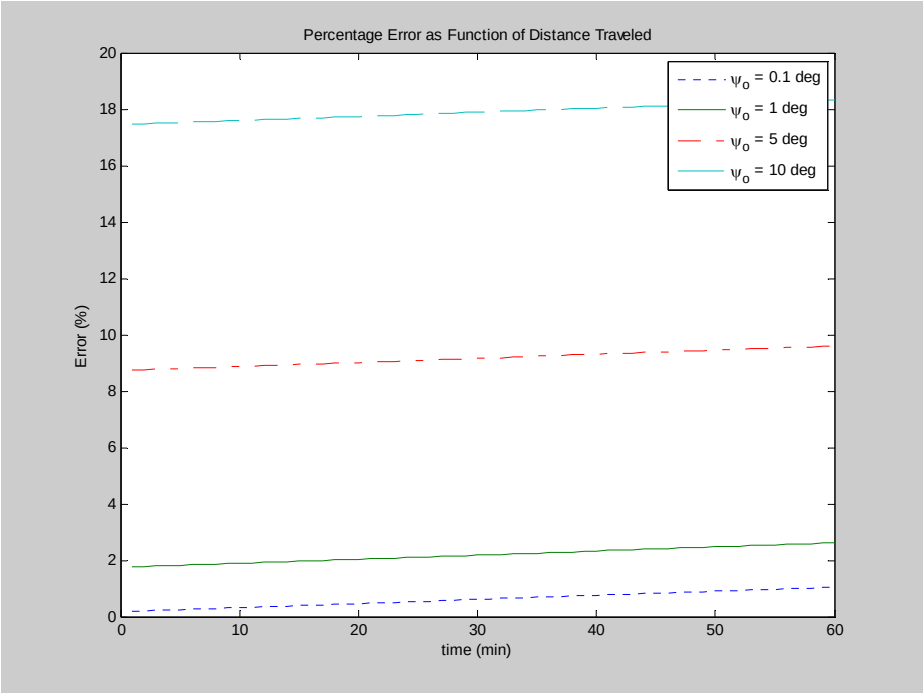

Figure 5. Theoretical Error Growth

As can be seen from these two figures the amount of error in position estimates that will develop is strongly dependent on the initialization error of the IMU and to some degree the drift rate of the gyro. Therefore, it is critical that the navigation filter learn the heading bias quickly in order to compensate for this initialization error. The rate of learning must be balanced against the overall performance of the filter.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HARDWARE IMPLEMENTATION

A. INTRODUCTION

The ARIES vehicle originally used a compass to provide heading and a Systron Donner Motion Pak IMU provided gyro rate inputs to the navigation filter. The compass was a Honeywell HMR3000 magneto-restrictive compass and provided reasonable outputs for the sensor (Marco et al., 2001). However, a compass has the disadvantage of induced errors from the true heading due to magnetic field variations caused locally around and within the vehicle. Additionally, there will be a deviation from true North based on location of operations that must be corrected for. As a result the compass would lose accuracy over time and produce significant navigational errors.

The accuracy achievable with the compass is a direct result of the ability to conduct calibrations with the compass and vehicle as well as a reasonably accurate and updated deviation table used for corrections. Another difficulty in utilizing the compass was that the heading bias learned was dependent on heading and thus required extensive use of the deviation tables (Kragelund, 2006). The calibrations, if successful, significantly reduced the error that resulted in the navigation filter but they were difficult to obtain good runs and required a considerable amount of time.

B. INERTIAL MOTION UNIT INSTALLATION

For this work a Honeywell HG1700 IMU was purchased and installed in the vehicle. The HG1700 is a strap-down type of IMU and is considered a low cost military grade production unit. The location was selected to place the unit as close to body center as possible without displacing elements of the vehicle previously installed. The inertial motion unit and associated electronic circuitry represent the only hardware modification associated with this work.

The IMU was installed in the aft compartment mounted vertically against the forward bulkhead. Figure 6 illustrates the installation of this unit.

Figure 6. IMU Installed in ARIES

The unit was centered on the body lateral and vertical axis and mounted in place for the output values to correspond directly to the body local x, y and z axis. The body defined axis' for the ARIES uses traditional orientations with the x-axis along the body centerline, the y-axis is athwartships on the body and the z axis is defined positive in the downward direction.

The unit utilizes +/- 15 volt DC and +5 volt DC input power for the associated electronics. This power is tapped off of the main battery bus power and stepped down from 60 volts to the required input power using two Calex DC-DC converters shown in Figure 6. The larger one supplies the +/- 15 volt DC power while the smaller supplies the +5 volt DC power (Kragelund, 2006).

IV. SOFTWARE IMPLEMENTATION

A. INTRODUCTION

Once the hardware was installed and the results of the hardware were analyzed, attention was turned towards the software architecture of the navigation filter realizing that to achieve the design goal of one percent of distance traveled both hardware and software improvements were going to be required. The approach taken was dual in nature utilizing simulation and field experiments. The EKF presented in Chapter II is embedded in the navigation code and is often referred to as the navigation filter. The terms EKF and navigation filter are synonymous with respect to this work.

This chapter will present the changes made to the system software as well as what was in use for the operating system. Additionally, the method in which MATLAB® was used to enhance the result of this research for the UKF and EKF will be shown. The order of the following sections represents modifications as performed in chronological order. The results obtained from these changes are shown in Chapter V in the same chronological order

1. Vehicle Operating System

The ARIES vehicle utilizes a QNX real time operating system with code written in ANSI – C for both the tactical and executive computers. The system operates on an 8 hertz (Hz) data cycle and the measurements obtained for state information are asynchronous. GPS data, when available, have a frequency of 1 Hz. Speed information from the RDI Doppler is provided at a frequency of 2 Hz. Lastly, the IMU provides data at 100 Hz but is selectively chosen over each 8 Hz cycle. This will be discussed more in section B.

2. MATLAB® Simulation

For simulations, a MATLAB® code was developed based on previous work by A. Healey, which modeled the navigation filter in the vehicle and utilized data obtained from

experiments. This code was then manipulated to determine effects on navigation performance from changes to the filter. The MATLAB® version of this code is contained in Appendix B.

3. Unscented Kalman Filter

In recent years there have been many attempts in the literature to improve upon the EKF due to the difficulties in implementation and tuning. The EKF requires reasonable determinations of the process and measurement noise for the system in order to implement; this can be rather difficult and may require much time to achieve optimal performance.

The Unscented Kalman Filter (UKF) was developed by Simon Julier and Jeffrey Uhlmann (1997) as an alternative method to the EKF. This filter characterizes the mean and covariance parameters by the use of a set of discrete points (Julier et. al, 1997). These points are then propagated through time and the mean and covariance are reconstructed to provide the updated estimate of the state. This filter does not require that the process or measurement noise be Gaussian and the Jacobian is not required to be calculated for the system. These last two are the major advantages for this type of filter over the EKF. The literature indicates that this particular application of Kalman Filter has shown better results for particular applications when compared to the EKF. Based on this latter claim a UKF was attempted for implementation in the navigation filter of the ARIES AUV.

A UKF, based on Julier and Uhlmann's work was developed, in MATLAB® for ARIES navigation by Dr. Hag Seong Kim, a research assistant with the AUV Research Group assisting with this work, utilizing data from experiments. The MATLAB® version of this filter is contained in Appendix C. This filter was run for a multitude of values for process and measurement noise and compared to the in-vehicle filter. Figure 7 shows the results of the UKF developed when compared to the EKF contained in the vehicle.

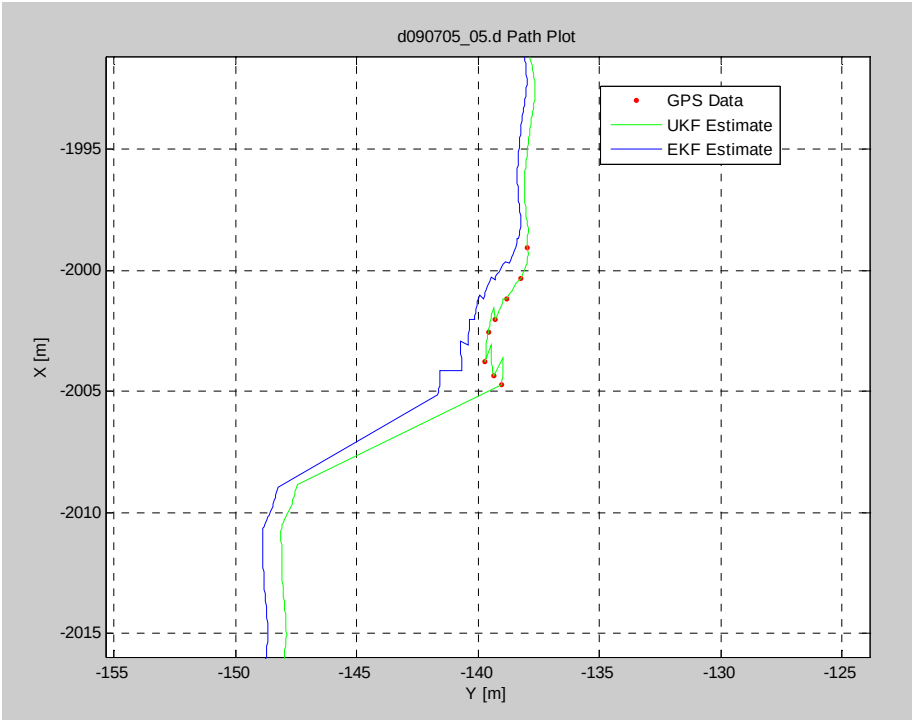

Figure 7. UKF vs. EKF for ARIES Run – 9/7/05

It was determined that, for this vehicle and application, the UKF produced results and errors on the same scale as the EKF without sufficient improvement in accuracy. It was noted that the UKF provided tighter estimates of position when updated with GPS and the same results can be observed with the speed components where less filtering occurs and more of the dynamics are retained in the estimates. However, when later work was performed with tuning the vehicle EKF, better results were obtained. Therefore, further work in code development and implementation in the vehicle was not warranted for the UKF. It can be said that the implementation of the filter was slightly easier since derivatives were not necessary, however, the same difficulties, i.e., determining appropriate process and measurement noise matrices, encountered in tuning the EKF, would be the same for the UKF.

4. Vehicle Code

There were two primary areas of vehicle code that were both developed and modified in order to install and make usable the data from the IMU. The first was an

operating code for the IMU that would control and convert the data flow from the IMU to be used by the navigation filter in the onboard computers. The original IMU code was developed by Jack Nicholson, CAPT, USN during the installation of the IMU. This code is contained in Appendix D. The second was the navigation filter code which contained the routines of the EKF that developed vehicle state information at each time step. A copy of this code is contained in Appendix E.

The ARIES operating system operates asynchronously accepting sensor inputs from multiple locations at different frequencies. The vehicle operates at a high enough time constant to be able to ensure no aliasing occurs among the sensors and the data. The vehicle time constant can be estimated to be:

$$\tau = \frac{L}{U} \approx \frac{3m}{1.5m/s} = 2\sec \tag{13}$$

This leads to a dynamic frequency for the vehicle of:

$$f_{\max} = \frac{1}{2\sec} = 0.5\ Hz \tag{14}$$

Therefore according to the Nyquist Sampling Criterion:

$$f_{sample} \geq 2 \cdot f_{\max} = 1\ Hz \tag{15}$$

The conclusion here is that as long as the sensor inputs occur at a rate greater than 1 Hz, the vehicle dynamics are slow enough to ensure that adequate sampling of those dynamics by the sensors are sufficient. This is important from the standpoint of utilizing IMU data to be discussed shortly.

B. INERTIAL MOTION UNIT CODE

The IMU code used by the ARIES was developed to handle the flow of data obtained from the Honeywell HG1700. The IMU provides inertial information for linear acceleration and angular rotation rates in three dimensions for the vehicle in the local body fixed frame. The code converted the readings into the local navigational tangent plane and corrected the measurements for the Earth's rotational rate. While three

dimensional inertial information can be obtained from the unit, it was the heading rate that was of primary concern for the navigation filter. The heading rate was integrated within the code using a simple Euler integration scheme over the 8 Hz time interval. The simple integration scheme was justified based on the short time interval compared with the rate of vehicle motion. The result of this integration was a sufficiently accurate heading input to replace the compass heading previously utilized.

The IMU code provides for the necessary rotations and adjustments that are necessary to obtain truly inertial measurements from the IMU. The accelerometers and gyros associated with the IMU measure the acceleration and rotations felt upon the sensor, and not all of the measurement is due to body motion of the vehicle. The IMU will sense the rotation of the Earth frame in which the vehicle is moving. The measurements thus obtained from the IMU must be corrected for the Earth's sidereal rotation rate. Once the IMU information has been rotated and corrected it can then be fused with other data obtained in the same reference plane with which position estimates can then be made.

One of the primary calculations needed is to rotate the Earths angular rotation rate of approximately 15.04 degrees per hour into the body fixed frame. This is necessary to obtain purely inertial measurements for the vehicle since the local navigation plane is rotating. Figure 8 provides the illustration showing the relationship between the navigation plane the vehicle moves in with the Earth's rotating reference frame. The local navigation plane oriented in the North-East-Down configuration rotates at the Earths sidereal rotate and this rotation is felt upon the gyros in the IMU.

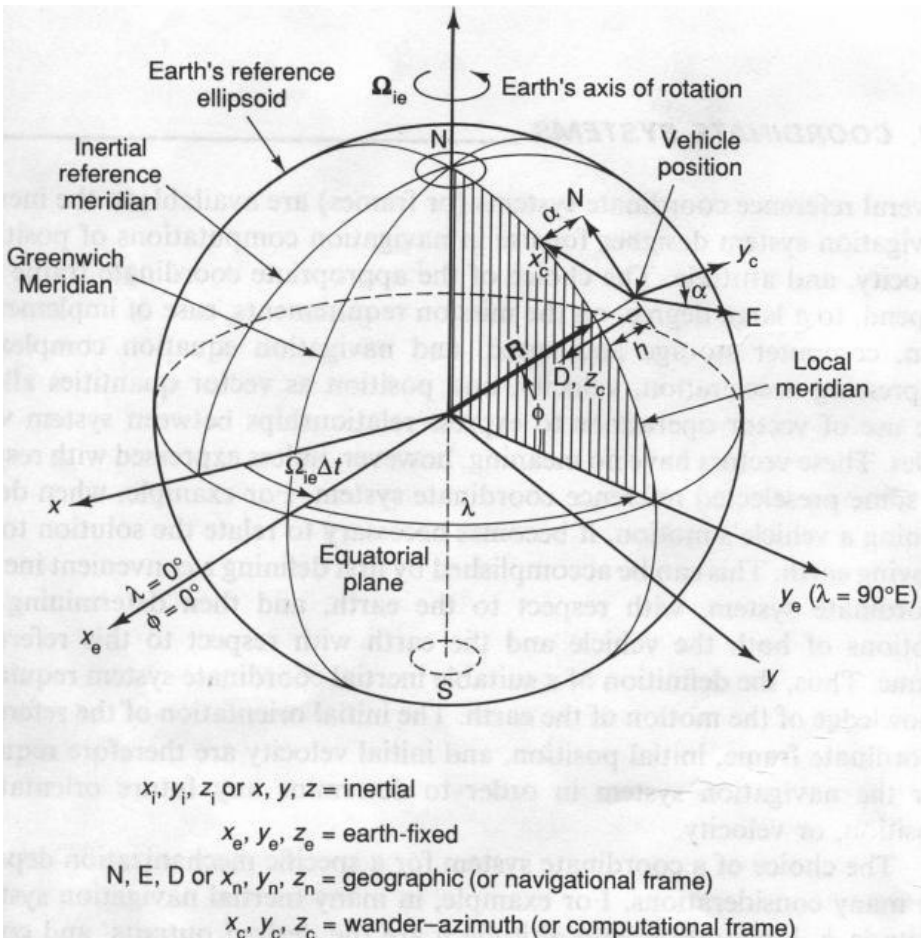

Figure 8. Earth Frame of Reference (From: Yakimenko, 2006)

Once rotated, the elements corresponding to the Earth's rotation can be removed from the measurements for angular rates within the IMU. Only those elements needed to obtain the heading rate were calculated. The equations and process used are outlined as follows and are contained in the IMU code in Appendix D.

1) The Earth's sidereal rotation vector [0,0,Ω]' is rotated into the body fixed frame $[p_e,q_e,r_e]$' using the current geodetic latitude of the vehicle and the vehicle body Euler angles (φ,θ,ψ).

$$q_e = \left[\text{-}\cos(\phi)\cdot\sin(\psi)\cdot\cos(\text{Lat})\text{-}\sin(\phi)\cdot\cos(\theta)\cdot\sin(\text{Lat})\right]\cdot\Omega \tag{16}$$

$$r_e = \left[(\sin(\phi)\cdot\sin(\psi)+\cos(\phi)\cdot\sin(\theta)\cdot\cos(\psi))\cdot\cos(\text{Lat})\text{-}\cos(\phi)\cdot\cos(\theta)\cdot\sin(\text{Lat})\right]\cdot\Omega \tag{17}$$

2) The elements of the Earths sidereal rotation rate are then removed from the angular rate measurements from the IMU gyros [p_{out}, q_{out}, r_{out}]'.

$$\begin{aligned} q_{out} &= q_{out} - q_e \\ r_{out} &= r_{out} - r_e \end{aligned} \tag{18}$$

3) The adjusted angular rate values which represent the true angular rate of the vehicle with respect to a true inertial frame can be used to obtained the heading rate ($\dot{\psi}$).

$$\dot{\psi} = \frac{\sin(\phi)}{\cos(\theta)}\cdot q_{out} + \frac{\cos(\phi)}{\cos(\theta)}\cdot r_{out} \tag{19}$$

4) The heading rate is then integrated using Euler integration with the result being an updated heading value at the i+1 time step.

$$\Delta\psi = \dot{\psi}\cdot\Delta t \tag{20}$$

$$\psi_{i+1} = \psi_i + \Delta\psi \tag{21}$$

The output of the IMU is at 100 Hz, i.e. there are 100 readings of all channels per second. Originally, twelve readings were averaged over an 8 Hz period in order to align the IMU output with the operating system. Checks for data stream integrity were utilized in the form of identification of a unique hexadecimal identifier that indicated a new 44 bit data stream. During the research period it was identified that the data streams were becoming corrupted and providing invalid information over some of the read cycles. This was contributing to errors propagating into the position estimates.

It was determined that, due to the errors developing in the data streams over an 8 Hz period, instead, a single 44 bit data stream would be used as the average for the interval. Additionally, more rigorous data checks would be placed on the data to ensure that it was a complete and valid data message. These checks included continuing to identify a new message by the unique identifier and the use of a checksum at the end of the data stream to ensure that the preceding 42 bits of data were correct and belonging to the same message. The new message identifier and checksum are IMU manufactured features in the data message. The use of a single reading of the 100 Hz data in each 8 Hz cycle was reasoned to be appropriate because the vehicle dynamics did not change rapidly enough to have significantly different readings from interval to interval and the sampling frequency was still greater than the system dynamics frequency as discussed above to justify this averaging scheme.

C. NAVIGATION FILTER CODE

The majority of the code changes were made in the navigation filter that contains the EKF in the vehicle. The next few sections will provide the methods by which the navigation filter code was changed with the results shown in Chapter V.

1. Gyro Rate Bias

The EKF originally used was developed for eight states as previously discussed. After examination of the first run utilizing the IMU on September 7, it was observed that the filter was learning a gyro rate bias that varied greatly and was significantly greater than the manufacturer specified maximum gyro drift rate of one degree per hour. Thus, it appeared as if the filter was developing a larger gyro rate error than the maximum specified. The correction for this was zeroing out the associated gyro rate bias state in the filter. This was accomplished by setting the rows and columns of the observation matrix to zero and to set the gyro rate bias state to zero during the calculation cycle.

2. Adaptive Process and Measurement Noise Algorithms

There has been much work performed to find optimal algorithms that can adaptively determine the values for the process and measurement noise matrices. Much of the available literature researched indicates that most of the work is based on ideas proposed by Mehra (1970) or work by Myers and Tapley (1976). The method originally proposed by Mehra (1970) as modified and presented by Busse et al. (2002), was attempted for use in the navigation filter for the ARIES.

Successful results have not yet been obtained for this application. The literature suggests that the algorithms are highly dependent on tuning factors in the schemes. Therefore, a purely adaptive routine has not been found for this particular application and the pseudo-adaptive routine developed in this work continues to provide the best results with respect to navigational accuracy.

3. Pseudo-Adaptive Measurement Noise Matrix

The last major innovation to the EKF was the tuning of the weights for measurement noise values used in the algorithm. Prior to this work the values for both process noise and measurement noise values were assumed to be constant and obtained experimentally over significant periods of operation with the vehicle and as such were a very good first guess to the optimal values to use in the filter.

The measurement vector is a six state vector in the navigation model consisting of position, both X and Y, forward and lateral speed over ground and heading and gyro rate. It was postulated that by selectively forcing the weights of specific states up or down that the overall filter would place more or less emphasis on the measurements for that state. It has been observed that the Kalman Gains are directly proportional to the weights used for the process noise and inversely proportional to the weights for the measurement noise values.

$$Kalman\ Gain\ (L) \propto \frac{Q}{R} \tag{22}$$

Equation (22) was used as a guide to adjusting the weights from their previous values using the simulation EKF in MATLAB® and the effect on position estimates compared to GPS positions were evaluated for all available data sets. Specifically, the pseudo-adaptive routine developed reduced the measurement noise weighting values used for the position states, thereby forcing the filter to rely more heavily on the information from the GPS fixes. The measurement noise matrix is 6x6 and only the diagonal values associated with the fifth and sixth state of the measurement vector corresponding to the position information is altered each time step through the filter based on the number of satellites seen by the GPS receiver.

The algorithm developed for the code altered the measurement noise values for the position states by reducing them by an order of magnitude as the number of satellites visible from the GPS receiver increased. The original values were used for three or less satellites visible, then reduced an order of magnitude for four satellites, the first position information in which a GPS fix is declared and reduced another order of magnitude for five or more satellites.

This combination was run on several of the data sets from the May, June, and July timeframe of 2006 and the effects on estimated position when compared with the current in-vehicle output as well as the GPS positions. Figure 9 shows the results of this algorithm on the data obtained from Run 3 on July 25. The improved filter performance can clearly be seen when compared with the GPS position after a two kilometer run. After simulation runs on different data sets, confirming the results observed in Figure 9, the algorithm shown in Figure 10 was implemented in the vehicle for field evaluation.

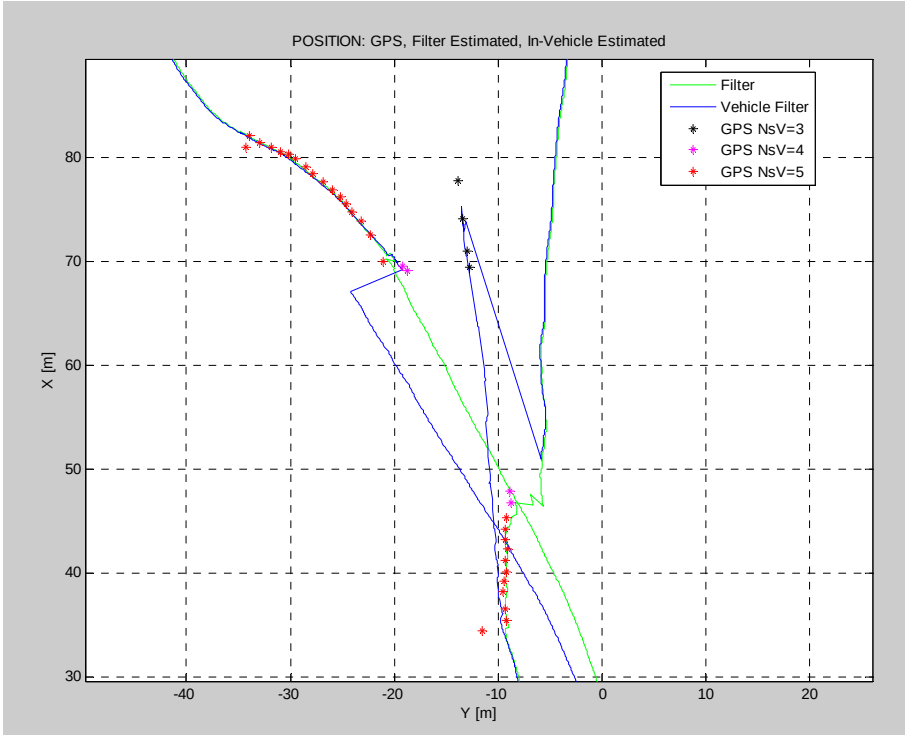

Figure 9. Simulation with Pseudo-Adaptive Algorithm of Run 3 - 7/25/06

```
// SPK 081606: Set elements of R Matrix based on
     NSv (as verified by SRV  by 08/2006)
if (NSv <= 3)
   nu[5]=1.0;
   nu[6]=1.0;
else if (NSv == 4)
   nu[5]=0.1;
   nu[6]=0.1;
else if (NSv >= 5)
   nu[5]=0.01;
   nu[6]=0.01;
 /* Update Diagonal R Matrix */
   for(i=1;i<=7;++i)
      R[i][i] = nu[i];
```

Figure 10. Pseudo-Adaptive Algorithm for Measurement Noise

Originally, the filter and the GPS software utilized a set point of three GPS satellites as being sufficient fix quality to use in the filter, however, it was observed that the use of three satellites produced a significant error in the solution as shown in Figure 11. Therefore, the GPS code was changed to declare GPS information available if the

number of visible satellites to the GPS receiver was greater than or equal to four satellites. This higher threshold ensured the GPS receiver was obtaining higher quality fix information.

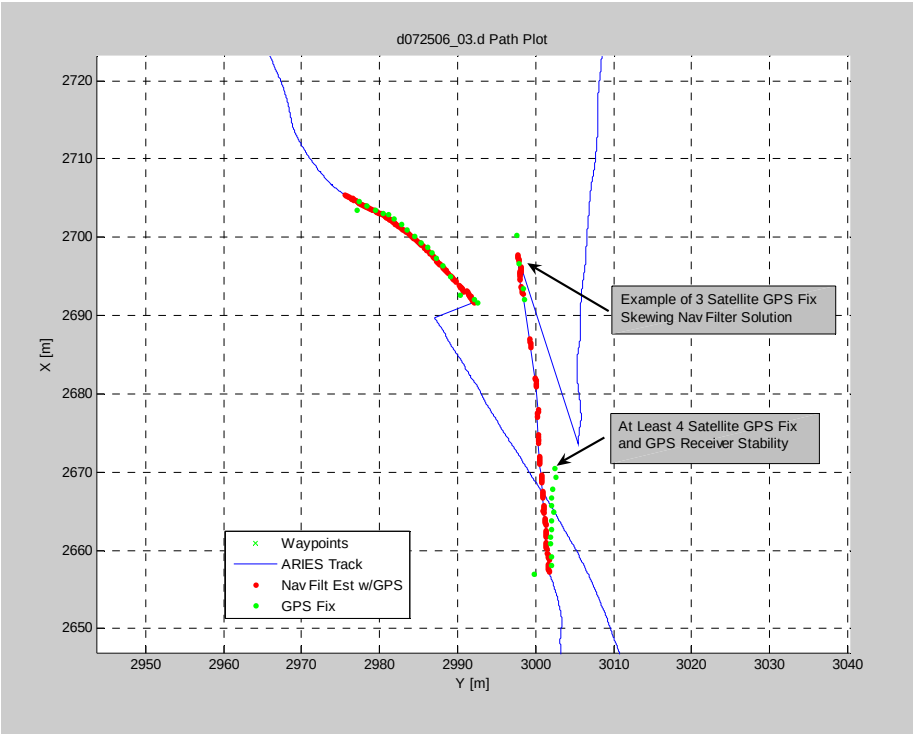

Figure 11. Three vs. Four GPS Satellite Fix

In addition to changing the measurement noise values for the position states, computer simulations were run altering only the speed noise weighting values to observe the effects on filter performance with respect to position estimates. The weighting values were changed individually for the forward speed measurement and the lateral speed measurement by orders of magnitude until satisfactory performance was achieved qualitatively in simulation. These values were then checked against other data sets to ensure that acceptable performance was maintained in terms of accurate position estimation compared with GPS. The result was that the forward speed measurement noise weighting value was reduced by two orders of magnitude and the lateral speed measurement noise weighting value was altered by a single order of magnitude, placing more reliance on the speed measurements overall.

Observing the resulting filter speed estimates when compared to actual measurements it was seen that the effect on the speed states was reduced smoothing of the data. The speed measurements from the RDI Doppler and the estimations of vehicle speed from the EKF in-vehicle compared to the new estimates using the modified weights are shown in Figure 12 and Figure 13. The red cross marks are the measured values for speed from the RDI Doppler speed sensor, the blue dots are the in-vehicle EKF estimates from that run and the green dots are the new estimates from the simulation EKF after modifying the noise values.

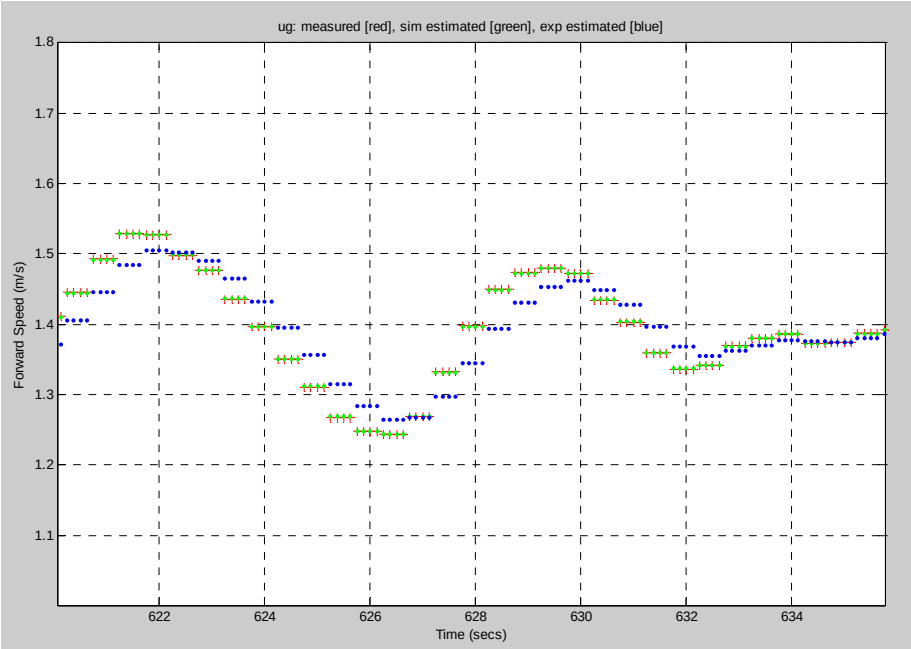

Figure 12. Forward Speed Filter Results of Run 3 -7/25/06

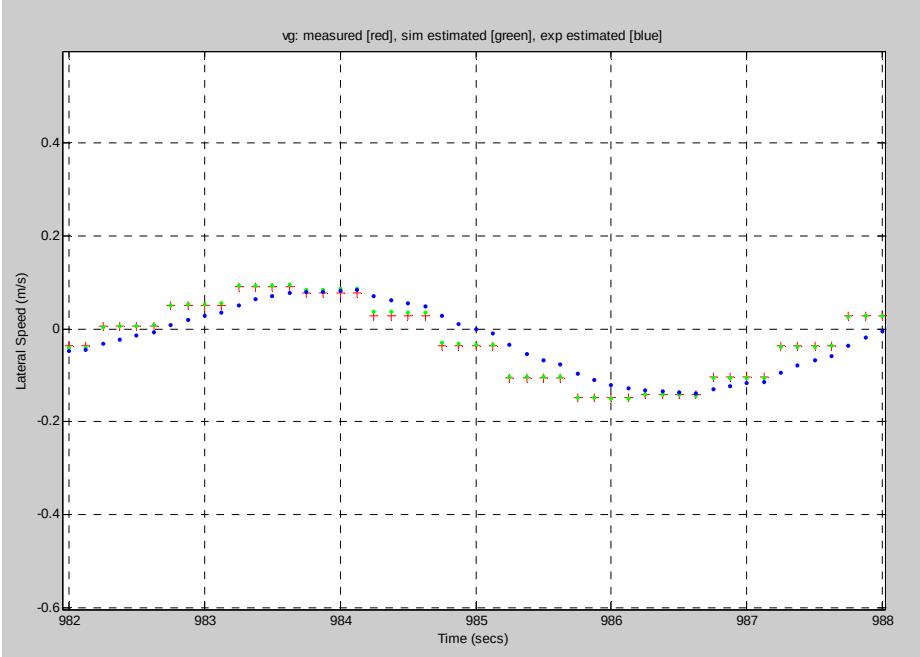

Figure 13. Lateral Speed Filter Results of Run 3 -7/25/06

The observation drawn here is that what appears to be noise in the measurements and captured as such in the system model could be actual slight changes in vehicle speed due to external forces, possibly from wave or current action or alterations in which the thrusters propel the vehicle through the water. It would thus introduce slight errors to model this information as noise vice actual variations in the speed measurement.

An additional feature of the fusion filter with the pseudo-adaptive algorithm is that it is more responsive to the heading bias learning process. This phenomenon can be observed in Figure 14; while not as smooth, it converges to the actual necessary initialization error quicker than previous runs due to the increased weighting on the position measurements. The oscillatory nature of the heading bias state early in the run is from speed information that is approximately zero when the vehicle is on the surface at the beginning of a run awaiting mission execution commands but with the navigation process running.

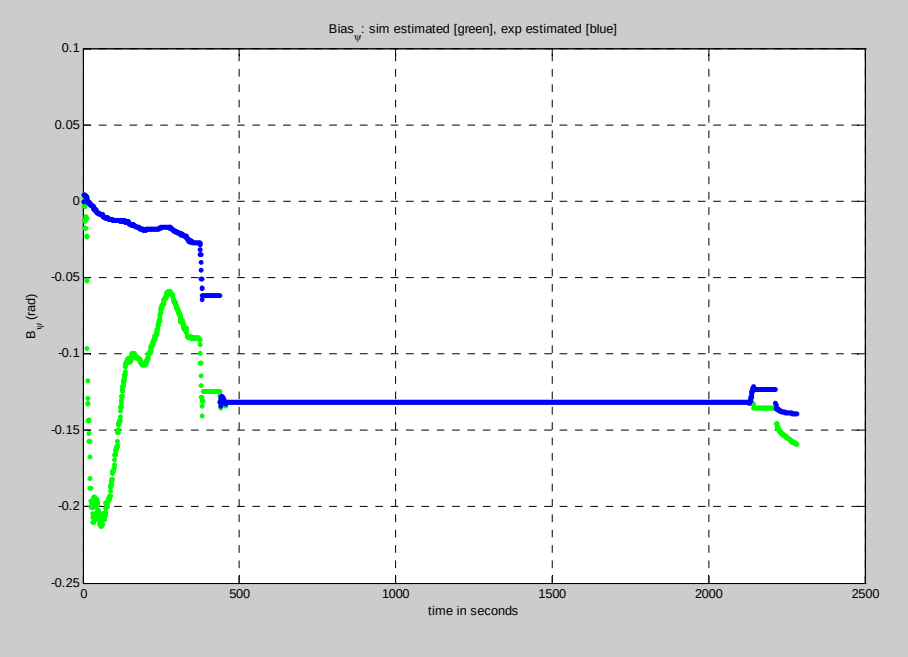

Figure 14. EKF Bias Learning with Pseudo-adaptive R Matrix – 7/25/06

Lastly, it was observed that once the vehicle started its thrusters and began a mission with an increase in speed the heading bias converges quickly to a steady state value that is very close to the final learned heading bias. This is due to the fact that accurate position information is still being obtained via GPS fixes prior to submerging and there is a non-zero speed component with a relatively accurate heading input. The heading bias as the only unobserved state during this time frame which converges rapidly within this estimator with all other states observed.

One last navigational modification was made based on this last observation after the August 24 run. A time delay of 10 seconds was inserted into the executive process of ARIES to delay submerging after thrusters were initiated. With this delay, the filter would have all the state information necessary for accurate heading bias estimation thereby eliminating needs for subsequent pop-ups. The value of this would be enhanced mission effectiveness by eliminating the need for a GPS fix to correct the heading bias prior to entering an operating area to conduct its mission.

THIS PAGE INTENTIONALLY LEFT BLANK

V. EXPERIMENTS AND RESULTS

A. INTRODUCTION

In order to understand the effects of the implementation of the IMU and modifications to the vehicles software architecture presented in Chapters III and IV, experiments with the vehicle were conducted in Monterey Bay with the ARIES vehicle from September 2005 through September 2006. These tests were established to run the vehicle in a controlled geometry with varying GPS pop-ups in order to quantify the navigational position error between the filter estimate and the vehicles position as determined by the GPS receiver.

B. EXPERIMENT GEOMETRIES

Experiment geometries were maintained rather simple and consistent to provide a basis for rapid qualitative comparison in-situ as well as post analysis quantitative comparison. This was done by utilizing a straight run of lengths of one kilometer initially and then eventually expanding to two kilometer runs and finally a four kilometer run to validate the findings. The runs were oriented on a North-South axis for simplicity of track planning and safety of vehicle in order to run parallel to the shoreline in Monterey Bay. The majority of runs were set to run at a depth of three to four meters unless other testing requirements dictated that the vehicle run on altitude control utilizing the bottom topography vice a commanded depth input.

1. GPS Pop-up Maneuvers

The number of GPS pop-ups varied from as many as approximately eight on the early runs to two on the final runs towards the end of the test period. The use of many GPS pop-ups allowed for statistical quantification of the mean error between filter estimate of position and actual position. Additionally, the use of many pop-ups allowed for analysis of filter performance with respect to estimating the gyro rate bias and the heading bias throughout the run.

2. Geometry Modifications

There were two modifications to the geometry described above during the experimental phase. First, a multi-heading, multi-turn geometry was utilized in order to qualitatively observe performance of the vehicle IMU and filter under a rather intense maneuvering scheme. This run can be seen below in Figure 15. The IMU performed excellently as a heading reference on multiple courses over this run without incurring additional errors that otherwise may have been incurred with a compass. Secondly, the final four kilometer validation run was performed on an East-West cardinal heading following a two kilometer North-South run to demonstrate that the results were independent of vehicle heading. Additionally, this ensured no artificial bias may have existed in the results obtained by the selection of repeated North-South runs.

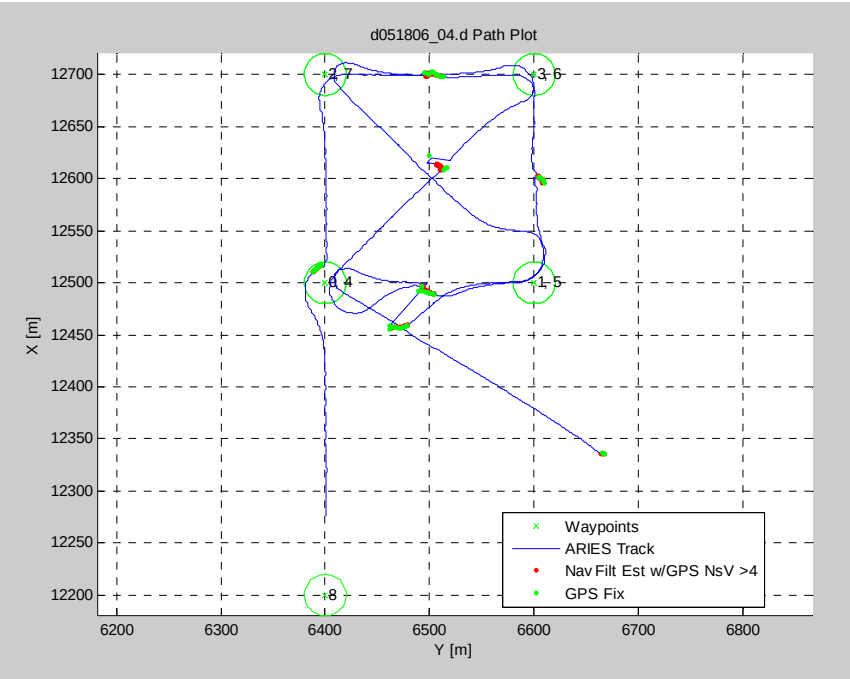

Figure 15. Multi-Heading Run Geometry – 5/18/06

C. EXPERIMENT RESULTS

The following sections will present the results of the modifications discussed in Chapters III and IV following the same chronology.

1. Original Navigation Filter

The original navigation filter utilized a compass as the heading reference input and the navigational errors incurred were relatively large. Figure 16. and Figure 17. show the evolution of the mean errors with each major change to the navigation filter as well as the mean error per unit distance traveled.

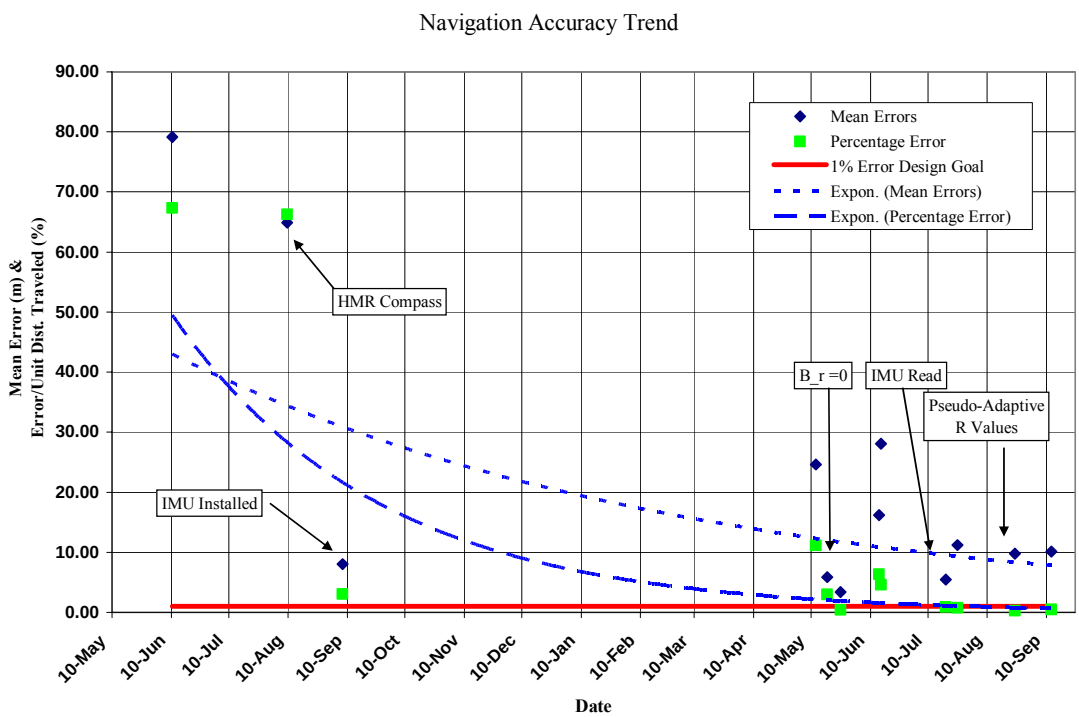

Figure 16. Evolution of Navigation Filter Errors

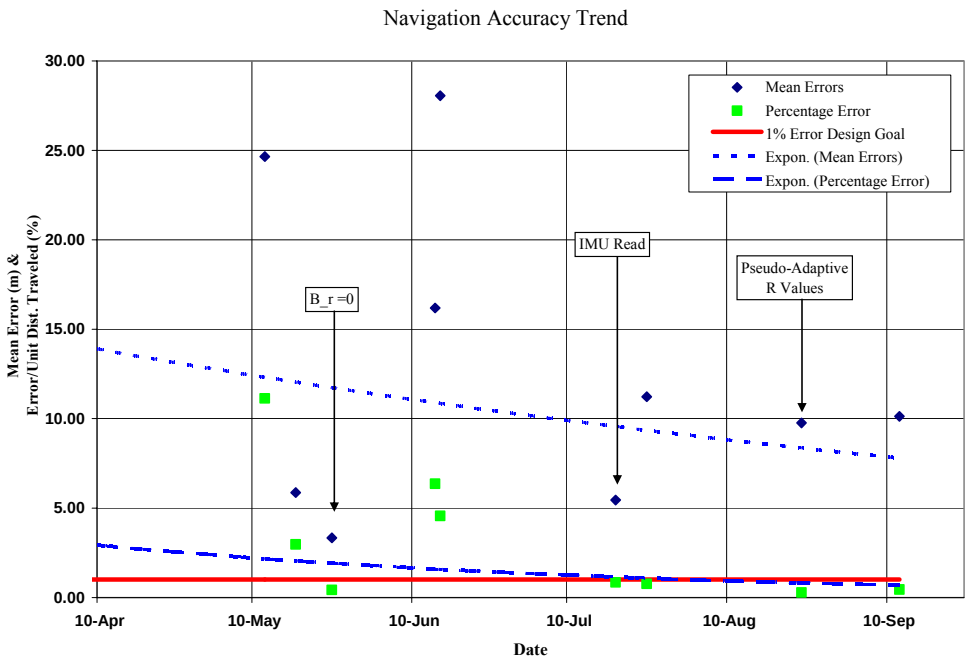

Figure 17. Expanded Evolution of Navigation

Figure 18 and Figure 19 illustrate typical tracks from the use of the compass as a heading input to the navigation filter.

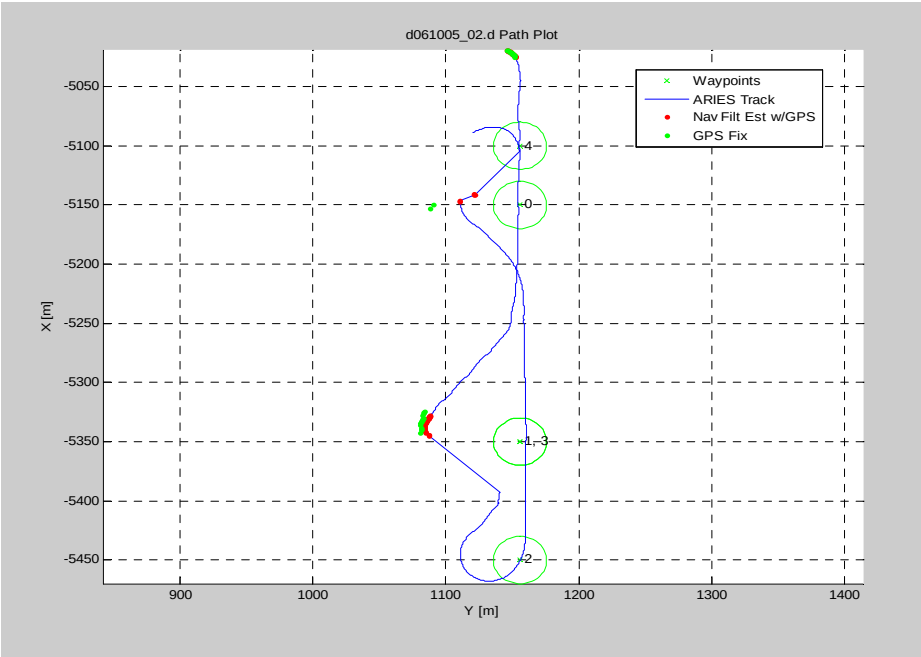

Figure 18. Compass Based Track – 6/10/05

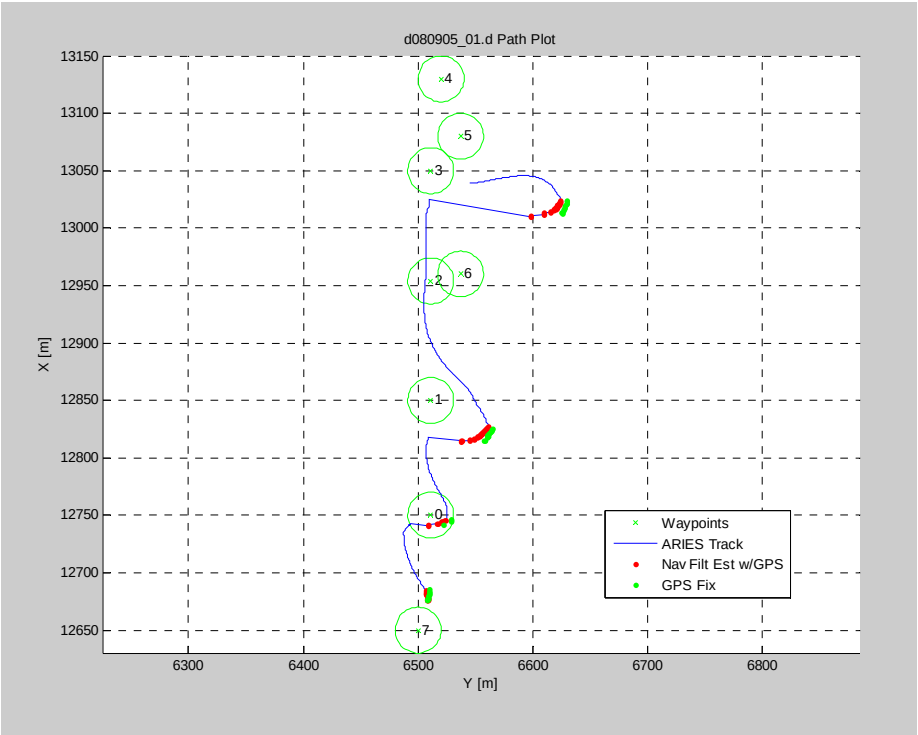

Figure 19. Compass Based Track – 8/9/05

The navigational errors were large and highly dependent on the quality of the compass calibration and the ability of the heading bias learning process to correctly obtain the initialization error. With the compass as the heading reference, the baseline mean error was 70.59 meters and the error per unit distance traveled was 66.67%.

2. Initial Hardware Modification

The installation of the Honeywell IMU replaced the Systron Donner Motion Pak IMU as the source of the gyro rate which when integrated would provide the heading input for the filter to produce better estimates of position based on the equations of motion. The first run with the IMU saw significant improvements in the quality of the position estimates. Figure 20 shows the results of this run.

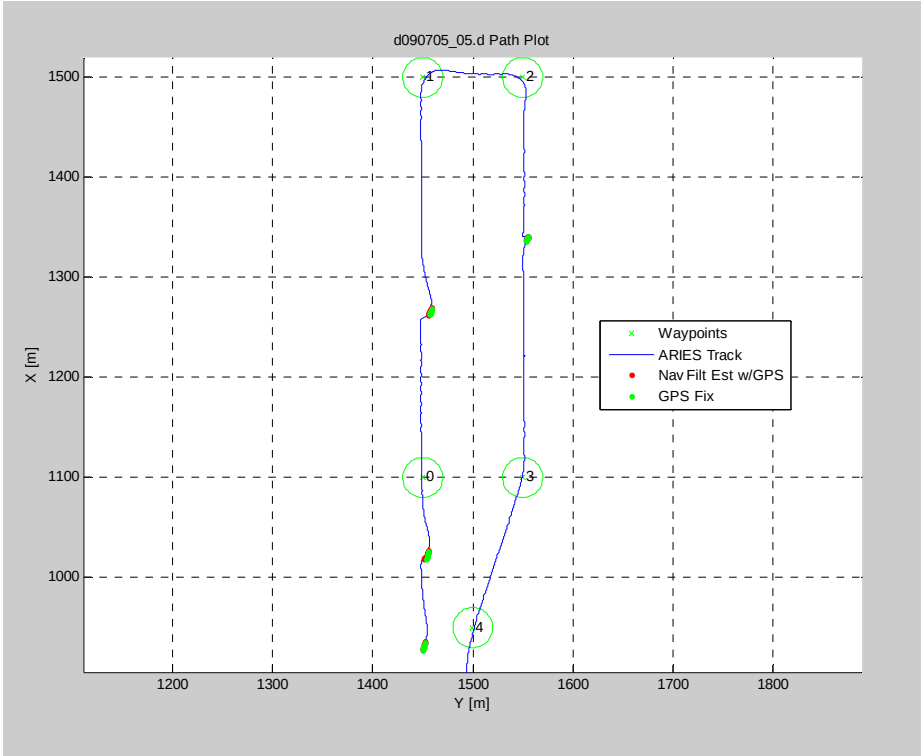

Figure 20. Initial Post-IMU Install Run – 9/7/05

There was a significant difference in navigation performance with the implementation of the IMU. Overall mean navigational errors were reduced to 8.04 meters and the error per unit distance traveled had been reduced to 3.04 % for this initial run with installation of the IMU.

3. Gyro Rate Bias Modification

The corresponding reduction in navigational error was an order of magnitude. Analysis of the September 7th run revealed that the rate bias state was producing rather large values, which when compared to the manufacturers published gyro drift rate for the IMU of one degree per hour indicated that there were additional errors being induced or captured by this state through the vehicle dynamics. The gyro rate biases obtained from the initial runs with the IMU are shown in Figure 21 and Figure 22.

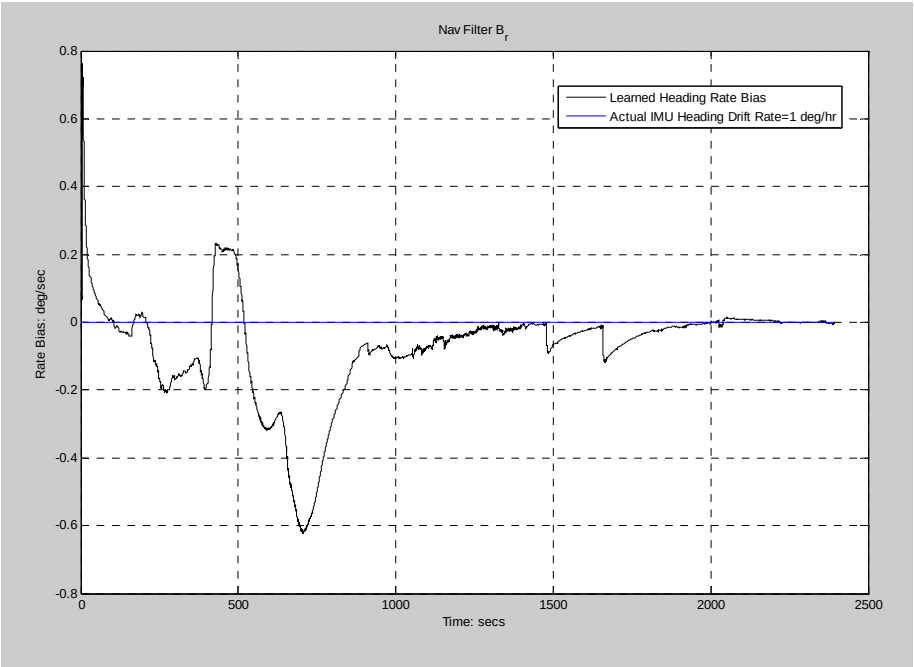

Figure 21. Navigation Filter Rate Bias – 9/7/05

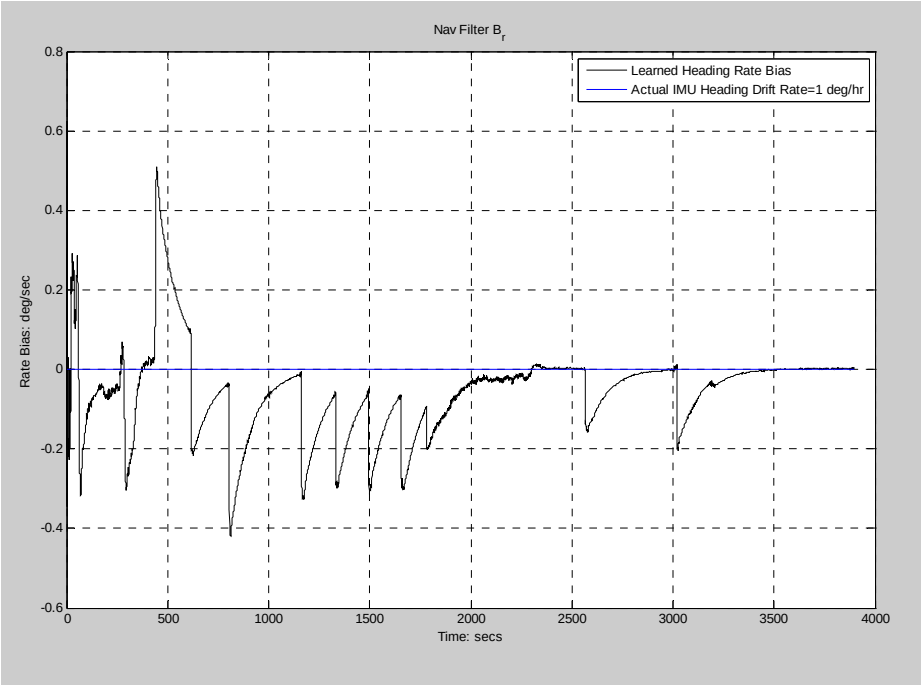

Figure 22. Navigation Filter Rate Bias – 5/12/06

Based on the observations of Figure 21. and Figure 22. above it was determined to force the rate bias state to zero within the filter to eliminate erroneous learning and thus introducing errors into the filter. The mean positional errors during the time period that the rate bias learning was activated and with the IMU installed was 19.23 m with an associated 8.69 % error per unit distance traveled. There was significant variability in the results as seen from the raw data in Table 2 of Appendix A.

With the rate bias state set to zero in the filter as discussed in Chapter IV, the ARIES was run again on May 25, June 14 and June 15 without any other changes to the navigation filter. Figure 23 is a typical result of the runs during this time period.

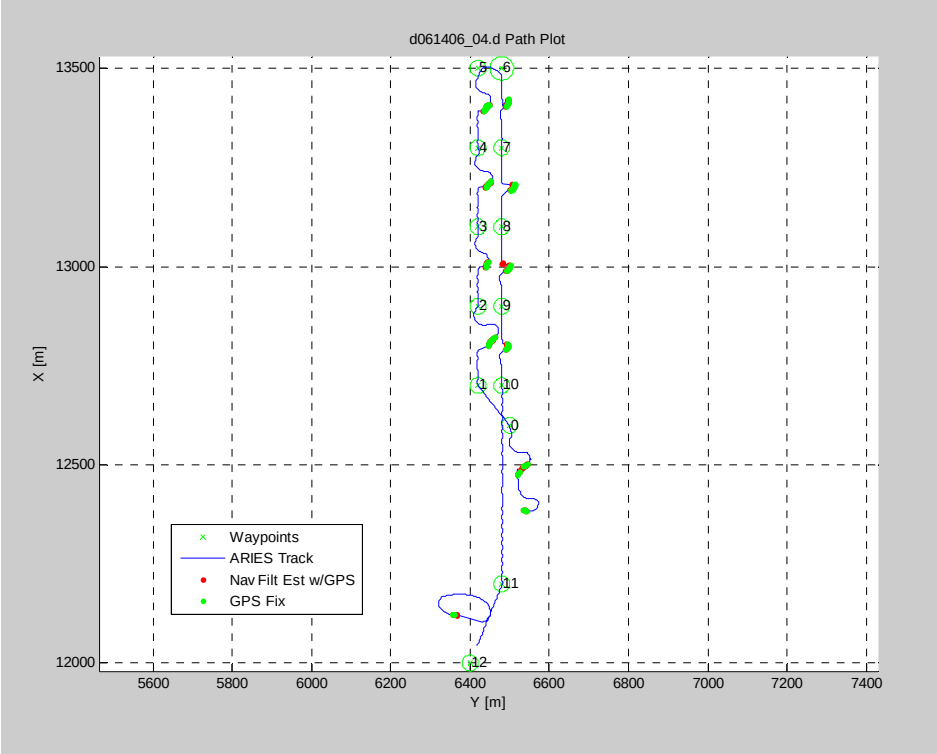

Figure 23. Typical Run with Rate Bias set to Zero – 6/14/06

The mean navigational errors during this period were 16.09 meters, which is not a significant change but the mean error as a percentage of the distance traveled decreased to 5.63 %. Again, there was much variability observed in the results but the trend is increasing downward.

4. IMU Code Averaging Scheme

During the June runs it was discovered that there were errors being introduced into the 44 bit data streams containing the measurements from the IMU. These errors, under the original architecture of the code, were being averaged over the 8 Hz period. The erroneous heading rate was then being integrated for heading for use by the navigation filter. The source of the errors was not discovered and the IMU code was changed as discussed Chapter IV. The ARIES was run on July 19 and July 25 with the results shown as follows in Figure 24 and Figure 25.

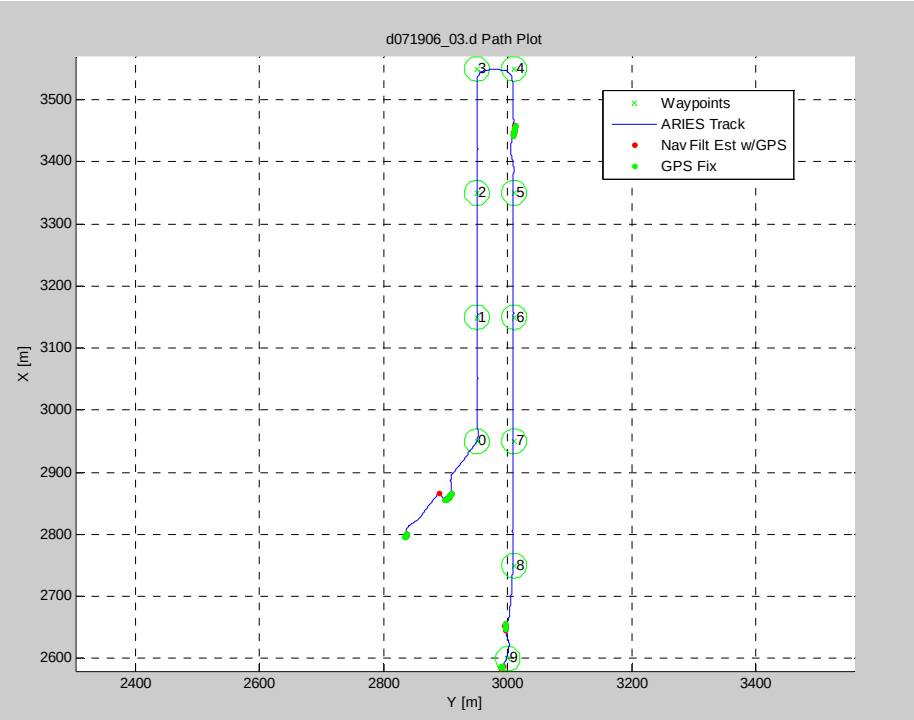

Figure 24. ARIES Run with IMU change – 7/19/06

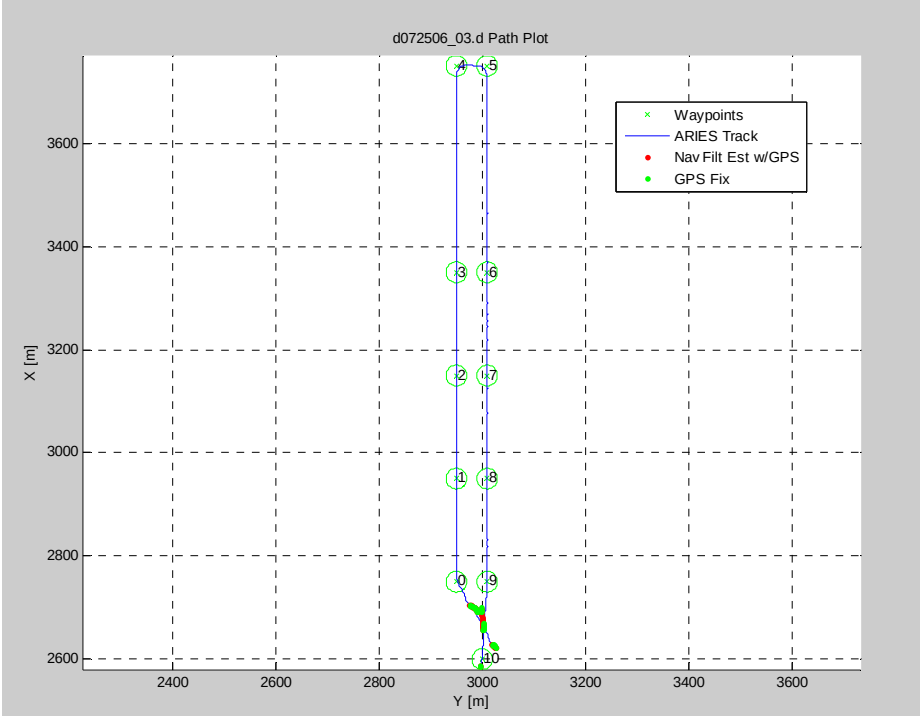

Figure 25. ARIES Run with IMU change – 7/25/06

A noticeable improvement was realized with this methodology which utilized a single good sample from the 8 Hz calculation interval. The underlying assumption made was that the true vehicle dynamics did not change quickly enough and that a single measurement made over the interval would be representative of the interval. During this period of runs, the mean navigational error improved to 7.92 meters and error per unit distance traveled was 0.81 %. The overall goal of one percent of distance traveled had been achieved but a review of Table 4 shows that there were some values that were right above the desired level.

5. Pseudo-Adaptive Measurement Noise Modification

The result of implementing the pseudo-adaptive noise algorithm were errors consistently less than one percent error over the distance traveled. An initial two kilometer run was conducted on August 24 to verify the results compared to previous runs of the same geometry and to validate what had been observed in simulation shown

in Figure 9. The total error after the two kilometers was 5.36 meters for a resulting percentage error of 0.23% of the distance traveled. This run is shown below in Figure 26.

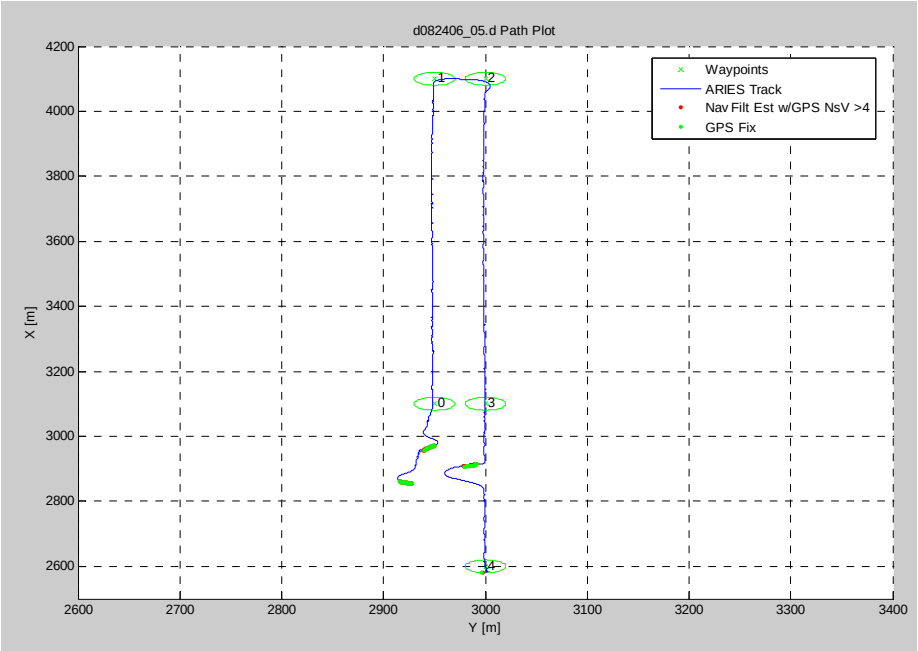

Figure 26. 2 km Run with Pseudo-Adaptive Algorithm – 8/24/06

A second run was then conducted to confirm the results obtained. For this run the geometry would be rotated so that the runs would occur on East-West cardinal headings. Additionally, the run would be doubled to four kilometers to observe how the filter handled the accumulated error due to the drift of the IMU gyros during this extended run. The four kilometer run is shown in Figure 27.

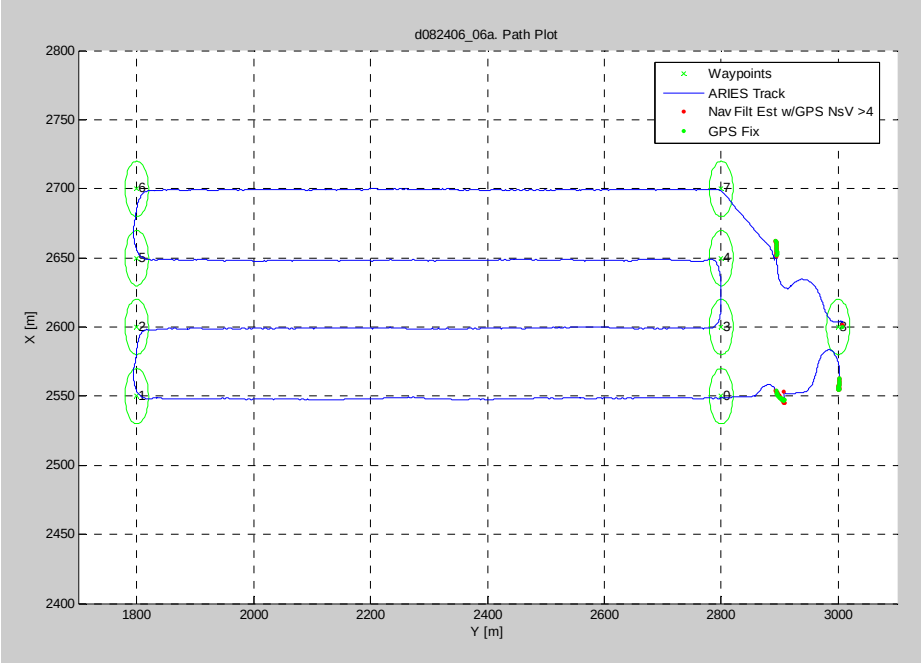

Figure 27. 4 km Run with Pseudo-Adaptive Algorithm – 8/24/06

The result for this run was an accumulated position error of 14.16 meters over the four kilometer run, resulting in a percentage error of 0.33%. This geometry demonstrated that the accuracy of the navigation was independent of any particular track heading.

6. Surface Time Delay Implementation

An initial ten second surface delay was inserted into the operating profile for the vehicle and tested on October 17. This based on observations made during simulation on the July 25 run as discussed in Chapter IV. The results of the surface delay are shown below in Figure 28. An enhanced view of the initial surface run is illustrated in Figure 29. The heading bias with accurate position information and non-zero speed information approaches a final steady state value, thus validating the hypothesis discussed in Chapter IV. The result of learning the bias quickly before submerging is that, if given the proper time, the filter estimates will converge with actual position by GPS which can be observed in Figure 30.

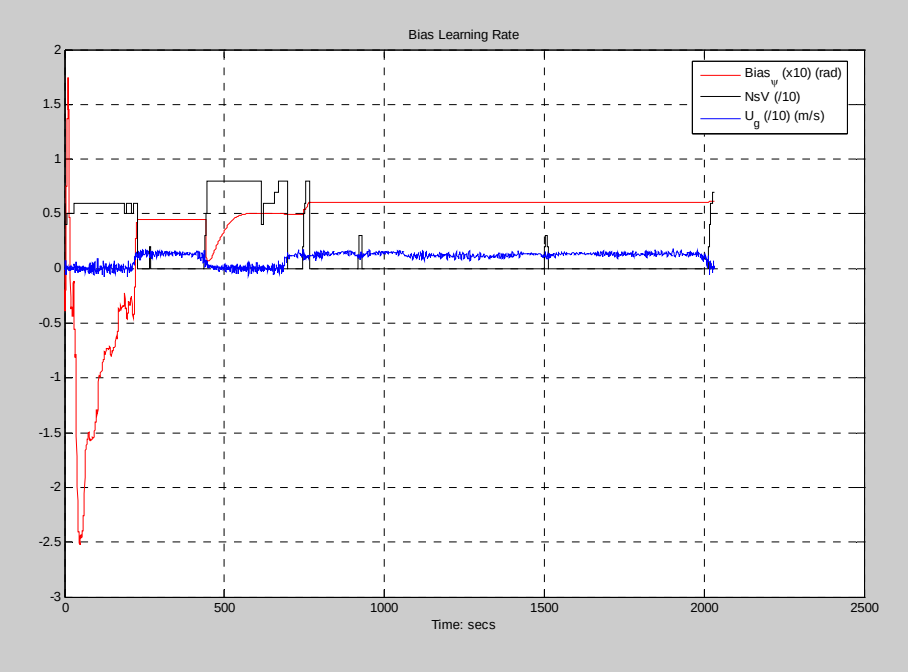

Figure 28. Bias Learning Rate after 10 second surface delay – 10/17/06

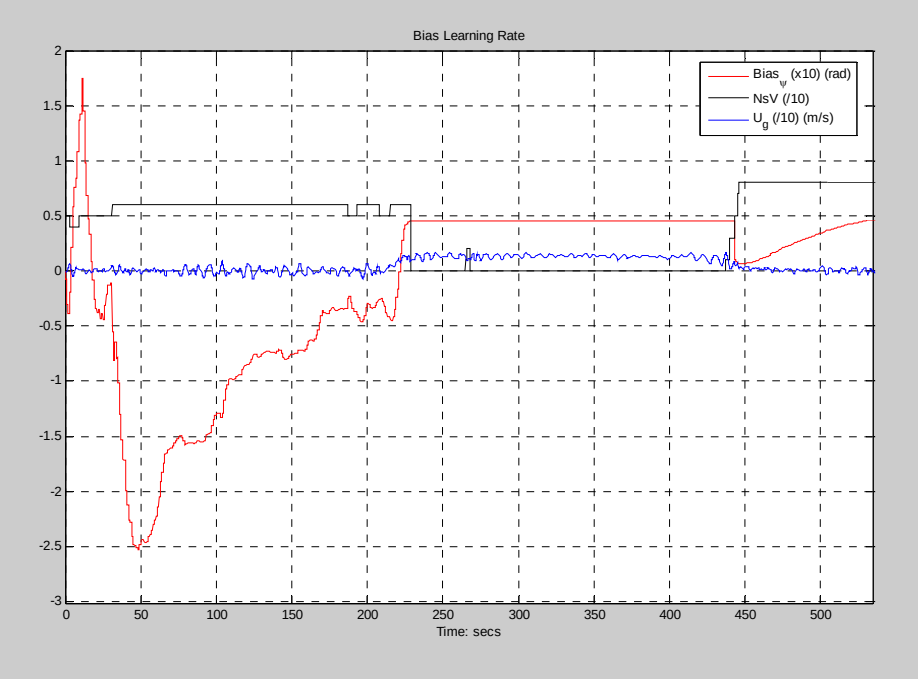

Figure 29. Enhanced view of Bias Learning – 10/17/06

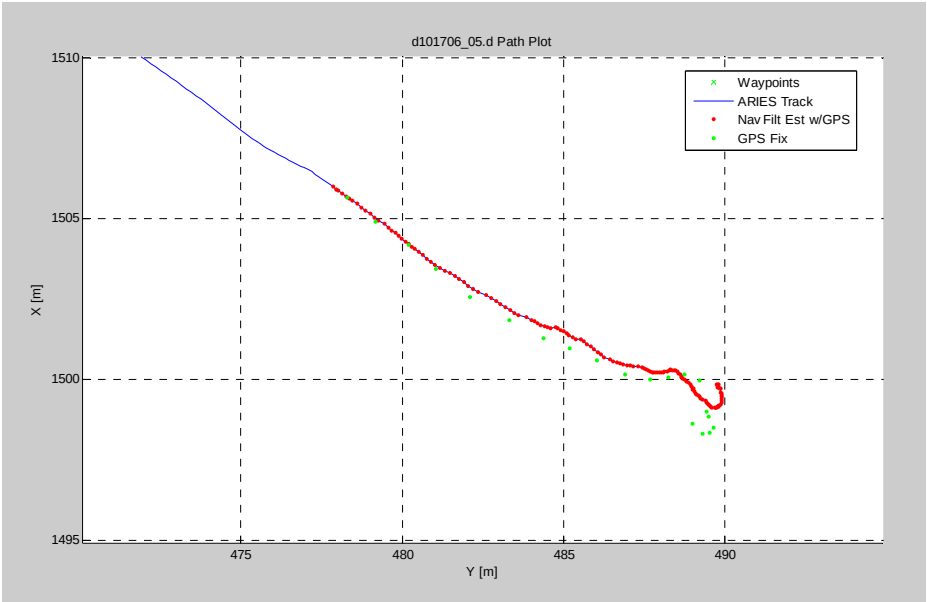

Figure 30. EKF Estimate and GPS Convergence – 10/17/06

The result of implementing the surface time delay and thus learning the heading bias while still surfaced is observed with an increase in accuracy of the initial GPS pop-up maneuver. The filter estimate when compared to GPS position for the first GPS pop-up can be seen in Figure 31.

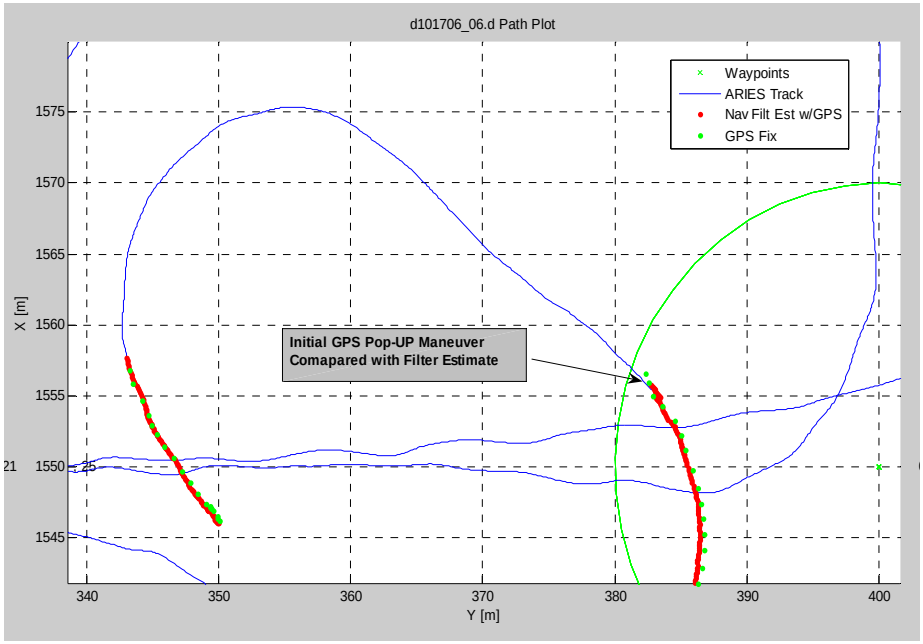

Figure 31. Initial GPS Pop-Up Maneuver with Surface Time Delay – 10/17/06

The work performed in this research has resulted in excellent navigational accuracy for the ARIES AUV. It will allow the ARIES to be used for work in which precise position information must be made available to the vehicle and will allow research that had been unable to be accomplished.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This research demonstrated that good navigational accuracy could be achieved through the use of a low cost IMU coupled with a properly tuned EKF. A one percent error as a function of distance traveled was achievable in this research. This was shown to be possible through the use of a higher quality IMU that had a lower gyro drift rate thus inducing smaller errors over time in the heading input for the equations of motion for the vehicle.

A pseudo-adaptive routine was developed that demonstrated how information obtained from measurements could be maximized based on the quality of the information by properly weighting the measurement noise values for certain states. In this manner better estimates of position were obtained as compared to the GPS positions with overall smaller errors. The better estimates of position were the result of a faster convergence to a particular heading bias for a given run. The pseudo-adaptive routine forced the heading bias state in the system to be more reactive and dynamic due to changes in the position, heading and speed states.

It was also shown how a navigation filter could be enhanced by changes in operating procedures. This was demonstrated through the surface time delay in which a highly reactive heading bias state could converge quickly to the necessary value by increasing speed while still on the surface receiving GPS information during the initial period of a run. The result is that with the heading bias learned prior to initial submergence, subsequent vehicle pop-up maneuvers to learn the heading bias could be eliminated.

B. RECOMMENDATIONS

This research stimulated many questions regarding the accuracy of underwater vehicle navigation. Some of these questions were answered by this thesis, more remain.

This work was based on establishing a GPS position as ground truth; however, there are inaccuracies within this system that would leave an absolute position error that should be dealt with.

The navigation filter during the course of the research was modified from an eight state filter to a seven state filter through zeroing the rate bias state. The code and filter should be redesigned to remove the zeroed state for the rate bias into a better defined seven state model.

This work focused on utilizing a single specific channel from the IMU, i.e. the yaw rate that was then integrated to provide a good heading reference. The IMU outputs three dimensional body linear accelerations and angular rates that could be used to further enhance the accuracy of the navigational estimates in both the horizontal and vertical planes. The method proposed based on current system architecture is to integrate the accelerations in the IMU code at 100 Hz and output the velocity components at 8 Hz intervals. These velocities could then be used in place of the velocities provided by the RDI Doppler in the EKF.

Adopting a two IMU system and removing the RDI Doppler as the primary speed sensor. Eliminating the RDI Doppler as an onboard sensor has the following benefits:

- Hotel load reduced, increasing mission endurance.
- Reduced electrical noise impacts on Forward Look Sonar.
- Reduced capital cost.
    - $35k (RDI Doppler) + $15k (IMU) vs.
    - $15k (Primary IMU) + $15k (Backup IMU)
- Eliminates water column constraints by eliminating need to see ocean bottom.
- Increases payload space for other components or reduces overall size of AUV.
- Enhanced mission flexibility due to above.

This work discovered that a pseudo-adaptive measurement noise scheme worked rather well in reducing the overall navigational error. While some work was performed

to adapt an algorithm by Busse et al., (2002) into the EKF, little success was made and the filter solutions diverged. Additional work ought to be performed to determine an adaptive process and measurement noise algorithm that will perform within the current architecture of this EKF application.

There is potential work that warrants attention in post-process smoothing of the vehicle track. In this manner by using more precise information from position measurements made through GPS the track that the vehicle actually ran between GPS pop-up maneuvers may be estimated. It can be seen from previous figures that the vehicle bases its control action from position estimates made through the navigation filter and that there are discontinuities in the solution from the (i-1) time step to the (i) time step at which a GPS measurement is made. By using a fixed-interval smoothing routine as proposed by Bar-Shalom (2001) the vehicle track may be post-processed for a more accurate representation of true vehicle position between GPS fixes.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: EXPERIMENT DATA

The mean errors and associated errors per unit distance traveled for runs analyzed during this research are summarized in tables 1 through 5. The tables are divided into each specific evolution of the vehicle for navigation improvements.

Date	NavD File	d File	Delta X	Delta Y	Error (m)	Error % Dist Traveled
6/10/2005	NavD_061005_01	d061005_01	-45.49	-65.53	79.77	116.29
			50.04	-60.47	78.49	18.33
Totals			4.55	-126.00	158.26	134.62
	Means		2.27	-63.00	79.13	67.31
	StDev		67.55	3.57	0.90	69.27
8/9/2005	NavD_080905_01	d080905_01	-1.15	28.35	28.37	54.94
			-2.63	49.49	49.56	81.63
			-12.39	116.11	116.77	62.15
Totals			-16.17	193.96	194.71	198.73
	Means		-5.39	64.65	64.90	66.24
	StDev		6.11	45.80	46.15	13.80

Table 1. Pre-IMU ARIES Data

Date	NavD File	d File	Delta X	Delta Y	Error (m)	Error % Dist Traveled
9/7/2005	NavD_090705_01	d090705_05	4.27	9.19	10.14	4.84
			-0.14	5.95	5.95	1.24
Totals			4.14	15.14	16.09	6.08
	Means		2.07	7.57	8.04	3.04
	StDev		3.12	2.29	2.96	2.54
5/12/2006	NavD_051206_02	d051206_02	3.85	-0.07	3.85	1.88
			-0.08	0.87	0.87	0.48
			-0.32	-1.44	1.47	0.67
			0.58	-4.73	4.77	2.34
			0.81	-6.24	6.30	3.17
			-1.85	-12.34	12.48	6.25
	NavD_051206_03	d051206_03	14.22	-55.17	56.97	29.13
			-25.92	49.32	55.71	27.52
			7.39	53.30	53.81	23.99
			7.89	-1.62	8.06	3.48
			1.94	-32.71	32.76	17.09
			-10.45	-25.79	27.82	14.33
			8.32	-31.07	32.16	17.26
	NavD_051206_03	d051206_04	0.95	48.04	48.05	8.32
Totals			7.33	-19.65	345.08	155.92
	Means		0.52	-1.40	24.65	11.14
	StDev		9.56	32.24	21.89	10.31
5/18/2006	NavD_051806_03	d051806_04	3.37	-1.32	3.62	0.71
			-6.00	-2.70	6.58	3.60
			-0.93	-0.47	1.04	0.42
			-6.59	10.30	12.22	7.15
Totals			-10.15	5.81	23.46	11.88
	Means		-2.54	1.45	5.87	2.97
	StDev		4.68	5.97	4.80	3.13

Table 2. Post-IMU Installation ARIES Data

Date	NavD File	d File	Delta X	Delta Y	Error (m)	Error % Dist Traveled
5/25/2006	NavD_052506_03	d052506_03	1.85	0.32	1.88	0.26
	NavD_052506_04	d052506_04	4.12	-2.47	4.80	0.61
Totals			5.97	-2.15	6.68	0.87
	Means		2.98	-1.08	3.34	0.43
	StDev		1.61	1.97	2.07	0.25
6/14/2006	NavD_061406_02	d061406_02	2.33	-8.76	9.07	1.74
			5.10	-3.18	6.01	1.36
	NavD_061406_04	d061406_04	10.30	25.36	27.37	7.88
			1.86	16.00	16.10	6.43
			2.00	17.89	18.01	8.54
			-0.11	12.32	12.32	5.95
			15.45	14.11	20.93	8.19
			-1.29	30.81	30.84	15.27
			-5.84	19.96	20.80	11.12
			-2.02	11.10	11.28	5.87
	NavD_061406_06	d061406_06	-1.05	45.52	45.53	6.82
			-0.02	4.17	4.17	1.92
			5.36	12.83	13.91	7.65
			1.85	8.62	8.81	4.81
			0.73	0.58	0.93	0.42
			-5.19	-10.23	11.47	5.98
			0.76	-17.55	17.57	8.24
Totals			30.22	179.54	275.10	108.19
	Means		1.78	10.56	16.18	6.36
	StDev		5.19	15.76	10.90	3.73
6/15/2006	NavD_061506_03	d061506_04	3.88	18.08	18.49	2.92
	NavD_061506_04	d061506_05	1.67	-37.58	37.62	6.21
Totals			5.55	-19.50	56.11	9.13
	Means		2.78	-9.75	28.05	4.57
	StDev		1.56	39.36	13.53	2.33

Table 3. ARIES Data with Gyro Rate Bias Set to Zero

Date	NavD File	d File	Delta X	Delta Y	Error (m)	Error % Dist Traveled
7/19/2006	NavD_071906_03	d071906_03	5.07	2.47	5.63	0.70
			-6.10	-5.58	8.27	1.24
	NavD_071906_06	d071906_07	3.81	1.29	4.02	0.70
			1.52	-3.56	3.87	0.74
Totals			4.29	-5.39	21.79	3.39
	Means		1.07	-1.35	5.45	0.85
	StDev		5.01	3.84	2.04	0.26
7/25/2006	NavD_072506_01	d072506_01	2.56	-3.36	4.22	0.42
	NavD_072506_02	d072506_02	5.16	-0.57	5.19	0.76
	NavD_072506_03	d072506_03	23.04	-7.62	24.27	1.13
Totals			30.75	-11.54	33.67	2.30
	Means		10.25	-3.85	11.22	0.77
	StDev		11.15	3.55	11.30	0.36

Table 4. ARIES Data with IMU Data Averaging Change

Date	NavD File	d File	Delta X	Delta Y	Error (m)	Error % Dist Traveled
8/24/2006	NavD_082406_05	d082406_05	0.71	-5.32	5.36	0.23
	NavD_082406_06a	d082406_06a	13.96	2.35	14.16	0.33
Totals			14.68	-2.96	19.52	0.56
	Means		7.34	-1.48	9.76	0.28
	StDev		9.37	5.42	6.22	0.07
9/12/2006	NavD_091206_01	d091206_02	3.18	9.61	10.13	0.45

Table 5. ARIES Data with Pseudo-Adaptive Algorithm

APPENDIX B: SIMULATION EKF CODE

The following MATLAB® code was used for running simulations on the data sets. The original code was developed by A. Healey with modifications made by H.S. Kim and S. Vonheeder during the course of this work. The codes contained in Appendices B and C are intended to run in MATLAB® with a specific data set structure from the ARIES vehicle. These data sets may be obtained from the NPS Center for AUV Research or the codes can be tailored to run with specific data sets to provide the correct state information.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        AUV Research Center                                %
%                        Extended Kalman Filter For ARIES AUV               %
%                        Edited: 11/9/06                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear all; clc

%% Load data from Desired Run with Aries using d and NavD Files
% generate default file name for dialog box
    defDateFields = datevec(date);
    defYear = sprintf('%02d',defDateFields(1) - 2000);
    defMon  = sprintf('%02d',defDateFields(2));
    defDay  = sprintf('%02d',defDateFields(3));
    defDateStamp = [defMon defDay defYear '_01.d'];
    defNameD = ['d' defDateStamp];

% get d-file name from dialog box
    fields = {'D File:'};
    boxTitle = 'Get Data File';
    lineNo = 1;
    defaultInput = {defNameD};
    input = inputdlg(fields, boxTitle, lineNo, defaultInput, 'on');
    if ( isempty(input) )
        disp('No data file selected. Exit program.');
        return;
    else
        dFileName = input{1};
    end;

% get NavD-file name from dialog box
    fields = {'NavD File:'};
    boxTitle = 'Get Nav Data';
    lineNo = 1;
    defNameNavD = ['NavD_' dFileName(2:10) '.d'];
    defaultInput = {defNameNavD};
    input = inputdlg(fields, boxTitle, lineNo, defaultInput, 'on');
    if ( isempty(input) )
```

```
        disp('No data file selected. Exit program.');
        return;
    else
        dFileNameNav = input{1};
    end;

d    =load(dFileName);
Nav  =load(dFileNameNav);

%% Data Assignment to variables
sss =Nav(1,2);
eee =length(Nav(:,2))+Nav(1,2)-1;
mmm =max(size(Nav));

startSample=1;
endSample=mmm;

d_khs(:,1)=Nav(:,21);         % gpsStatus check !-> ok)
d_khs(:,2)=Nav(:,32);         % IMU theta [rad]
d_khs(:,3)=Nav(:,31);         % IMU phi   [rad]
d_khs(:,4)=Nav(:,54);         % IMU psi   [rad]
d_khs(:,5)=Nav(:,8);          % yaw_rate  [rad/sec]
d_khs(:,6)=Nav(:,10);         % u [m/sec]
d_khs(:,7)=Nav(:,11);         % v [m/sec]
d_khs(:,8)=Nav(:,12);         % w [m/sec]
d_khs(:,9)=Nav(:,47);         % GPS longitude
d_khs(:,10)=Nav(:,46);        % GPS latitude

d_khs(:,11)=Nav(:,35);        % Nav_lat (X [m])
d_khs(:,12)=Nav(:,36);        % Nav_long (Y [m])
d_khs(:,13)=Nav(:,37);        % Nav_heading [rad]
d_khs(:,14)=Nav(:,38);        % Nav_yaw_rate [rad/sec]
d_khs(:,15)=Nav(:,39);        % Nav_u [m/sec]
d_khs(:,16)=Nav(:,40);        % Nav_v [m/sec]
d_khs(:,17)=Nav(:,42);        % Nav_Bias_r [rad/sec] (experimentally
% estimated Bias_psi)
d_khs(:,18)=Nav(:,41);        % Nav_Bias_psi [rad] (experimentally
% estimated Bias_r)

Bias_psi   =Nav(:,41);        % [rad] (experimentally estimated
Bias_psi)
Bias_r     =Nav(:,42);        % [rad/sec] (experimentally estimated
Bias_r)

gpsStatus =d_khs(:,1);
pitch     =d_khs(:,2);
roll      =d_khs(:,3);
heading   =d_khs(:,4);
yaw_rate  =d_khs(:,5);
ug        =d_khs(:,6);

vg_RDI    =d_khs(:,7);
vg        =d_khs(:,7)-1.0*d_khs(:,5);
```

```
wg        =d_khs(:,8);
long      =d_khs(:,9);
lat       =d_khs(:,10);
NsV       =Nav(:,23);

l1=long(startSample);
l2=lat(startSample);
long=long-l1*ones(length(lat),1);
lat =lat-l2*ones(length(lat),1);

for i=1:mmm,
    if (gpsStatus(i,1)==0)
        long(i,1)=0.0; lat(i,1)=0.0;
    end
end

NsV_3=find(NsV==3 & Nav(:,46)~=0);
    GPS_3(:,1)=Nav(NsV_3,46);
    GPS_3(:,2)=Nav(NsV_3,47);
NsV_4=find(NsV==4 & Nav(:,46)~=0);
    GPS_4(:,1)=Nav(NsV_4,46);
    GPS_4(:,2)=Nav(NsV_4,47);
NsV_5=find(NsV>=5 & Nav(:,46)~=0);
    GPS_5(:,1)=Nav(NsV_5,46);
    GPS_5(:,2)=Nav(NsV_5,47);

dt=1/8;
t=0:dt:(length(ug)-1)*dt;

nheading = zeros(1,length(heading));
nheading(1) = heading(1);

for i=2:length(heading)
  if abs(heading(i) - heading(i-1)) <= pi
     nheading(i) = nheading(i-1) + heading(i) - heading(i-1);
  end
  if heading(i) - heading(i-1) > pi
     nheading(i) = nheading(i-1) + heading(i) - heading(i-1) - 2*pi;
  end
  if heading(i-1) - heading(i) > pi
    nheading(i) = nheading(i-1) + heading(i) - heading(i-1) + 2*pi;
  end
end

heading = nheading';

% MEASUREMENT VECTOR
y=[ug,vg,heading,yaw_rate,lat,long]; % Complete measured data

% STATE VECTOR
x(:,s)=[lat(s),long(s),psi0,yaw_rate(s)*pi/180,ug,vg,br,bpsi]';
psi0=heading(startSample);
```

```
% Initialize the state vector, x is 8, y is 6
x=zeros(8,endSample); err=zeros(6,endSample);
s=startSample;
x(:,s)=[lat(s),long(s),psi0,yaw_rate(s),ug(s),vg(s),Bias_r(s),…
Bias_psi(s)]';

% Initial A matrix
    A=zeros(8,8);
    X=x(1,s);Y=x(2,s);psi=x(3,s);r=x(4,s); dop_ug=x(5,s);dop_vg=x(6,s);
    br=x(7,s);bpsi=x(8,s);
    A(1,3)=-dop_ug*sin(psi)-dop_vg*cos(psi);
    A(1,5)=cos(psi);
    A(1,6)=-sin(psi);
    A(2,3)=dop_ug*cos(psi)-dop_vg*sin(psi);
    A(2,5)=sin(psi);
    A(2,6)=cos(psi);
    A(3,4)=1;

if (gpsStatus(1,1)==1.0 & NsV(1)>3) % The NsV>3 rejects fixes that are
less than 4 satellites
    C=zeros(6,8);
    C(1,5)=1;
    C(2,6)=1;
    C(3,3)=1;C(3,8)=1;
    C(4,4)=1;C(4,7)=0;  % C(4,7) Forces B_r to 0
    C(5,1)=1;
    C(6,2)=1;
else
    C=zeros(6,8);
    C(1,5)=1;
    C(2,6)=1;
    C(3,3)=1;C(3,8)=1;
    C(4,4)=1;C(4,7)=0;
    C(5,1)=0;
    C(6,2)=0;
end

%% System Noise
q1=0.0;         %variance on X, m^2
q2=0.0;         %variance on Y, m^2
q3=0.001;       %variance on psi, rad^2
q4=0.1;         %variance on r, rad/s)^2
q5=0.01;        %variance on ug,(m/s)^2
q6=0.01;        %variance on vg,(m/s)^2
q7=0.0000001;   %
q8=0.0;         %
Q_dummy=[q1;q2;q3;q4;q5;q6;q7;q8];
Q=diag(Q_dummy);

    nu1    =0.001;
    nu2    =0.0001;
    nu3    =0.001;
    nu4    =0.001;
    nu5    =1.0;
    nu6    =1.0;
```

```
%% Initial CoVariance Matrix
% Note, old_after means measured data at old time, new_before means
% model predicted value

p_old_after=eye(8)*1e-2;
delx_old_after=zeros(8,1);
g=ones(8,1);
psave=zeros(8,startSample:endSample);

%% Main Calculation Loop
for i=startSample:endSample-1

        % Measurement Noise - Adjusts the values of the R-matrix based
        % on the quality of the fix or if no fix present.
        if NsV(i)<=3
            nu5=1.0;
            nu6=1.0;
        elseif NsV(i)==4
            nu5=0.1;
            nu6=0.1;
        elseif NsV(i)>=5
            nu5=0.01;
            nu6=0.01;
        end

R_dummy=[nu1;nu2;nu3;nu4;nu5;nu6];
R=diag(R_dummy);

% Compute linearized PHI matrix using updated A
% Phi=expm(A*dt);
    phi=eye(8,8)+A*dt+(A*dt)^2/2+(A*dt)^3/6;
    x_old_after=x(:,i);  %reset initial state
    %[x_new_before]=prop8(x_old_after,0,0,dt);  % nonlinear state
    % propagation
        xold=x_old_after;

%X=xold(1);Y=xold(2);psi=xold(3);r=xold(4);ug=xold(5);vg=xold(6);
        %bu=xold(7);bpsi=xold(8);
        f1=xold(5)*cos(xold(3))-xold(6)*sin(xold(3));
        f2=xold(5)*sin(xold(3))+xold(6)*cos(xold(3));
        f3=xold(4);
        f4=0; %rdot=0;
        f5=0; %ugdot=0;
        f6=0; %vgdot=0;
        f=[f1;f2;f3;f4;f5;f6;0;0];
        xnew=xold+f.*dt; %xd=f;
    x_new_before=xnew;
    p_new_before=phi*p_old_after*phi' + Q;     % error covariance
% propagation

% new gain calculation using linearized new C matrix and current state
% error covariances.
% formulate the innovation using nonlinear output propagation as
```

```
% compared to new sampled data from measurements

if (gpsStatus(i+1,1)==1.0 & NsV(i+1)>3)
    ny_sub=6;
    xold=x_new_before;

yhat=[xold(5);xold(6);xold(3)+xold(8);xold(4)+xold(7);xold(1);xold(2)];
else
    ny_sub=4;
    xold=x_new_before;
    yhat=[xold(5);xold(6);xold(3)+xold(8);xold(4)+xold(7);0;0];
end

err(:,i+1)=(y(i+1,1:6)' - yhat); % Innovation

if (gpsStatus(i+1,1)==1.0 & NsV(i+1,1)>3)
    C=zeros(6,8);
    C(1,5)=1;
    C(2,6)=1;
    C(3,3)=1;C(3,8)=1;
    C(4,4)=1;C(4,7)=0;
    C(5,1)=1;
    C(6,2)=1;
else
    C=zeros(6,8);
    C(1,5)=1;
    C(2,6)=1;
    C(3,3)=1;C(3,8)=1;
    C(4,4)=1;C(4,7)=0;
    C(5,1)=0;
    C(6,2)=0;
end

if ny_sub==6,
    if
((y(i+1,1)==y(i,1))||(y(i+1,1)==0.0)||(abs(y(i+1,1))>=5.0)),C(1,:)=…
0.0*g';end;
    if
((y(i+1,2)==y(i,2))||(y(i+1,2)==0.0)||(abs(y(i+1,2))>=5.0)),C(2,:)=…
0.0*g';end;
    if (abs(y(i+1,3)-y(i,3))<=0.000001),C(3,:)=0.0*g';end;
    if (abs(y(i+1,4)-y(i,4))<=0.000001),C(4,:)=0.0*g';end;
    if (abs(y(i+1,5)-y(i,5))<=0.000001),C(5,:)=0.0*g';end;
    if (abs(y(i+1,6)-y(i,6))<=0.000001),C(6,:)=0.0*g';end;
else
    if
((y(i+1,1)==y(i,1))||(y(i+1,1)==0.0)||(abs(y(i+1,1))>=5.0)),C(1,:)=…
0.0*g';end;
    if
((y(i+1,2)==y(i,2))||(y(i+1,2)==0.0)||(abs(y(i+1,2))>=5.0)),C(2,:)=…
0.0*g';end;
    if (abs(y(i+1,3)-y(i,3))<=0.000001),C(3,:)=0.0*g';end;
    if (abs(y(i+1,4)-y(i,4))<=0.000001),C(4,:)=0.0*g';end;
end
```

```
% Compute gain, update Total State and error covariance
G=p_new_before*C(1:ny_sub,:)'*inv(C(1:ny_sub,:)*p_new_before*…
C(1:ny_sub,:)' + R(1:ny_sub,1:ny_sub)); % Kalman Gain

p_temp=G*C(1:ny_sub,:)*p_new_before;
p_old_after=p_new_before-p_temp;

psave(:,i+1)=diag(p_old_after);
x_new_after=x_new_before + G*err(1:ny_sub,i+1);

%i
%endSample

%carry new state into next update
x(:,i+1)=x_new_after;

%resetting up the linearized A matrix
A=zeros(8,8);
X=x(1,i+1);Y=x(2,i+1);psi=x(3,i+1);r=x(4,i+1);
dop_ug=x(5,i+1);dop_vg=x(6,i+1);bu=x(7,i+1);bpsi=x(8,i+1);

A(1,3)=-dop_ug*sin(psi)-dop_vg*cos(psi);
A(1,5)=cos(psi);
A(1,6)=-sin(psi);
A(2,3)=dop_ug*cos(psi)-dop_vg*sin(psi);
A(2,5)=sin(psi);
A(2,6)=cos(psi);
A(3,4)=1;

end;

%% Plot Results in Figures
% figure(1)
%
plot(long(startSample:endSample),lat(startSample:endSample),'r+'),grid
%     hold on
%     plot(x(2,startSample:endSample),x(1,startSample:endSample),'g.')
%     hold off
%     title('Filter Estimated Path(green), GPS (red)')
%     ylabel('latitude in meters')
%     xlabel('longitude in meters')
%     grid
%     axis equal

figure(2)  % Path Plot
    %plot(long,lat,'r+');
    %hold on
    plot(x(2,:),x(1,:),'g-');
    hold on
    plot(d_khs(:,12)-d_khs(s,12),d_khs(:,11)-d_khs(s,11),'b-');
    plot(GPS_3(:,2)-d_khs(s,12),GPS_3(:,1)-d_khs(s,11),'*k',...
         GPS_4(:,2)-d_khs(s,12),GPS_4(:,1)-d_khs(s,11),'*m',...
```

```
         GPS_5(:,2)-d_khs(s,12),GPS_5(:,1)-d_khs(s,11),'*r');
    hold off
    grid
    title('POSITION: GPS, Filter Estimated, In-Vehicle Estimated');
    legend('Filter','Vehicle Filter','GPS NsV=3','GPS NsV=4',…
'GPS NsV=5');
    ylabel('X [m]')
    xlabel('Y [m]')
    axis equal

% figure(3) % X Dimension
%     plot(t(startSample:endSample),lat(startSample:endSample),'r+',...
%          t(startSample:endSample),x(1,startSample:endSample),'g.',...
%          t(startSample:endSample),d_khs(startSample:endSample,11)-
%            d_khs(s,11),'b.');
%     title('X: measured [red], sim estimated [green], exp estimated
%           [blue]');
%
% figure(4) % Y Dimension
%
% plot(t(startSample:endSample),long(startSample:endSample),'r+',...
%          t(startSample:endSample),x(2,startSample:endSample),'g.',...
%          t(startSample:endSample),d_khs(startSample:endSample,12)-
% d_khs(s,12),'b.');
%     title('Y: measured [red], sim estimated [green], exp estimated
% [blue]');

figure(5) % Heading Results

plot(t(startSample:endSample),heading(startSample:endSample),'r+',...
         t(startSample:endSample),(x(3,startSample:endSample)),'g.',...

t(startSample:endSample),(d_khs(startSample:endSample,13)),'b.');
    title('Heading (radians): measured [red], sim estimated [green],…
exp estimated [blue]');

figure(6) % Yaw Rate

plot(t(startSample:endSample),yaw_rate(startSample:endSample),'r+',...
         t(startSample:endSample),x(4,startSample:endSample),'g.',...

t(startSample:endSample),d_khs(startSample:endSample,14),'b.');
    title('Yaw_rate: measured [red], sim estimated [green], exp…
estimated [blue]');

figure(7) % Speed, ug
    plot(t(startSample:endSample),ug(startSample:endSample),'r+',...
         t(startSample:endSample),x(5,startSample:endSample),'g.',...

t(startSample:endSample),d_khs(startSample:endSample,15),'b.');
    title('ug: measured [red], sim estimated [green], exp estimated…
[blue]');

figure(8) % Speed, vg
```

```
plot(t(startSample:endSample),vg_RDI(startSample:endSample),'r+',...
         t(startSample:endSample),x(6,startSample:endSample),'g.',...

t(startSample:endSample),d_khs(startSample:endSample,16),'b.');
    title('vg: measured [red], sim estimated [green], exp estimated…
[blue]');

figure(9) % Bias_r
    plot(t(startSample:endSample),x(7,startSample:endSample),'g.',...
         t(startSample:endSample),Bias_r(startSample:endSample,1),'b.')
    grid
    title('Bias_r: sim estimated [green], exp estimated [blue]');
    xlabel('time in seconds')
    ylabel('[deg/sec]')

figure(10) % Bias_psi
    plot(t(startSample:endSample),x(8,startSample:endSample),'g.',...

t(startSample:endSample),Bias_psi(startSample:endSample,1),'b.')
    grid
    title('Bias_\psi: sim estimated [green], exp estimated [blue]');
    xlabel('time in seconds')
    ylabel('B_\psi (rad)')
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: SIMULATION UKF CODE

The following MATLAB code was used for running simulations on the data sets to evaluate UKF performance with EKF performance. The original code was developed by H.S. Kim based on Julier and Uhlmann's algorithm (1997) with modifications made by S. Vonheeder during the course of this work.

```
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       AUV Research Center                          %
%                       Unscented Kalman Filter For ARIES AUV        %
%                       Edited: 11/09/06                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all; clear all; clc

%% Load data from Desired Run with Aries using d and NavD Files
% generate default file name for dialog box
    defDateFields = datevec(date);
    defYear = sprintf('%02d',defDateFields(1) - 2000);
    defMon  = sprintf('%02d',defDateFields(2));
    defDay  = sprintf('%02d',defDateFields(3));
    defDateStamp = [defMon defDay defYear '_01.d'];
    defNameD = ['d' defDateStamp];

% get d-file name from dialog box
    fields = {'D File:'};
    boxTitle = 'Get Data File';
    lineNo = 1;
    defaultInput = {defNameD};
    input = inputdlg(fields, boxTitle, lineNo, defaultInput, 'on');
    if ( isempty(input) )
        disp('No data file selected. Exit program.');
        return;
    else
        dFileName = input{1};
    end;

% get NavD-file name from dialog box
    fields = {'NavD File:'};
    boxTitle = 'Get Nav Data';
    lineNo = 1;
    defNameNavD = ['NavD_' dFileName(2:10) '.d'];
    defaultInput = {defNameNavD};
    input = inputdlg(fields, boxTitle, lineNo, defaultInput, 'on');
    if ( isempty(input) )
        disp('No data file selected. Exit program.');
        return;
    else
        dFileNameNav = input{1};
    end;
```

```
d    =load(dFileName);
Nav  =load(dFileNameNav);

%% ================================================================
% Data Manipulation from the experiment NAV data
% ================================================================
mmm=max(size(Nav));
startSample=1;
endSample=mmm;

d_khs(:,1) =Nav(:,21);         % gpsStatus check !-> ok)
d_khs(:,2) =Nav(:,32);         % IMU theta [rad]
d_khs(:,3) =Nav(:,31);         % IMU phi   [rad]
d_khs(:,4) =Nav(:,54);         % IMU psi   [rad]
d_khs(:,5) =Nav(:,8);          % yaw_rate  [rad/sec]
d_khs(:,6) =Nav(:,10);         % u [m/sec]
d_khs(:,7) =Nav(:,11);         % v [m/sec]
d_khs(:,8) =Nav(:,12);         % w [m/sec]
d_khs(:,9) =Nav(:,47);         % GPS longitude
d_khs(:,10)=Nav(:,46);         % GPS latitude

d_khs(:,11)=Nav(:,35);        % Nav_lat (X [m])
d_khs(:,12)=Nav(:,36);        % Nav_long (Y [m])
d_khs(:,13)=Nav(:,37);        % Nav_heading [rad]
d_khs(:,14)=Nav(:,38);        % Nav_yaw_rate [rad/sec]
d_khs(:,15)=Nav(:,39);        % Nav_u [m/sec]
d_khs(:,16)=Nav(:,40);        % Nav_v [m/sec]
d_khs(:,17)=Nav(:,42);        % Nav_Bias_r [rad/sec](exp est Bias_psi)
d_khs(:,18)=Nav(:,41);        % Nav_Bias_psi [rad] (exp ested Bias_r)

Bias_psi   =Nav(:,41);        % [rad] (exp est Bias_psi)
Bias_r     =Nav(:,42);        % [rad/sec] (exp est Bias_r)

%%=================================================================
% Initialize the experimental results
% ================================================================
gpsStatus   =d_khs(:,1);
pitch       =d_khs(:,2);
roll        =d_khs(:,3);
heading     =d_khs(:,4);
yaw_rate    =d_khs(:,5);
ug          =d_khs(:,6);
vg          =d_khs(:,7)-1.0*d_khs(:,5);
wg          =d_khs(:,8);

long        =d_khs(:,9);
lat         =d_khs(:,10);
l1=long(startSample);
l2=lat(startSample);
long=long-l1.*ones(length(lat),1);
lat=lat-l2*ones(length(lat),1);

for i=1:mmm,
    if (gpsStatus(i,1)==0)
        long(i,1)=0.0;lat(i,1)=0.0;
```

```
    else
    end
end

% ================================================================
% Time Vector
% ================================================================
dt=1/8;
t=0:dt:(length(ug)-1)*dt;

% ================================================================
% Heading Signal Wrapping
% ================================================================
nheading = zeros(1, length(heading));
nheading(1) = heading(1);

for i=2:length(heading)
  if abs(heading(i) - heading(i-1)) <= pi
     nheading(i) = nheading(i-1) + heading(i) - heading(i-1);
  end
  if heading(i) - heading(i-1) > pi
     nheading(i) = nheading(i-1) + heading(i) - heading(i-1) - 2*pi;
  end
  if heading(i-1) - heading(i) > pi
    nheading(i) = nheading(i-1) + heading(i) - heading(i-1) + 2*pi;
  end
end

heading = nheading';

%=================================================================
% MEASUREMENT VECTOR
%=================================================================
y=[ug,vg,heading,yaw_rate,lat,long]; %complete measured data

%% ===============================================================
% Initialize the UKF parameters and state  vector, x is 8, y is 6
%=================================================================
nx=8;
ny=6;
nv=8;
nn=6;
na=nx;

scale=3;
kappa=scale-na;
W0_m=kappa/(na+kappa);
Wi_m=1/(2*(na+kappa));
W0_c=kappa/(na+kappa);
Wi_c=1/(2*(na+kappa));

s=startSample;
x=zeros(8,endSample); err=zeros(6,endSample);
psi0=heading(s);
x(:,s)=[lat(s),long(s),psi0,yaw_rate(s),ug(s),vg(s),...
```

```
        Bias_r(s),Bias_psi(s)]';

xa(:,s)=[x(:,s)];
xa_old=xa(:,s);
x_old=x(:,s);

%% System Noise
q1=0.01;        %variance on X, m^2  Cannot be 0, Q pos semi-def
q2=0.01;        %variance on Y, m^2  for Cholesky Decomposition
q3=0.001;       %variance on psi, rad^2
q4=0.1;         %variance on r, (rad/s)^2
q5=0.01;        %variance on ug,(m/s)^2
q6=0.01;        %variance on vg,(m/s)^2
q7=0.0000001;   %variance on B_r
q8=0.0          %variance on B_psi
Q_dummy=[q1;q2;q3;q4;q5;q6;q7;q8];
Q=diag(Q_dummy);

% Measurement Noise
nu1=0.1;
nu2=0.1;
nu3=0.001;
nu4=0.001;
nu5=1.0;
nu6=1.0;
R_dummy=[nu1;nu2;nu3;nu4;nu5;nu6];
R=diag(R_dummy);

%% ================================================================
% Initial Pa matrix
%=================================================================
p_old_after=eye(nx)*1e-2;

delx_old_after=zeros(nx,1);
g=ones(nx,1);
psave=zeros(nx,startSample:endSample);

Pa_old=p_old_after;

%% Main Calculation Loop
for i=startSample:endSample-1
    %i
    %endSample
    %=================================================================
    % 1. Calculate Sigma points
    %=================================================================
    bias_Pa=chol((na+kappa)*Pa_old);

    %=================================================================
    Sigma_points_old(:,1)=xa_old;

    for j=1:na,
        Sigma_points_old(:,j+1)=xa_old+bias_Pa(j,:)';
        Sigma_points_old(:,na+j+1)=xa_old-bias_Pa(j,:)';
    end
```

```
%=================================================================
% Time Update
%=================================================================
% 2. Nonlinear state propagation
%-----------------------------------------------------------------
for j=1:(2*na+1)
    % Sigma_points_new_before(1:nx,j)=prop_ukf...
    %   (Sigma_points_old(1:nx,j),(process_noise*0.0),dt);
    xold=Sigma_points_old(1:nx,j);
    f1=xold(5)*cos(xold(3))-xold(6)*sin(xold(3));
    f2=xold(5)*sin(xold(3))+xold(6)*cos(xold(3));
    f3=xold(4);
    f4=0; %rdot=0;
    f5=0; %ugdot=0;
    f6=0; %vgdot=0;
    f7=0; %Bias_r_dot=0;
    f8=0; %Bias_psi_dot=0;
    f=[f1;f2;f3;f4;f5;f6;f7;f8];
    xnew=xold+f.*dt;
    Sigma_points_new_before(1:nx,j)=xnew;
end
%-----------------------------------------------------------------
% 3. mean value of x_new_before
%-----------------------------------------------------------------
mean_x_new_before=zeros(nx,1);
for j=1:(2*na+1)
    if j==1
        mean_x_new_before=mean_x_new_before+W0_m*...
                            Sigma_points_new_before(1:nx,j);
    else
        mean_x_new_before=mean_x_new_before+Wi_m*...
                            Sigma_points_new_before(1:nx,j);
    end
    mean_x_new_before(7)=0;  % B_r=0
end
%-----------------------------------------------------------------
% 4. error covariance propagation
%-----------------------------------------------------------------
P_new_before=Q;
for j=1:(2*na+1),
    if j==1
        P_new_before=P_new_before+W0_c*...
           (Sigma_points_new_before(1:nx,j)-mean_x_new_before)*...
           (Sigma_points_new_before(1:nx,j)-mean_x_new_before)';
    else
        P_new_before=P_new_before+Wi_c*...
           (Sigma_points_new_before(1:nx,j)-mean_x_new_before)*...
           (Sigma_points_new_before(1:nx,j)-mean_x_new_before)';
    end
end
%-----------------------------------------------------------------
% 5. output Sigma_yhat
%-----------------------------------------------------------------
for j=1:(2*na+1),
```

```
    if gpsStatus(i+1)==1
        ny_sub=ny;
        % Sigma_yhat_before(1:ny,j)=output_ukf_w_GPS...
        %(Sigma_points_new_before(1:nx,j),(measurement_noise*0.0));
        xold(1:nx)=Sigma_points_new_before(1:nx,j);
        y1=xold(5);
        y2=xold(6);
        y3=xold(3)+xold(8);
        y4=xold(4)+xold(7)*0;
        y5=xold(1);
        y6=xold(2);
        yhat=[y1;y2;y3;y4;y5;y6];
        Sigma_yhat_before(1:ny,j)=yhat;
    else
        ny_sub=ny-2;
        % Sigma_yhat_before(1:ny,j)=output_ukf_wo_GPS...
        %(Sigma_points_new_before(1:nx,j),(measurement_noise*0.0));
        xold(1:nx)=Sigma_points_new_before(1:nx,j);
        y1=xold(5);
        y2=xold(6);
        y3=xold(3)+xold(8);
        y4=xold(4)+xold(7)*0;
        y5=xold(1);
        y6=xold(2);
        yhat=[y1;y2;y3;y4;0;0];
        Sigma_yhat_before(1:ny,j)=yhat;
    end
end

%-------------------------------------------------------------------
% 6. mean value of yhat
%-------------------------------------------------------------------
mean_yhat=zeros(ny,1);
for j=1:(2*na+1),
    if j==1
        mean_yhat=mean_yhat+W0_m*Sigma_yhat_before(1:ny,j);
    else
        mean_yhat=mean_yhat+Wi_m*Sigma_yhat_before(1:ny,j);
    end
end

%===================================================================
% Measurement Update
%===================================================================
%-------------------------------------------------------------------
% 1. Pyy_bar and Pxy
%-------------------------------------------------------------------
Pyy_bar=zeros(ny_sub,ny_sub);
Pxy=zeros(nx,ny_sub);

for j=1:(2*na+1),
    err_x=Sigma_points_new_before(1:nx,j)-mean_x_new_before;
    err_y=Sigma_yhat_before(1:ny_sub,j)-mean_yhat(1:ny_sub,1);
    if j==1
        Pxy=Pxy+W0_c*err_x*err_y';
```

```
        Pyy_bar=Pyy_bar+W0_c*err_y*err_y';
    else
        Pxy=Pxy+Wi_c*err_x*err_y';
        Pyy_bar=Pyy_bar+Wi_c*err_y*err_y';
    end
end

S_temp=Pyy_bar-R(1:ny_sub,1:ny_sub);

if gpsStatus(i+1)==1
    if ((y(i+1,1)==y(i,1))||(y(i+1,1)==0.0)||...
                   (abs(y(i+1,1))>=5.0)),Pxy(:,1)=0.0*g;
        S_temp(:,1)=0.0*g(1:ny_sub,1);
        S_temp(1,:)=0.0*g(1:ny_sub,1)';
    end
    if ((y(i+1,2)==y(i,2))||(y(i+1,2)==0.0)||...
                   (abs(y(i+1,2))>=5.0)),Pxy(:,2)=0.0*g;
        S_temp(:,2)=0.0*g(1:ny_sub,1);
        S_temp(2,:)=0.0*g(1:ny_sub,1)';
    end
    if (abs(y(i+1,3)-y(i,3))<=0.000001),Pxy(:,3)=0.0*g;
        S_temp(:,3)=0.0*g(1:ny_sub,1);
        S_temp(3,:)=0.0*g(1:ny_sub,1)';
    end
    if (abs(y(i+1,4)-y(i,4))<=0.000001),Pxy(:,4)=0.0*g;
        S_temp(:,4)=0.0*g(1:ny_sub,1);
        S_temp(4,:)=0.0*g(1:ny_sub,1)';
    end
    if (abs(y(i+1,5)-y(i,5))<=0.000001),Pxy(:,5)=0.0*g;
        S_temp(:,5)=0.0*g(1:ny_sub,1);
        S_temp(5,:)=0.0*g(1:ny_sub,1)';
    end
    if (abs(y(i+1,6)-y(i,6))<=0.000001),Pxy(:,6)=0.0*g;
        S_temp(:,6)=0.0*g(1:ny_sub,1);
        S_temp(6,:)=0.0*g(1:ny_sub,1)';
    end
else
    if ((y(i+1,1)==y(i,1))||(y(i+1,1)==0.0)||...
                   (abs(y(i+1,1))>=5.0)),Pxy(:,1)=0.0*g;
        S_temp(:,1)=0.0*g(1:ny_sub,1);
        S_temp(1,:)=0.0*g(1:ny_sub,1)';
    end;
    if ((y(i+1,2)==y(i,2))||(y(i+1,2)==0.0)||...
                   (abs(y(i+1,2))>=5.0)),Pxy(:,2)=0.0*g;
        S_temp(:,2)=0.0*g(1:ny_sub,1);
        S_temp(2,:)=0.0*g(1:ny_sub,1)';
    end;
    if (abs(y(i+1,3)-y(i,3))<=0.000001),Pxy(:,3)=0.0*g;
        S_temp(:,3)=0.0*g(1:ny_sub,1);
        S_temp(3,:)=0.0*g(1:ny_sub,1)';
    end;
    if (abs(y(i+1,4)-y(i,4))<=0.000001),Pxy(:,4)=0.0*g;
        S_temp(:,4)=0.0*g(1:ny_sub,1);
        S_temp(4,:)=0.0*g(1:ny_sub,1)';
    end;
```

```
    end

    %-----------------------------------------------------------------
    % 2. UKF Gain matrix
    %-----------------------------------------------------------------
    UKF_K=Pxy*inv(S_temp+R(1:ny_sub,1:ny_sub));
    %-----------------------------------------------------------------
    % 3. x_new
    % 3. x_new
    %-----------------------------------------------------------------
    err(:,i+1)=(y(i+1,:)'-mean_yhat);
    %-----------------------------------------------------------------
        if sqrt(err(5,i+1)^2+err(6,i+1)^2)>100,
           err(5,i+1)=0;err(6,i+1)=0;
        end;
    %-----------------------------------------------------------------
    x_new=mean_x_new_before+UKF_K*err(1:ny_sub,i+1);
    x_new(7)=0;
    %-----------------------------------------------------------------
    % 4. P_new
    %-----------------------------------------------------------------
    P_new=P_new_before-UKF_K*(Pyy_bar)*UKF_K';
    %-----------------------------------------------------------------
    % 5. xa_old & Pa_old
    %-----------------------------------------------------------------
    xa_old=x_new;
    Pa_old=P_new;
    psave(:,i+1)=diag(P_new);
    x(:,i+1)=x_new;
end

%% Plot Results in Figures
figure(1),clf
    plot(long(startSample:endSample),lat(startSample:endSample),'r.')
    grid
    hold on
    plot(x(2,startSample:endSample),x(1,startSample:endSample),'g-')
    hold off
    title('Filter Estimated Path(green),GPS (red)')
    ylabel('latitude in meters')
    xlabel('longitude in meters')
    grid
    axis equal

figure(2), clf  % Path Plot
    plot(long,lat,'r.');
    hold on
    plot(x(2,:),x(1,:),'g-');
    plot(d_khs(:,12)-d_khs(s,12),d_khs(:,11)-d_khs(s,11),'b-');
    hold off
    grid
    legend('GPS Data', 'UKF Estimate', 'EKF Estimate');
    title(strcat(dFileName(1:7),'\_',dFileName(9:12),' Path Plot'));
    % title('POSTION: GPS [red], UKF Estimated [green] , In-Vehicle
    % EKF Estimated [blue]');
```

```
    ylabel('X [m]')
    xlabel('Y [m]')
    axis equal

figure(3), clf % X Dimension
    plot(t(startSample:endSample),lat(startSample:endSample),'r.',...
         t(startSample:endSample),x(1,startSample:endSample),'g-',...
         t(startSample:endSample),d_khs(startSample:endSample,11)…
           -d_khs(s,11),'b-');
    title('X: measured [red], sim estimated [green],…
          exp estimated [blue]');

figure(4), clf % Y Dimension
    plot(t(startSample:endSample),long(startSample:endSample),'r.',...
         t(startSample:endSample),x(2,startSample:endSample),'g-',...
         t(startSample:endSample),d_khs(startSample:endSample,12)…
           -d_khs(s,12),'b-');
    title('Y: measured [red], sim estimated [green],…
           exp estimated [blue]');

figure(5), clf % Heading Results

plot(t(startSample:endSample),heading(startSample:endSample),'r.',...
         t(startSample:endSample),(x(3,startSample:endSample)),'g-',...
         t(startSample:endSample),(d_khs(startSample:endSample,13)),'b-
');
    title('Heading (radians): measured [red], sim estimated [green],
exp estimated [blue]');

figure(6), clf % Yaw Rate

plot(t(startSample:endSample),yaw_rate(startSample:endSample),'r.',...
         t(startSample:endSample),x(4,startSample:endSample),'g-',...
         t(startSample:endSample),d_khs(startSample:endSample,14),…
           'b-');
    title('Yaw rate: measured [red], sim estimated [green],…
           exp estimated [blue]');

figure(7), clf % Speed, ug
    plot(t(startSample:endSample),ug(startSample:endSample),'r.',...
         t(startSample:endSample),x(5,startSample:endSample),'g.',...

t(startSample:endSample),d_khs(startSample:endSample,15),'b.');
    title('ug: measured [red], sim estimated [green],…
           exp estimated [blue]');

figure(8), clf % Speed, vg
    plot(t(startSample:endSample),vg(startSample:endSample),'r.',...
        t(startSample:endSample),x(6,startSample:endSample),'g.',...
        t(startSample:endSample),d_khs(startSample:endSample,16),'b.');
    title('vg: measured [red], sim estimated [green],…
           exp estimated [blue]');

figure(9), clf % Bias_r
    plot(t(startSample:endSample),x(7,startSample:endSample),'g.',...
```

```
          t(startSample:endSample),Bias_r(startSample:endSample,1),'b.')
    grid
    title('Bias_r: sim estimated [green], exp estimated [blue]');
    xlabel('time in seconds')
    ylabel('[deg/sec]')

figure(10), clf % Bias_psi
    plot(t(startSample:endSample),x(8,startSample:endSample),'g.',...

t(startSample:endSample),Bias_psi(startSample:endSample,1),'b.')
    grid
    title('Bias_\psi: sim estimated [green], exp estimated [blue]');
    xlabel('time in seconds')
    ylabel('B_\psi (rad)')
```

APPENDIX D: VEHICLE IMU CODE

The following provides the IMU code, written in C, that processes the IMU data output and performs the necessary reference frame rotations as well as the integration of the yaw rate for the heading reference. The accelerations, angular rates and heading are sent to the navigation filter for further processing. The original code was developed by J. Nicholson, CAPT, USN and modified by S. Kragelund for this research.

```
/*-----------------------------------------------------------------------
** Modifications to Jack Nicholson's IMU.c for realtime (8Hz) operation
** This program downsamples the 100Hz IMU data, processing one message
** each 8Hz interval. Checksum verification has been added to ensure
** only valid data gets passed to the Nav process
**
** SPK/DTD 07/14/06
-----------------------------------------------------------------------*/
#define TRUE  1
#define FALSE 0

#include "IMU.h"

// SPK 010306: Need this to access migration library
#include <mig4nto.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>

// SPK 011106: Added for Neutrino timer pulses
#define  MP_PULSE_CODE  _PULSE_CODE_MINAVAIL

#define IMU_MSG_SIZE 44
#define PITCH_LIMIT (60.0)  // degrees
#define ROLL_LIMIT  (60.0)  // degrees

// SPK 070506: Define values for IMU status word #1 bits
#define IMU_ACCEL_TEMP     0xFF00 // Bits 15-8 (LSB = 1 deg C)
#define IMU_RLG_A_PLC      0x0080 // Bit 7 (a-axis RLG in PLC reset)
#define IMU_RLG_B_PLC      0x0040 // Bit 6 (b-axis RLG in PLC reset)
#define IMU_RLG_C_PLC      0x0020 // Bit 5 (c-axis RLG in PLC reset)
#define IMU_FAILURE        0x0010 // Bit 4
#define IMU_SW1_ERR_BITS   0x00F0 // Bits 7-4
#define IMU_COUNTER        0X000F // Bits 3-0 (4-bit counter)

// SPK 070506: Define values for IMU status word #2 bits
#define IMU_PROC_FAILURE   0x8000 // Bit 15 (Processor tests failed)
#define IMU_MEM_FAILURE    0x4000 // Bit 14 (Memory tests failed)
#define IMU_OTHER_FAILURE  0x2000 // Bit 13 (Other tests failed)
```

```
#define IMU_ACCEL_FAILURE  0x1000 // Bit 12 (Accelerometer tests
failed)
#define IMU_GYRO_FAILURE   0x0800 // Bit 11 (Gyro tests failed)
#define IMU_RSV1_FAILURE   0x0400 // Bit 10 (Reserved)
#define IMU_RSV2_FAILURE   0x0200 // Bit  9 (Reserved)
#define IMU_RSV3_FAILURE   0x0100 // Bit  8 (Reserved)
#define IMU_SW2_ERR_BITS   0xF800 // Bits 15-11
#define IMU_SW_VER_NUMBER  0x00FF // Bits 7-0 (Software Version #)

// SPK 070606: Define value to identify a checksum error
#define IMU_CHKSUM_ERROR   0x0001

FILE *IMUalignfp;
FILE *NavLatfp;
FILE *Outfp;         // Use during testing
FILE *DateStampInfp; // Use during testing

double pi=3.14159265;

// This function retrieves the last IMU message in the buffer at time,
// verifies its checksum, and returns an integer offset for the first
// byte of the message
int getLastMessage(buffer,numBytes)
   u_char* buffer;
   int numBytes;
{
   int i;
   int found = FALSE;
   unsigned short chkSum;
   unsigned short computedChkSum;

   if (numBytes < IMU_MSG_SIZE)
      return -1;

   for (i = (numBytes - IMU_MSG_SIZE); ((i >= 0) && (!found)); i--)
   {
      if ((*(buffer+i) == 0xA5) && (*(buffer+i+1) == 0x02))
      {
         chkSum         = *(unsigned short*)(buffer + i + IMU_MSG_SIZE
- 2);
         computedChkSum = *(unsigned short*)(buffer + i + 2) +
                          *(unsigned short*)(buffer + i + 4) +
                          *(unsigned short*)(buffer + i + 6) +
                          *(unsigned short*)(buffer + i + 8) +
                          *(unsigned short*)(buffer + i + 10) +
                          *(unsigned short*)(buffer + i + 12) +
                          *(unsigned short*)(buffer + i + 14) +
                          *(unsigned short*)(buffer + i + 16) +
                          *(unsigned short*)(buffer + i + 18) +
                          *(unsigned short*)(buffer + i + 20) +
                          *(unsigned short*)(buffer + i + 22) +
                          *(unsigned short*)(buffer + i + 24) +
                          *(unsigned short*)(buffer + i + 26) +
                          *(unsigned short*)(buffer + i + 28) +
```

```
                        *(unsigned short*)(buffer + i + 30) +
                        *(unsigned short*)(buffer + i + 32) +
                        *(unsigned short*)(buffer + i + 34) +
                        *(unsigned short*)(buffer + i + 36) +
                        *(unsigned short*)(buffer + i + 38) +
                        *(unsigned short*)(buffer + i + 40);
         if (chkSum == computedChkSum)
         {
            found = TRUE;
            i++;
         }
         else
         {
            printf("IMU.c: getLastMessage: Bad Checksum!\n");
         }
      } // if ((*buffer+1...
   } // for (i...
   //printf("Index %d\n", i);
   if ( i < 0 )
   {
       printf("IMU.c: getLastMessage: No Valid Record Found!\n");
   }
   return i;
} // getLastMessage


main()
{
   int IMU_Shmid,IMUFlag_Shmid, HMR_Shmid,read_status;
   int i,k,i_r,i_m,Msg,StartByte;
   int Port, BytesRead;

   double g = 32.2;

   // SPK 010306: changed from char to unsigned char arrays to prevent
   //                     compiler warnings due to comparisons with 165
//value
   //                     (greater than maximum value for signed chars)
   u_char ReadBuf[2000];      // Buffer for data coming in from serial
//port
   u_char MsgBuf[44];         // Message buffer, for aligning and
//parsing data
   char FileCommand[256];     // For reading latitude...
   char FileString[256];      // ...from Nav.inp
   short p, q, r, ax, ay, az; // Flt control angular rates and accel
   int dAx, dAy, dAz, dVx, dVy, dVz;         // Inertial delta angles /
//velocities

   // SPK 070506: Added to verify IMU message checksums
   unsigned short int msgCkSum = 0;
   unsigned short int status1 = 0;
   unsigned short int status2 = 0;
   int LastMsgIdx = -1;
```

```
   // SPK 062606: added variables for first-order accelerometer filter
   double Tau,Coef1,Coef2,phiThresh,thetaThresh;
   int    initFilter = FALSE;
   int    filterAcc  = FALSE; // set to TRUE for first-order filtering
of accelerometer data
   double p_out, q_out, r_out, ax_out, ay_out, az_out;              //
Variables written...
   double dAx_out,dAy_out,dAz_out,dPsi_out;                         //
...to shared memory.
   double theta_out, phi_out;                                       //
2 Euler angles
   double dt;                                                       //
sampling interval
   double Lat0, LatD, Lat;                                          //
GPS Origin latitude
   double s_phi,c_phi,s_theta,c_theta,s_psi,c_psi,s_Lat,c_Lat;      //
Sines and cosines
   double q_e,r_e;                                                  //
Earth rates in q,r
   double psi_out, psi_dot, dPsi;                                   //
Heading

   // SPK 062606: Default behavior will write to shared memory AND file
unless FILE_FLAG = 0
   int FILE_FLAG=FALSE; // <<-- Set FILE_FLAG to zero to stop writing
100 Hz data to a file
   int INIT_ERROR = 0;
   int IMU_Error=0;                      // Flag to abort mission upon
IMU failure
   unsigned char IMU_msgError = 1;       // Flag if individual message
contains an error
   unsigned short int IMU_msgErrBits = 0;// bits within status words
containing error bits
   unsigned short int IMU_ErrorType = 0; // Bits indicating what kind
of error occurred
   int IMU_Id=0;
   int IMU_Kill=0;
   int N_Samples=12; // Number of 100Hz samples to process before
writing to shared memory
   int nSamp = 0;    // number of 100Hz samples actually used to
calculate average
   double Align_theta =0.0;                  // Default IMU mechanical
alignment...
   double Align_phi = 0.0;                   // ...angles (radians).
   double C_Rate=pow(2,-20)*600;             // Coefficients by which
to multiply ...
   double C_Accel=pow(2,-14)*600;            // ...summed data before
writing to shared...
   double C_DeltaAng=pow(2,-33);             // ...memory. They scale
by data LSB,...
   double C_DeltaV=pow(2,-27);               // ...and average flight
control data.
   double Rotation=0.0;                      // Integrated yaw rate
initialization
```

```
   double Omega=2*pi/(24*60*60);             // Earth rate in radians
per second
   double DegRad = 0.01745329;               // Convert degrees to
radians


   // Open IMU shared memory **** FOR TESTING, OPEN IMUd_mmddyy_nn.d
DATA FILE
   // SPK 071006: allocate space for null character in s
   char Filename[17],s[3];
   char ShellCom[] = "date '+IMUd_%m%d%y_  .d' > IMUDateStamp";

   // Real Time Loop Stuff
   int Hz,t_count;
   double t;
   pid_t LoopTimerProxy;
   timer_t LoopTimerId;
   struct itimerspec LoopTimer;
   struct sigevent event;

   // SPK 011106: Added for Neutrino timer pulses
   int timerChId;
   int timerConId;
   int timerRcvId;
   struct _pulse pulseMsg;

   Hz            = 8;
   dt            = 1.0 / (double)Hz;

   // SPK 010306: Get I/O privity for this thread in Neutrino
   // (This process must still be run as root)
   // Note: This replaces the -T1 compiler option from QNX4
   if ( ThreadCtl ( _NTO_TCTL_IO, NULL ) == -1 )
   {
      perror ("I/O Privilege Request failed!\n");
      return;
   }

   system(ShellCom);
   if(FILE_FLAG)
   {
      DateStampInfp = fopen("IMUDateStamp","r");
      for(k=0;k<12;++k) Filename[k] = getc(DateStampInfp);
      k=1;
      while(1)
      {
         if(k<10)
            sprintf(s,"0%d",k);
         else
            sprintf(s,"%d",k);
         Filename[12] = s[0];
         Filename[13] = s[1];
         Filename[14] = '.';
         Filename[15] = 'd';
```

```
      Filename[16] = NULL;
      // SPK 062606: Make this file binary for size/speed reasons
      if((Outfp=fopen(Filename,"r+b"))==NULL)
      {
         Outfp = fopen(Filename,"wb");
         // SPK DEBUG
         if ( Outfp == NULL )
         {
            printf("Cannot open output file!\n");
            exit(0);
         }

         break;
      }
      else
      {
         fclose(Outfp);
         ++k;
         if(k==100)
         {
            printf("Cannot Create File Number 100\n");
            exit(0);
         }
      }
   } //-- end while(1) --
} //-- end if(FILE_FLAG) --

// Open IMU shared memory
if((IMU_Shmid = OpenIMUShm()) == -1)
{
   printf("Cannot Attach IMU Shared Memory\n");
   INIT_ERROR = 1;
}
if((IMUFlag_Shmid = OpenIMUFlagShm()) == -1)
{
   printf("Cannot Create IMU Flag Shared Memory\n");
   INIT_ERROR = 1;
}
if(INIT_ERROR) exit(0);
ResetIMUFlagShm(IMUFlag_Shmid);

// Open HMR shared memory to get psi value to initialize to
if((HMR_Shmid = OpenHMRShm()) == -1)
{
   printf("Cannot Attach HMR Shared Memory\n");
   INIT_ERROR = 1;
}

sleep(2);   // To ensure HMR has a value to initialize to
ReadHMRShm(HMR_Shmid,&psi_out);
CloseHMRShm(HMR_Shmid);

printf("Initial psi=%f degrees.\n", psi_out  );
psi_out=psi_out*DegRad;
```

```
   //-- Read IMU angle offsets from IMU.inp --
   IMUalignfp = fopen("IMU.inp","r");
   if(IMUalignfp)  fscanf(IMUalignfp,"%lf
%lf\n",&Align_theta,&Align_phi);
   fclose(IMUalignfp);

   //-- Read latitude from Nav.inp --
   OpenInputFile();
   while(1)
   {
      read_status = ReadFromInputFile(&FileString[0]);
      if(read_status > 0)
      {
         sscanf(FileString,"%s",&FileCommand[0]);
         if(!strcmp(FileCommand,"END")) break;
      }
      else if(!strcmp(FileCommand,"SET_GPS_ORIGIN"))
      {
         sscanf(FileString,"%s %lf",&FileCommand[0],&Lat0);
         // Convert to dd.dddd
         LatD  = (double) ((int) (Lat0/100.0));
         Lat  = LatD  + (Lat0  - LatD*100.0)/60.0;
      }
   }
   CloseInputFile();
   s_Lat=sin(Lat*pi/180);
   c_Lat=cos(Lat*pi/180);

    //-- Open and flush the RS-422 port --
   Port = open("/dev/ser4", O_RDONLY | O_NOCTTY);
   BytesRead = 2000;
   while(BytesRead == 2000)
   {
      BytesRead = dev_read(Port,ReadBuf,2000,0,0,0,0,0);
   }

   // Initial load of message buffer
   BytesRead = dev_read(Port,ReadBuf,2000,44,0,0,0,0);
   for(i=0;i<44;i++)MsgBuf[i]=ReadBuf[i];

   // SPK 011106: Create channel and connection for timer pulse
   timerChId  = ChannelCreate(0);
   timerConId = ConnectAttach(ND_LOCAL_NODE, 0, timerChId,
_NTO_SIDE_CHANNEL, 0);

   /* Attach to the Timer */
   // SPK 011106: Initialize pulse event using Neutrino macro
   SIGEV_PULSE_INIT( &event, timerConId, getprio(0), MP_PULSE_CODE, 0
);

   // SPK 010306: Neutrino version returns timer ID in third parameter
   // old: LoopTimerId = timer_create(CLOCK_REALTIME,&event);
   timer_create (CLOCK_REALTIME, &event, &LoopTimerId);
```

```
   if(LoopTimerId == -1)
   {
      printf( "Unable to Attach Timer." );
      return;
   }

   /*
    * 1 nano-seconds before initial firing,
    * 1.0/Hz second repetitive timer afterwards.
    */
   LoopTimer.it_value.tv_sec     = 0L;
   LoopTimer.it_value.tv_nsec    = 1L;
   LoopTimer.it_interval.tv_sec  = 0L;
   /* Convert Hz into NanoSecond Period */
   LoopTimer.it_interval.tv_nsec = (int) (1.0/((float)
Hz)*pow(10.0,9.0));
   timer_settime(LoopTimerId,0,&LoopTimer,NULL);

/*********************************************************************/
   /*-- Main loop - once per shared memory write cycle
(N_samples*10msec) --*/

/*********************************************************************/
   i_m=44;
   i_r=0;
   while(1)
   {
      /* Wait for the Proxy */
      // SPK 011106: No longer using old code (or mig4nto version)
      //old: Receive(LoopTimerProxy,0,0);
      timerRcvId = MsgReceive(timerChId, &pulseMsg, sizeof(pulseMsg),
NULL);

      // SPK 011106: Can check if timerRcvId == 0 to verify a pulse was
received and
      //  can check if pulseMsg.code == MP_PULSE_CODE to verify it's
from our timer

      /****************************************************/
      /*-- Read loop - once per IMU data message period --*/
      /****************************************************/
      IMU_Error = 0;

      BytesRead = dev_read(Port,ReadBuf,2000,44,0,0,0,0);
      //printf("bytes read = %d\n", BytesRead);

      LastMsgIdx = getLastMessage(ReadBuf,BytesRead);

      if (LastMsgIdx < 0 )
      {
         IMU_Error = 1;
      }
      else
      {
```

```
         // parse the message
         p  =*(short*)(ReadBuf+LastMsgIdx+2); q
=*(short*)(ReadBuf+LastMsgIdx+4);  r  =*(short*)(ReadBuf+LastMsgIdx+6);
         ax =*(short*)(ReadBuf+LastMsgIdx+8); ay
=*(short*)(ReadBuf+LastMsgIdx+10); az
=*(short*)(ReadBuf+LastMsgIdx+12);
         status1=*(unsigned short*)(ReadBuf+LastMsgIdx+14);
status2=*(unsigned short*)(ReadBuf+LastMsgIdx+16);
         dAx=*(int*)(ReadBuf+LastMsgIdx+18);
dAy=*(int*)(ReadBuf+LastMsgIdx+22);
dAz=*(int*)(ReadBuf+LastMsgIdx+26);
         dVx=*(int*)(ReadBuf+LastMsgIdx+30);
dVy=*(int*)(ReadBuf+LastMsgIdx+34);
dVz=*(int*)(ReadBuf+LastMsgIdx+38);
         msgCkSum=*(unsigned short*)(ReadBuf+LastMsgIdx+42);

         p_out   = (((double)p) * C_Rate);
         q_out   = (((double)q) * C_Rate);
         r_out   = (((double)r) * C_Rate);
         ax_out  = (((double)ax) * C_Accel);
         ay_out  = (((double)ay) * C_Accel);
         az_out  = (((double)az) * C_Accel);
         // Note: IMU outputs negative acceleration values
         g = ((ax_out == 0.0) && (ay_out == 0.0) && (az_out == 0.0)) ?
32.2 :
            pow((pow(ax_out,2)+pow(ay_out,2)+pow(az_out,2)),0.5);

         // Bound input to asin function to prevent NANs
         if ( fabs(ax_out/g) < 1.0 )
         {
            theta_out = asin(ax_out/g);
         }
         else
         {
            theta_out = ( (fabs(ax_out)) / (ax_out) ) *
(PITCH_LIMIT*DegRad);
         }
         // Bound input to asin function to prevent NANs
         if ( fabs(ay_out/(g*cos(theta_out))) < 1.0 )
         {
            phi_out   = -asin(ay_out/(g*cos(theta_out)));
         }
         else
         {
            phi_out   = -( (fabs(ay_out)) / (ay_out) ) *
(ROLL_LIMIT*DegRad);
         }

         // Remove earth rate, convert angular rates into heading rate
(psi dot)
         s_phi  = sin(phi_out);     c_phi  = cos(phi_out);
         s_theta= sin(theta_out);   c_theta= cos(theta_out);
         s_psi  = sin(psi_out);     c_psi  = cos(psi_out);
         q_e=(-c_phi*s_psi*c_Lat-s_phi*c_theta*s_Lat)*Omega;
```

```
         r_e=((s_phi*s_psi+c_phi*s_theta*c_psi)*c_Lat-
c_phi*c_theta*s_Lat)*Omega;
         q_out=q_out-q_e;
         r_out=r_out-r_e;
         psi_dot = s_phi*q_out/c_theta + c_phi*r_out/c_theta;
         dPsi   = psi_dot*dt;
         psi_out = psi_out+dPsi;
      } // if (LastMsgIdx... else...

      ReadIMUFlagShm(IMUFlag_Shmid,&IMU_Kill);
      if(IMU_Kill) break;

      //--- SHARED MEMORY OUTPUT ---

WriteIMUShm(IMU_Shmid,IMU_Id,phi_out,theta_out,psi_out,p_out,q_out,r_ou
t,ax_out,
                  ay_out,az_out,60.0,60.0,IMU_Error);
      IMU_Id++;
      if(IMU_Id>65534) IMU_Id = 0;

   } // while(1)

   ConnectDetach( timerConId );

   /* Get Rid of the Timer */
   timer_delete(LoopTimerId);

   if(FILE_FLAG)
      fclose(Outfp);

   close(Port);
   CloseIMUShm(IMU_Shmid);
   CloseIMUFlagShm(IMUFlag_Shmid);
   //CloseHMRShm(HMR_Shmid);
}
```

APPENDIX E: VEHICLE NAVIGATION FILTER CODE

The following provides the C-code for the navigation filter. This code has been developed over many years of experience and was originally written by D. Marco and A. Healey for use in the ARIES vehicle.

```
/*-----------------------------------------------------------------------
** SPK 010306: Modified for QNX Neutrino 6.3
           - comment out includes for obsolete files in Nav.h
           - include migration library header file in Nav.h
           - start the migration process manager (mig4nto_init)
           - updated timer_create() to Neutrino version
** SPK 010906: Replaced Creceive() with qnx_proxy_detach()
** SPK 011106: Implemented Neutrino timer pulses instead of
                        migration library cover functions
           - no longer need migration process manager
**-----------------------------------------------------------------------
------*/
#include "Nav.h"
// SPK 011106: Included for Neutrino timer pulses
#include <sys/neutrino.h>
#include <sys/netmgr.h>

#define TRUE  1
#define FALSE 0

// SPK 011106: Added for Neutrino timer pulses
#define  Nav_PULSE_CODE  _PULSE_CODE_MINAVAIL

    FILE *Outfp;
    FILE *NavErrorfp;
    FILE *Inputfp;
    FILE *AbortLogfp;
    char FileString[256];  /* This is the Entire Line of the File */
    char FileCommand[256]; /* This is the File Command i.e. Leading
Text */

    double pi     = 3.14159265358979;
    double RadDeg = 57.2957795;
    double DegRad = 0.01745329;

    /* Wait until RDI_AltEst is not equal to 0 - added by ajh 12/1/04
*/

    /* Altitude Filter Stuff */

    int KALMAN_INIT = TRUE;

/* added for RDI wait  12/01/04*/
```

```
    int WAIT = FALSE;
    int count = 0;

    double Alt_est        = 0.0;
    double Alt_dot        = 0.0;
    double Alt_ddot       = 0.0;
    double RDI_AltRawPrev = 0.0;

    FILE *TestFilefp;

main()
{
   int read_status;
   double MaxDepth;
   double MinBatteryVoltage;
   double MaxLeakVoltage;

   int i,j,k,kk,jj,ip;
   int INIT_ERROR        = FALSE;
   int ABORT_FLAG        = FALSE;
   int KALMAN_ABORT_FLAG = FALSE;
   int Nav_Kill          = FALSE;

   /* Real Time Loop Stuff */
   int Hz,t_count;
   double t,dt;
   pid_t LoopTimerProxy;
   timer_t LoopTimerId;
   struct itimerspec LoopTimer;
   struct sigevent event;

   // SPK 011106: Added for Neutrino timer pulses
   int timerChId;
   int timerConId;
   int timerRcvId;
   struct _pulse pulseMsg;

   /* Absolute Date & Time Stuff */
   struct timeb timebuf;
   time_t time_of_day;
   char buf[26];
   struct tm tmbuf;

   char TacticalMessage[16];

   /* date "+%m%d%y%H%M%S" */

   int Nav_Shmid;
   int NavFlag_Shmid;
   int NetFlag_Shmid;
   int RDI_Shmid;
   int MP_Shmid;
```

```
int GPS_Shmid;
int Analog_Shmid;
int Bob_Shmid;
int HMR_Shmid;

double LatD,LongD;

double KG_psi;

int    Nav_Id       = 0;     /* Navigation ID Process (1-65535) */
int    Month;
int    Day;
int    Year;
int    Hour;                 /* 24 Hour GMT                     */
int    Minute;
double Second;               /* Seconds and Fraction of Seconds */
double DepthRaw     = 0.0;   /* Raw Depth from Depth Cell       */
                            /* (No Filtering) (m)              */
double DepthEst     = 0.0; /* Estimated Depth from Depth Filter(m)*/
double DepthDot     = 0.0;   /* Estimated Depth Rate from Depth */
                            /* Filter (m/sec)                  */
int    Nav_Error    = FALSE;

int    RDI_Id;      /* RDI ID Process (1-65535)                */
double RDI_Ug;      /* RDI U Ground (m/sec)                    */
double RDI_Vg;      /* RDI V Ground (m/sec)                    */
double RDI_Wg;      /* RDI W Ground (m/sec)                    */
double RDI_Uf;      /* RDI U Fluid (m/sec)                     */
double RDI_Vf;      /* RDI V Fluid (m/sec)                     */
double RDI_Wf;      /* RDI W Fluid (m/sec)                     */
double RDI_AltRaw;  /* Raw RDI Altitude (m)                    */
double RDI_AltEst;  /* Estimated Altitude from Alt Filter (m)  */
double RDI_AltDot; /* Estimated Altitude Rate from Filter (m/sec)*/
double RDI_Heading; /* RDI Heading (Degrees)                   */
double psi;         /* RDI Heading (Radians)                   */
int    RDI_Error;

int    HMR_Id;      /* HMR3000 Compass Process ID (1-65535)    */
double HMR_HeadingRaw; /* HMR3000 Uncompensated Compass Heading
(Degrees) */
double HMR_Heading; /* HMR3000 Compass Heading Angle (Degrees) */
double HMR_Pitch;   /* HMR3000 Compass Pitch Angle (Degrees)   */
double HMR_Roll;    /* HMR3000 Compass Roll Angle (Degrees)    */
int    HMR_Error;


/* Kalman Filter Stuff */
double RDI_AltRawComp;
int AltZeroCount = 0;
int SKIP_KALMAN = FALSE;
double PrevRDI_AltRawComp = 0.0;

// SPK 060206: Update variable definitions to reflect IMU process
```

```
   // Note: still need to rename all process and shared memory
variables
   int    MP_Id;             // IMU Process ID (1-65535)
   double MP_phi;            // IMU Roll Angle (Rad)
   double MP_theta;          // IMU Pitch Angle (Rad)
   double MP_psi;            // IMU Heading Angle (Rad)
   double KG_q;              // IMU Roll Rate (Rad/Sec)
   double MP_q;              // IMU Pitch Rate (Rad/Sec)
   double MP_r;              // IMU Yaw Rate (Rad/Sec)
   double MP_XAccel;         // IMU X-axis Acceleration (m/sec^2)
   double MP_YAccel;         // IMU Y-axis Acceleration (m/sec^2)
   double KG_r;              // IMU Z-axis Acceleration (m/sec^2)
   double BatteryVoltageRaw; // Raw Battery Voltage Level (Volts)
   double BatteryVoltage;    // Filtered Battery Voltage Level (Volts)
   int    MP_Error;          // IMU_Error

   int    GPS_Id;       /* GPS Process ID (1-65535)                  */
   int    GPS_Signal;   /* Status of Signal 1 = Present, 0 = Not
Present    */
   double Lat0;         /* GPS Latitude Origin in ddmm.mmmmmm        */
   double Long0;        /* GPS Longitude Origin in dddmm.mmmmmm      */
   double LatDeg0;      /* GPS Latitude Origin in dd.ddddd           */
   double LongDeg0;     /* GPS Longitude Origin in ddd.ddddd         */
   double GPS_X;        /* Distance in meters North/South from GPS
Latitude */
                        /* Origin (Lat0)                             */
   double GPS_Y;        /* Distance in meters East/West from GPS
Longitude  */
                        /* Origin (Long0)                            */
   int    Diff;         /* Raw or Diff.                      */
   int    NSv;          /* Number of SVs                     */
   double ToC;          /* Time of Computation               */
   double Lat;          /* Latitude in ddmm.mmmmmm           */
   double LatDeg;       /* Latitude in dd.ddddd              */
   double Long;         /* Longitude in dddmm.mmmmmm         */
   double LongDeg;      /* Longitude in ddd.ddddd            */
   double SbA;          /* Sensor-Based Altitude             */
   double TtTc;         /* True Track/True Course            */
   double SoG;          /* Speed over Ground                 */
   double Vv;           /* Vertical Velocity                 */
   double Pdop;         /* Position dilution of position     */
   double Hdop;         /* Horiz. dilution of position       */
   double Vdop;         /* Vertical dilution of position     */
   double Tdop;         /* Time dilution of position         */
   int    GPS_Error;

   int    Analog_Id;         /* Analog Process ID (1-65535)          */
   double BowLeakVoltage;    /* Bow Section Leak Detector Voltage    */
                             /* Level (Volts)                        */
   double MidLeakVoltage;    /* Mid Section Leak Detector Voltage    */
                             /* Level (Volts)                        */
   double SternLeakVoltage;  /* Stern Section Leak Detector Voltage  */
                             /* Level (Volts)                        */
   double v_ls;              /* Left Screw Speed (Rot/Sec)           */
   double v_rs;              /* Right Screw Speed (Rot/Sec)          */
```

```
   double v_bvt;             /* Bow Vertical Thruster Speed (Rot/Sec)*/
   double v_svt;        /* Stern Vertical Thruster Speed (Rot/Sec)   */
   double v_blt;        /* Bow Lateral Thruster Speed (Rot/Sec)      */
   double v_slt;        /* Stern Lateral Thruster Speed (Rot/Sec)    */
   int    Analog_Error;

   /* Filter Vars */

   /* These are States (Outputs, Estimates from the Filter)          */
   double NavFil_X;        /* Global X Position (meters) Relative to */
                           /* a GPS Origin (LatDeg0,LongDeg0)        */
   double NavFil_Y;        /* Global Y Position (meters) Relative to */
                           /* a GPS Origin (LatDeg0,LongDeg0)        */

   double NavFil_psi;      /* Yaw Angle (Radians)                 */
   double NavFil_r;        /* Yaw Rate (Rad/sec)                  */
   double NavFil_Ug;       /* Longitudinal Ground Speed (m/sec)   */
   double NavFil_Vg;       /* Lateral Ground Speed (m/sec)        */
   double NavFil_Bias_r;   /* Yaw Rate Bias (Rad/sec)             */
   double NavFil_Bias_psi; /* Yaw angle Bias (Radians)            */

   double y[1][8],ym[8];

   double I8[9][9];
   double x[9];
   double x_bar1[9];
   double res[8][2],ym_prev[8],yhat[8];
   double psi0;

   double spsi,cpsi;
   double Adt[9][9];
   double C[8][9];
   double
P[9][9],M[9][9],LC[9][9],ImLC[9][9],P_diag_out[9],P_1_8_out,P_2_8_out;
   double phi[9][9],phi_t[9][9],Pphi_t[9][9],phiPphi_t[9][9];
   double C_t[9][8],MC_t[9][8],CMC_t[8][8];
   double CMCR[8][8],CMCRI[8][8],L[9][8],Lres[9][2];
   double Q[9][9];
   double R[8][8];
   double q[9],nu[8],p[9];

   /* Process Vars */
   int LastRDI_Id         = 0;
   int RDI_IdDupCnt       = 0;
   int MaxRDI_IdDupCnt    = 10;

   int LastGPS_Id         = 0;
   int GPS_IdDupCnt       = 0;
   int MaxGPS_IdDupCnt    = 12;

   int LastMP_Id          = 0;
   int MP_IdDupCnt        = 0;
   int MaxMP_IdDupCnt     = 10;
   // SPK 071806
```

```
int LastMP_Error       = 0;
int MP_ErrorCnt        = 0;
int MaxMP_ErrorCnt     = 10;

int LastAnalog_Id      = 0;
int Analog_IdDupCnt    = 0;
int MaxAnalog_IdDupCnt = 10;

int LastHMR_Id         = 0;
int HMR_IdDupCnt       = 0;
int MaxHMR_IdDupCnt    = 10;

double Coef1;

Coef1 = 3443.9*(1852.47)*(pi/180.0);
/*Coef1 = 111318.8938906694;*/

for(i=1;i<=8;++i)
{
   for(j=1;j<=8;++j)
   {
      Adt[i][j] = 0.0;
      M[i][j]   = 0.0;
      P[i][j]   = 0.0;
      Q[i][j]   = 0.0;
      I8[i][j]  = 0.0;
   }
}

for(i=1;i<=8;++i)
{
   I8[i][i] = 1.0;
   x_bar1[i] = 0.0;
}

for(i=1;i<=7;++i)
{
   for(j=1;j<=7;++j)
   {
      R[i][j]  = 0.0;
   }
}

for(i=1;i<=7;++i)
{
   for(j=1;j<=8;++j)
   {
      C[i][j]  = 0.0;
   }
}

/* Measurement Vector:
   y = [Ug,Vg,psi,r,LatDeg,LongDeg]; */
```

```
/* New Measurement Vector:
   y = [Ug,Vg,psi,r,LatDeg,LongDeg,KG_psi];
   KG_psi = Integrted KG_r;

*/



Hz = 8;
t = 0.0;
t_count = 0;
dt = 1.0/((double) Hz);

NavErrorfp = fopen("NavError.d","w");
AbortLogfp = fopen("Abort.Log","w");
OpenNavDataFile();
OpenInputFile();

if((Nav_Shmid = OpenNavShm()) == -1)
{
   printf("Cannot Attach Nav Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((NavFlag_Shmid = OpenNavFlagShm()) == -1)
{
   printf("Cannot Create NavFlag Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((NetFlag_Shmid = OpenNetFlagShm()) == -1)
{
   printf("Cannot Create NetFlag Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((RDI_Shmid = OpenRDIShm()) == -1)
{
   printf("Cannot Attach RDI Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((MP_Shmid = OpenMotPakShm()) == -1)
{
   printf("Cannot Attach MotPak Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((GPS_Shmid = OpenGPSShm()) == -1)
{
   printf("Cannot Attach GPS Shared Memory\n");
```

```
   INIT_ERROR = TRUE;
}

if((Analog_Shmid = OpenAnalogShm()) == -1)
{
   printf("Cannot Attach Analog Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((Bob_Shmid = OpenBobShm()) == -1)
{
   printf("Cannot Attach Bob Shared Memory\n");
   INIT_ERROR = TRUE;
}

if((HMR_Shmid = OpenHMRShm()) == -1)
{
   printf("Cannot Attach HMR Shared Memory\n");
   INIT_ERROR = TRUE;
}

if(INIT_ERROR) exit(0);

ResetNavFlagShm(NavFlag_Shmid);

WriteTacticalMessage(NetFlag_Shmid,"GO");

/* Get Initialization Values */
while(TRUE)
{
   read_status = ReadFromInputFile(&FileString[0]);

   if(read_status > 0)
   {
      sscanf(FileString,"%s",&FileCommand[0]);

      /* Break if End */
      if(!strcmp(FileCommand,"END"))
      {
         break;
      }
      else if(!strcmp(FileCommand,"SET_MAX_DEPTH"))
      {
         sscanf(FileString,"%s %lf",&FileCommand[0],&MaxDepth);
      }
      else if(!strcmp(FileCommand,"SET_GPS_ORIGIN"))
      {
         sscanf(FileString,"%s %lf %lf",&FileCommand[0],
                                        &Lat0,&Long0);
         /* Convert to dd.dddd */
         LatD  = (double) ((int) (Lat0/100.0));
         LongD = (double) ((int) (Long0/100.0));
```

```
            LatDeg0  = LatD  + (Lat0  - LatD*100.0)/60.0;
            LongDeg0 = LongD + (Long0 - LongD*100.0)/60.0;
         }
         else if(!strcmp(FileCommand,"SET_MIN_BATTERY_VOLTAGE"))
         {
            sscanf(FileString,"%s
%lf",&FileCommand[0],&MinBatteryVoltage);
         }
         else if(!strcmp(FileCommand,"SET_MAX_LEAK_VOLTAGE"))
         {
            sscanf(FileString,"%s
%lf",&FileCommand[0],&MaxLeakVoltage);
         }
         else
         {
            printf("FileCommand Not Recognized\n");
         }
      }
   }

   CloseInputFile();

   /* Read Initial Measurement Vector and Wait until RDI_AltRaw is not
equal to 0 - ajh 12/1/04 */
   while(WAIT)
   {

        ReadRDIShm(RDI_Shmid,&RDI_Id,&RDI_Ug,&RDI_Vg,&RDI_Wg,
                        &RDI_Uf,&RDI_Vf,&RDI_Wf,
                        &RDI_AltRaw,&RDI_Heading,
                        &RDI_Error);
        sleep(1);
        if (RDI_AltRaw > 0.0)
           {
             count = count+1;
           }
         if (count == 3)
           {
             WAIT=FALSE;
           }
   }

   LastRDI_Id = RDI_Id;

   /* Read Initial Measurement Vector */

ReadHMRShm(HMR_Shmid,&HMR_Id,&HMR_HeadingRaw,&HMR_Heading,&HMR_Pitch,
                        &HMR_Roll,
                        &HMR_Error);

   LastHMR_Id = HMR_Id;

   ReadMotPakShm(MP_Shmid,&MP_Id,&MP_phi,&MP_theta,&MP_psi,
                          &KG_q,&MP_q,&MP_r,
```

```
&MP_XAccel,&MP_YAccel,&KG_r,&BatteryVoltageRaw,
                          &BatteryVoltage,
                          &MP_Error);

   // SPK 09/01/05
   psi = MP_psi;

   LastMP_Id  = MP_Id;
   // SPK 071806
   LastMP_Error = MP_Error;


ReadGPSShm(GPS_Shmid,&GPS_Id,&GPS_Signal,&Diff,&NSv,&ToC,&Lat,&LatDeg,
                        &Long,&LongDeg,&SbA,&TtTc,&SoG,&Vv,
                        &Pdop,&Hdop,&Vdop,&Tdop,&GPS_Error);

   LastGPS_Id = GPS_Id;

   ReadAnalogShm(Analog_Shmid,&Analog_Id,&DepthRaw,&DepthEst,&DepthDot,
                 &BowLeakVoltage,&MidLeakVoltage,&SternLeakVoltage,

&v_ls,&v_rs,&v_bvt,&v_svt,&v_blt,&v_slt,&Analog_Error);

   LastAnalog_Id = Analog_Id;

   if(!GPS_Signal)
   {
      /* We Don't have a GPS Signal, Zero GPS_X and GPS_Y */
      GPS_X = 0.0;
      GPS_Y = 0.0;
   }
   else
   {
      GPS_X = Coef1*(LatDeg - LatDeg0);
      GPS_Y = Coef1*(LongDeg - LongDeg0)*cos(DegRad*LatDeg);
   }

   /* Assign Initial Measurement Vector */
   ym[1] = RDI_Ug;

   // SPK 09/01/05
   //ym[2] = RDI_Vg-1.0*KG_r;
   ym[2] = RDI_Vg-1.0*MP_r;

   // SPK 021606: Add delay to ensure we read the latest value from IMU
   sleep(1);
   ym[3] = psi;

   // SPK 09/01/05
   //ym[4] = KG_r;  /* Was MP_r */
   ym[4] = MP_r;
```

```
   ym[5] = GPS_X;
   ym[6] = GPS_Y;

   // SPK 09/01/05
   //ym[7] = KG_psi; /* This is a Virtual Heading Ref. */
   ym[7] = psi;

   /* State Vector:
   x = [NavFil_X NavFil_Y NavFil_psi NavFil_r NavFil_Ug NavFil_Vg
        NavFil_Bias_r NavFil_Bias_psi]; */

   x[1] = ym[5];  /* Initial GPS_X */
   x[2] = ym[6];  /* Initial GPS_Y */
   x[3] = ym[3];
   x[4] = ym[4];
   x[5] = ym[1];
   x[6] = ym[2];
   x[7] = 0.0;    /* Bias = 0 at Start */
   x[8] = 0.0;    /* Bias = 0 at Start */

   /* Init the State Estimate Vector */

   NavFil_X        = x[1];
   NavFil_Y        = x[2];
   NavFil_psi      = x[3];
   NavFil_r        = x[4];
   NavFil_Ug       = x[5];
   NavFil_Vg       = x[6];
   NavFil_Bias_r   = x[7];
   NavFil_Bias_psi = x[8];

   spsi = sin(NavFil_psi);
   cpsi = cos(NavFil_psi);

   Adt[1][3] = (-NavFil_Ug*spsi-NavFil_Vg*cpsi)*dt;
   Adt[1][5] = cpsi*dt;
   Adt[1][6] = -spsi*dt;
   Adt[2][3] = (NavFil_Ug*cpsi-NavFil_Vg*spsi)*dt;
   Adt[2][5] = spsi*dt;
   Adt[2][6] = cpsi*dt;
   Adt[3][4] = 1.0*dt;

   C[1][5] = 1.0;
   C[2][6] = 1.0;
   C[3][3] = 1.0;
   C[3][8] = 1.0; /* 120303 Mod to eliminate Bias learning for Compass
Tests. returned to 1.0 111904 */
   C[4][4] = 1.0;
   C[4][7] = 0.0; // SPK 061406: Force NavFil_Bias_r to 0.0
   C[5][1] = 1.0;
   C[6][2] = 1.0;
   C[7][3] = 0.0; // measurement 7 is not necessary when using the IMU
psi
```

```
   C[7][8] = 0.0; // measurement 7 is not necessary when using the IMU
psi


   /* q[3] = q[4] = 0.01, q[5] = q[6] = 0.1, q[7] = 0.0 */


   /* Orig Vals returned to original 111904 - ajh

   q[1] = 0.0;
   q[2] = 0.0;
   q[3] = 0.001;
   q[4] = 0.1;
   q[5] = 0.01;
   q[6] = 0.01;
   q[7] = 0.0000001;
   q[8] = 0.000001; */

   /* New Vals */

   q[1] = 0.0;       /* variance on LatDeg                    */
   q[2] = 0.0;       /* variance on LongDeg                   */
   q[3] = 0.001;       /* variance on psi, (Radians)^2 returned 111904
ajh         */
   q[4] = 0.1;       /* variance on r, rad/sec)^2             */
   q[5] = 0.01;      /* variance on Ug,(m/sec)^2              */
   q[6] = 0.01;      /* variance on Vg,(m/sec)^2              */
   q[7] = 0.0000001; /* variance on NavFil_Bias_r, (Rad/sec)  */
   q[8] = 0.0;  /* variance on NavFil_Bias_psi (Radians) */

   /* Create Diagonal Q Matrix */
   for(i=1;i<=8;++i)
   {
      Q[i][i] = q[i];
   }

   /* Orig Vals

   // nu[1] = 0.01;
   // nu[2] = 0.01;
   // nu[3] = 0.1;
   // nu[4] = 0.001;
   // nu[5] = 1.0;
   // nu[6] = 1.0; */

   /* New Vals */

   // SPK 081606: Default values
   nu[1] = 0.001;
   nu[2] = 0.0001;
   nu[3] = 0.001;
   nu[4] = 0.001;
   nu[5] = 1.0;
```

```
   nu[6] = 1.0;
   nu[7] = 1.0;

   // SPK 081606: Set elements of R Matrix based on NSv (as verified by
SRV 08/2006)
   if (NSv <= 3)
   {
      nu[5]=1.0;
      nu[6]=1.0;
   }
   else if (NSv == 4)
   {
      nu[5]=0.1;
      nu[6]=0.1;
   }
   else if (NSv >= 5)
   {
      nu[5]=0.01;
      nu[6]=0.01;
   }

   /* Create Diagonal R Matrix */
   for(i=1;i<=7;++i)
   {
      R[i][i] = nu[i];
      /* R[i][i] = nu[i]; */
   }

   p[1] = 0.01;
   p[2] = 0.01;
   p[3] = 0.01;
   p[4] = 0.01;
   p[5] = 0.01;
   p[6] = 0.01;
   p[7] = 0.01;
   p[8] = 0.01;

   /* Create Diagonal P Matrix */
   for(i=1;i<=8;++i)
   {
      P[i][i] = p[i];
   }

   // SPK 060106: Write initialized values to first line of NavD file
   time_of_day = time(NULL);
   gmtime_r(&time_of_day,&tmbuf);
   ftime(&timebuf);
   Month  = tmbuf.tm_mon+1;
   Day    = tmbuf.tm_mday;
   Year   = tmbuf.tm_year+1900;
   Hour   = tmbuf.tm_hour;
   Minute = tmbuf.tm_min;
   Second = ((double) (tmbuf.tm_sec+timebuf.millitm/1000.0));
```

```
   fprintf(Outfp,"%d %d %d %f %d %f %f %f %d %f %f %f %f %f %f %f %f %f
%f %d %d %d %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%d %f %f %f %f %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f\n",
      Nav_Id,
      Hour,
      Minute,
      Second,
      MP_Id,
      KG_q,
      MP_q,
      MP_r,
      RDI_Id,
      RDI_Ug,
      RDI_Vg,
      RDI_Wg,
      RDI_Uf,
      RDI_Vf,
      RDI_Wf,
      0.0,//RDI_AltRawComp,    //SPK: use 0.0 because AltRawComp not
initialized yet
      RDI_AltRaw,//RDI_AltEst, //SPK: use AltRaw because AltEst not
initialized yet
      0.0,//RDI_AltDot,        //SPK: use 0.0 because AltDot not
initialized yet
      RDI_Heading,
      GPS_Id,
      GPS_Signal,
      Diff,
      NSv,
      ToC,
      LatDeg,
      LongDeg,
      TtTc,
      DepthRaw,
      DepthEst,
      DepthDot,
      MP_phi,
      MP_theta,
      v_ls,
      v_rs,
      NavFil_X,
      NavFil_Y,
      NavFil_psi,
      NavFil_r,
      NavFil_Ug,
      NavFil_Vg,
      NavFil_Bias_psi,
      NavFil_Bias_r,
      Analog_Id,
      BatteryVoltageRaw,
      BatteryVoltage,
      GPS_X,
      GPS_Y,
      HMR_Id,
```

```
      HMR_HeadingRaw,
      HMR_Heading,
      HMR_Pitch,
      HMR_Roll,
      KG_r,
      psi,
      Pdop,
      Hdop,
      Vdop,
      Tdop,
      MP_XAccel,
      MP_YAccel,
      P[1][1], // SPK: initial diagonal elements of P matrix
      P[2][2], // SPK: initial diagonal elements of P matrix
      P[3][3], // SPK: initial diagonal elements of P matrix
      P[4][4], // SPK: initial diagonal elements of P matrix
      P[5][5], // SPK: initial diagonal elements of P matrix
      P[6][6], // SPK: initial diagonal elements of P matrix
      P[7][7], // SPK: initial diagonal elements of P matrix
      P[8][8], // SPK: initial diagonal elements of P matrix
      P[1][8], // SPK: initial 1,8 element of P matrix
      P[2][8]);// SPK: initial 2,8 element of P matrix
   // SPK 060106: End write of initialized values to first line of NavD
file


   // SPK 011106: Create channel and connection for timer pulse
   timerChId  = ChannelCreate(0);
   timerConId = ConnectAttach(ND_LOCAL_NODE, 0, timerChId,
_NTO_SIDE_CHANNEL, 0);

   /* Get a Proxy for the Timer to Kick */
   // SPK 011106: No longer using the old code (or mig4nto version)
/*
   LoopTimerProxy = qnx_proxy_attach( 0, 0, 0, -1 );
   if(LoopTimerProxy == -1)
   {
      printf( "Unable to Attach Proxy." );
      return;
   }
*/

   /* Attach to the Timer */
   // SPK 011106: Initialize pulse event using Neutrino macro
   // old: event.sigev_signo = -LoopTimerProxy;
   SIGEV_PULSE_INIT( &event, timerConId, getprio(0), Nav_PULSE_CODE, 0
);

   // SPK 010306: Neutrino version returns timer ID in third paramete
   // old: LoopTimerId = timer_create(CLOCK_REALTIME,&event);
   timer_create (CLOCK_REALTIME, &event, &LoopTimerId);
   if(LoopTimerId == -1)
   {
      printf( "Unable to Attach Timer." );
```

```
      return;
   }

   /*
    * 1 nano-seconds before initial firing,
    * 1.0/Hz second repetitive timer afterwards.
    */
   LoopTimer.it_value.tv_sec     = 0L;
   LoopTimer.it_value.tv_nsec    = 1L;
   LoopTimer.it_interval.tv_sec  = 0L;
   /* Convert Hz into NanoSecond Period */
   LoopTimer.it_interval.tv_nsec = (int) (1.0/((float)
Hz)*pow(10.0,9.0));
   timer_settime(LoopTimerId,0,&LoopTimer,NULL);

   while(TRUE)
   {
      /* Wait for the Proxy */
      // SPK 011106: No longer using old code (or mig4nto version)
      //old: Receive(LoopTimerProxy,0,0);
      timerRcvId = MsgReceive(timerChId, &pulseMsg, sizeof(pulseMsg),
NULL);

      // SPK 011106: Can check if timerRcvId == 0 to verify a pulse was
received and
      //  can check if pulseMsg.code == Nav_PULSE_CODE to verify it's
from our timer

      /* Do Work */

      expAdt88(&Adt,&phi,8);

      spsi = sin(*(x+3));
      cpsi = cos(*(x+3));

      *(x_bar1+1) = *(x+1) + (*(x+5)*cpsi - *(x+6)*spsi)*dt;
      *(x_bar1+2) = *(x+2) + (*(x+5)*spsi + *(x+6)*cpsi)*dt;
      *(x_bar1+3) = *(x+3) + *(x+4)*dt;
      *(x_bar1+4) = *(x+4);
      *(x_bar1+5) = *(x+5);
      *(x_bar1+6) = *(x+6);
      *(x_bar1+7) = *(x+7);
      *(x_bar1+8) = *(x+8);

      transpose(&phi,&phi_t,8,8);
      AdotB(&P,&phi_t,&Pphi_t,8,8,8);
      AdotB(&phi,&Pphi_t,&phiPphi_t,8,8,8);
      ApmB(&phiPphi_t,&Q,&M,8,8,'+');

      *(yhat+1) = *(x_bar1+5);
      *(yhat+2) = *(x_bar1+6);
      *(yhat+3) = *(x_bar1+3) + *(x_bar1+8);
      *(yhat+4) = *(x_bar1+4) + *(x_bar1+7);
      *(yhat+5) = *(x_bar1+1);
```

```
*(yhat+6) = *(x_bar1+2);
*(yhat+7) = *(x_bar1+3) + *(x_bar1+8);

for(j=1;j<=7;++j)
{
   *(ym_prev+j) = *(ym+j);
}

ReadRDIShm(RDI_Shmid,&RDI_Id,&RDI_Ug,&RDI_Vg,&RDI_Wg,
                &RDI_Uf,&RDI_Vf,&RDI_Wf,
                &RDI_AltRaw,&RDI_Heading,
                &RDI_Error);

/* Check for HungUp Process */
if(RDI_Id == LastRDI_Id)
{
  ++RDI_IdDupCnt;
}
else
{
   RDI_IdDupCnt = 0;
}
if(RDI_IdDupCnt >= MaxRDI_IdDupCnt)
{
   fprintf(AbortLogfp,"MaxRDI_IdDupCount = %d @ RDI_Id = %d\n",
       RDI_IdDupCnt,RDI_Id);
   fflush(AbortLogfp);
   WriteTacticalMessage(NetFlag_Shmid,"ABORT");
   ABORT_FLAG = TRUE;
}
LastRDI_Id = RDI_Id;

/* psi = RDI_Heading*DegRad; */

ReadHMRShm(HMR_Shmid,&HMR_Id,&HMR_HeadingRaw,&HMR_Heading,
                     &HMR_Pitch,&HMR_Roll,&HMR_Error);

/* Check for HungUp Process */
if(HMR_Id == LastHMR_Id)
{
  ++HMR_IdDupCnt;
}
else
{
   HMR_IdDupCnt = 0;
}
if(HMR_IdDupCnt >= MaxHMR_IdDupCnt)
{
   fprintf(AbortLogfp,"MaxHMR_IdDupCount = %d @ HMR_Id = %d\n",
       HMR_IdDupCnt,HMR_Id);
   fflush(AbortLogfp);
   WriteTacticalMessage(NetFlag_Shmid,"ABORT");
   ABORT_FLAG = TRUE;
}
```

```
      LastHMR_Id = HMR_Id;

      /********* This is the New psi for the EKF *******/

      // SPK 09/01/05
      //psi = HMR_Heading*DegRad;

      /* psi = RDI_Heading*DegRad;    */


      // SPK 09/01/05
      /* Integrate KG_r to obtain KG_psi */
      //KG_psi = KG_psi + dt*KG_r;

      /* Added to eliminate KG problems 10 Sep 02 */

       /* KG_psi = psi;  */


      ReadMotPakShm(MP_Shmid,&MP_Id,&MP_phi,&MP_theta,&MP_psi,
                            &KG_q,&MP_q,&MP_r,
                            &MP_XAccel,&MP_YAccel,&KG_r,

&BatteryVoltageRaw,&BatteryVoltage,&MP_Error);

      // SPK 09/01/05
      psi = MP_psi;

      /* Bias KG_r */
      /* Perform bias correction in MotPakf.c instead -- SPK 3/22/04
      KG_r = KG_r - 1.219e-4;
      */

      /* Check for HungUp Process */
      if(MP_Id == LastMP_Id)
      {
        ++MP_IdDupCnt;
      }
      else
      {
         MP_IdDupCnt = 0;
      }
      if(MP_IdDupCnt >= MaxMP_IdDupCnt)
      {
         fprintf(AbortLogfp,"MaxMP_IdDupCount = %d @ MP_Id = %d\n",
             MP_IdDupCnt,MP_Id);
         fflush(AbortLogfp);
         WriteTacticalMessage(NetFlag_Shmid,"ABORT");
         ABORT_FLAG = TRUE;
      }
      LastMP_Id = MP_Id;
```

```
ReadGPSShm(GPS_Shmid,&GPS_Id,&GPS_Signal,&Diff,&NSv,&ToC,&Lat,&LatDeg,
                           &Long,&LongDeg,&SbA,&TtTc,&SoG,&Vv,
                           &Pdop,&Hdop,&Vdop,&Tdop,&GPS_Error);

   // SPK 081606: Set elements of R Matrix based on NSv (as verified by
SRV 08/2006)
   if (NSv <= 3)
   {
      nu[5]=1.0;
      nu[6]=1.0;
   }
   else if (NSv == 4)
   {
      nu[5]=0.1;
      nu[6]=0.1;
   }
   else if (NSv >= 5)
   {
      nu[5]=0.01;
      nu[6]=0.01;
   }

      /* Update Diagonal R Matrix */
      for(i=1;i<=7;++i)
      {
         R[i][i] = nu[i];
         /* R[i][i] = nu[i]; */
      }

      /* Check for HungUp Process */
      if(GPS_Id == LastGPS_Id)
      {
        ++GPS_IdDupCnt;
      }
      else
      {
         GPS_IdDupCnt = 0;
      }
      if(GPS_IdDupCnt >= MaxGPS_IdDupCnt)
      {
         fprintf(AbortLogfp,"MaxGPS_IdDupCount = %d @ GPS_Id = %d\n",
             GPS_IdDupCnt,GPS_Id);
         fflush(AbortLogfp);
         WriteTacticalMessage(NetFlag_Shmid,"ABORT");
         ABORT_FLAG = TRUE;
      }
      LastGPS_Id = GPS_Id;

      if(!GPS_Signal)
      {
         /* We Don't have a GPS Signal, Zero GPS_X and GPS_Y */
         GPS_X = 0.0;
         GPS_Y = 0.0;
      }
```

```
      else
      {
         GPS_X = Coef1*(LatDeg - LatDeg0);
         GPS_Y = Coef1*(LongDeg - LongDeg0)*cos(DegRad*LatDeg);
      }


ReadAnalogShm(Analog_Shmid,&Analog_Id,&DepthRaw,&DepthEst,&DepthDot,
                    &BowLeakVoltage,&MidLeakVoltage,&SternLeakVoltage,

&v_ls,&v_rs,&v_bvt,&v_svt,&v_blt,&v_slt,&Analog_Error);

      /* Check for HungUp Process */
      if(Analog_Id == LastAnalog_Id)
      {
        ++Analog_IdDupCnt;
      }
      else
      {
         Analog_IdDupCnt = 0;
      }
      if(Analog_IdDupCnt >= MaxAnalog_IdDupCnt)
      {
         fprintf(AbortLogfp,"MaxAnalog_IdDupCount = %d @ Analog_Id =
%d\n",
             Analog_IdDupCnt,Analog_Id);
         fflush(AbortLogfp);
         WriteTacticalMessage(NetFlag_Shmid,"ABORT");
         ABORT_FLAG = TRUE;
      }
      LastAnalog_Id = Analog_Id;

      /* Assign the Measurements */
      *(ym+1) = RDI_Ug;

      // SPK 09/01/05
      //*(ym+2) = RDI_Vg-1.0*KG_r; /* This Takes into Account the RDI
Offset */
      *(ym+2) = RDI_Vg-1.0*MP_r;

      *(ym+3) = psi;

      // SPK 09/01/05
      //*(ym+4) = KG_r; /* Was MP_r */
      *(ym+4) = MP_r;

      *(ym+5) = GPS_X;
      *(ym+6) = GPS_Y;

      // SPK 09/01/05
      //*(ym+7) = KG_psi; /* This is a Virtual Heading Ref. */
      *(ym+7) = psi;

      for(j=1;j<=7;++j)
```

```
{
   res[j][1] = *(ym+j) - *(yhat+j);
}

/* To Kill Spikes from RDI */

/* Ug & Vg */
for(k=1;k<=2;++k)
{
   if( (*(ym+k) == *(ym_prev+k)) || (fabs(*(ym+k)) > 5.0 )
       || (*(ym+k) == 0.0) )
   {
      for(j=1;j<=8;++j)
      {
         C[k][j] = 0.0;
      }
   }
}

/* Check Psi & r */
for(k=3;k<=4;++k)
{
   if(fabs( *(ym+k) - *(ym_prev+k) ) < 0.000001)
   {
      for(j=1;j<=8;++j)
      {
         C[k][j] = 0.0;
      }
   }
}

/* Check X & Y */
for(k=5;k<=6;++k)
{
   if( fabs( *(ym+k) - *(ym_prev+k) ) < 0.000001 )
   {
      for(j=1;j<=8;++j)
      {
         C[k][j] = 0.0;
      }
   }
}

if(!GPS_Signal)
{
   /* We Don't a GPS Signal, Zero the Rows */
   for(k=5;k<=6;++k)
   {
      for(j=1;j<=8;++j)
      {
         C[k][j] = 0.0;
      }
   }
}
```

```
      transpose(&C,&C_t,7,8);

      AdotB(&M,&C_t,&MC_t,8,8,7);
      AdotB(&C,&MC_t,&CMC_t,7,8,7);
      ApmB(&CMC_t,&R,&CMCR,7,7,'+');
      inv(&CMCR,&CMCRI,7);
      AdotB(&MC_t,&CMCRI,&L,8,7,7);

      AdotB(&L,&C,&LC,8,7,8);
      ApmB(&I8,&LC,&ImLC,8,8,'-');
      AdotB(&ImLC,&M,&P,8,8,8);

      // SPK 11/21/05: store diagonal elements of P matrix
      for(ip=1;ip<=8;ip++)
      {
         P_diag_out[ip] = P[ip][ip];
      }

      // SPK 04/17/06: store elements 1,8 and 2,8 of P matrix
      P_1_8_out = P[1][8];
      P_2_8_out = P[2][8];

      AdotB(&L,&res,&Lres,8,7,1);
      for(j=1;j<=8;++j)
      {
         *(x+j) = *(x_bar1+j) + Lres[j][1];
      }

      // SPK 052406: Force NavFil_Bias_r to 0.0
      // SPK 061406: Undo this change and do it by zeroing the C-Matrix
elements
      //*(x+7) = 0.0;

      spsi = sin(*(x+3));
      cpsi = cos(*(x+3));

      Adt[1][3] = ( -(*(x+5)*spsi)-(*(x+6))*cpsi )*dt;
      Adt[1][5] = cpsi*dt;
      Adt[1][6] = -spsi*dt;
      Adt[2][3] = ( *(x+5)*cpsi - (*(x+6))*spsi)*dt;
      Adt[2][5] = spsi*dt;
      Adt[2][6] = cpsi*dt;
      Adt[3][4] = 1.0*dt;

      C[1][5] = 1.0;
      C[2][6] = 1.0;
      C[3][3] = 1.0;
      C[3][8] = 1.0; /* Turn on Bias learning 090105 ajh */

      C[4][4] = 1.0;
      C[4][7] = 0.0; // SPK 061406: Force NavFil_Bias_r to 0.0
      C[5][1] = 1.0;
```

```
      C[6][2] = 1.0;

      // SPK 09/01/05
      //C[7][3] = 1.0;
      C[7][3] = 0.0;
      C[7][8] = 0.0; /* Mod to turn off Bias learning 050405 ajh */

      NavFil_X        = *(x+1);
      NavFil_Y        = *(x+2);
      NavFil_psi      = *(x+3);
      NavFil_r        = *(x+4);
      NavFil_Ug       = *(x+5);
      NavFil_Vg       = *(x+6);
      NavFil_Bias_r   = *(x+7);
      NavFil_Bias_psi = *(x+8);

      /* Compensate for Pitch & Roll */
      PrevRDI_AltRawComp = RDI_AltRawComp;
      RDI_AltRawComp = RDI_AltRaw*cos(MP_phi)*cos(MP_theta);

      /* Check for Sequential Zeros */
      if(RDI_AltRawComp == 0.0)
      {
         ++AltZeroCount;
         if(AltZeroCount<24)
         {
            SKIP_KALMAN = TRUE;
            RDI_AltDot = 0.0;
         }
         else
         {
            /* Too many Seqential Holidays */
             /* KALMAN_ABORT_FLAG = TRUE; */
            SKIP_KALMAN = TRUE;
         }
      }
      else
      {
         AltZeroCount = 0;
         if(PrevRDI_AltRawComp == 0.0)
         {
            SKIP_KALMAN = FALSE;
            KALMAN_INIT = TRUE;
         }
      }

      /* AltEst and Alt_dot are Filtered Outputs from the Kalman Filter
*/
      if(!SKIP_KALMAN)
      {

AltitudeKalman(RDI_AltRawComp,&RDI_AltEst,&RDI_AltDot,&KALMAN_ABORT_FLA
G);
      }
```

```
      if(KALMAN_ABORT_FLAG == TRUE)
      {
         if(ABORT_FLAG == FALSE)
         {
            WriteTacticalMessage(NetFlag_Shmid,"ABORT");
            fprintf(AbortLogfp,
                   "Abort Due to Unstable Altitude Kalman Filter\n");
            fprintf(AbortLogfp,"RDI_AltRaw = %f\n",RDI_AltRaw);
            fprintf(AbortLogfp,"RDI_AltEst = %f\n",RDI_AltEst);
            fflush(AbortLogfp);
            ABORT_FLAG = TRUE;
         }
      }

      if((DepthRaw > MaxDepth) || (DepthEst > MaxDepth))
      {
         if(ABORT_FLAG == FALSE)
         {
            WriteTacticalMessage(NetFlag_Shmid,"ABORT");
            fprintf(AbortLogfp,"Abort Due to Exceeding MaxDepth =
%f\n",MaxDepth);
            fprintf(AbortLogfp,"DepthRaw = %f\n",DepthRaw);
            fprintf(AbortLogfp,"DepthEst = %f\n",DepthEst);
            fflush(AbortLogfp);
            ABORT_FLAG = TRUE;
         }
      }

      if(BatteryVoltage < MinBatteryVoltage)
      {
         if(ABORT_FLAG == FALSE)
         {
            WriteTacticalMessage(NetFlag_Shmid,"ABORT");
            fprintf(AbortLogfp,
                  "Abort Since Below Minimum Battery Voltage = %f\n",
                           MinBatteryVoltage);
            fprintf(AbortLogfp,"BatteryVoltage = %f\n",BatteryVoltage);
            fflush(AbortLogfp);
            ABORT_FLAG = TRUE;
         }
      }

      if(BowLeakVoltage > MaxLeakVoltage)
      {
         if(ABORT_FLAG == FALSE)
         {
            WriteTacticalMessage(NetFlag_Shmid,"ABORT");
            fprintf(AbortLogfp,"Abort Due to Leak in Bow
Compartment\n");
            fprintf(AbortLogfp,"BowLeakVoltage = %f\n",BowLeakVoltage);
            fflush(AbortLogfp);
            ABORT_FLAG = TRUE;
         }
      }
```

```
      if(MidLeakVoltage > MaxLeakVoltage)
      {
         if(ABORT_FLAG == FALSE)
         {
            WriteTacticalMessage(NetFlag_Shmid,"ABORT");
            fprintf(AbortLogfp,"Abort Due to Leak in Mid
Compartment\n");
            fprintf(AbortLogfp,"MidLeakVoltage = %f\n",MidLeakVoltage);
            fflush(AbortLogfp);
            ABORT_FLAG = TRUE;
         }
      }

      if(SternLeakVoltage > MaxLeakVoltage)
      {
         if(ABORT_FLAG == FALSE)
         {
            WriteTacticalMessage(NetFlag_Shmid,"ABORT");
            fprintf(AbortLogfp,"Abort Due to Leak in Stern
Compartment\n");
            fprintf(AbortLogfp,"SternLeakVoltage =
%f\n",SternLeakVoltage);
            fflush(AbortLogfp);
            ABORT_FLAG = TRUE;
         }
      }


      /* After setting Time using date Execute

         rtc -s -l hw

      */


      time_of_day = time(NULL);

      // SPK 010306: Neutrino uses gmtime_r() instead of _gmtime()
      // old:  _gmtime(&time_of_day,&tmbuf);
      gmtime_r(&time_of_day,&tmbuf);
      ftime(&timebuf);
      Month  = tmbuf.tm_mon+1;
      Day    = tmbuf.tm_mday;
      Year   = tmbuf.tm_year+1900;
      Hour   = tmbuf.tm_hour;
      Minute = tmbuf.tm_min;
      Second = ((double) (tmbuf.tm_sec+timebuf.millitm/1000.0));

      WriteBobShm(Bob_Shmid,Month,Day,Year,Hour,Minute,Second,

NavFil_X,NavFil_Y,DepthEst,RDI_AltEst,GPS_Signal,
                             LatDeg0,LongDeg0,HMR_Heading);
```

```
/* Write Estimates and Direct Measurements Here */
WriteNavShm(Nav_Shmid,
            Nav_Id,
            Month,Day,Year,Hour,Minute,Second,
            NavFil_X,
            NavFil_Y,
            DepthEst,
            RDI_AltEst,
            MP_phi,
            MP_theta,
            NavFil_psi,
            NavFil_Ug,
            NavFil_Vg,
            DepthDot,
            RDI_AltDot,
            KG_q,
            MP_q,
            NavFil_r,
            NavFil_Bias_psi,
            NavFil_Bias_r,
            Diff,
            NSv,
            LatDeg,
            LongDeg,
            TtTc,
            v_ls,
            v_rs,
            v_bvt,
            v_svt,
            v_blt,
            v_slt,
            Lat0,
            Long0,
            Nav_Error);

fprintf(Outfp,"%d %d %d %f %d %f %f %f %d %f %f %f %f %f %f %f %f
%f %f %d %d %d %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %d %f %f %f %f %d %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f
%f %f %f %f %f\n",
      Nav_Id,
      Hour,
      Minute,
      Second,
      MP_Id,
      KG_q,
      MP_q,
      MP_r,
      RDI_Id,
      RDI_Ug,
      RDI_Vg,
      RDI_Wg,
      RDI_Uf,
      RDI_Vf,
      RDI_Wf,
```

```
RDI_AltRawComp,
RDI_AltEst,
RDI_AltDot,
RDI_Heading,
GPS_Id,
GPS_Signal,
Diff,
NSv,
ToC,
LatDeg,
LongDeg,
TtTc,
DepthRaw,
DepthEst,
DepthDot,
MP_phi,
MP_theta,
v_ls,
v_rs,
NavFil_X,
NavFil_Y,
NavFil_psi,
NavFil_r,
NavFil_Ug,
NavFil_Vg,
NavFil_Bias_psi,
NavFil_Bias_r,
Analog_Id,
BatteryVoltageRaw,
BatteryVoltage,
GPS_X,
GPS_Y,
HMR_Id,
HMR_HeadingRaw,
HMR_Heading,
HMR_Pitch,
HMR_Roll,
KG_r,

// SPK 09/01/05
//KG_psi,
psi,

Pdop,
Hdop,
Vdop,
Tdop,

// SPK 10/24/05
MP_XAccel,
MP_YAccel,

// SPK 11/22/05
P_diag_out[1],
P_diag_out[2],
```

```
            P_diag_out[3],
            P_diag_out[4],
            P_diag_out[5],
            P_diag_out[6],
            P_diag_out[7],
            P_diag_out[8],

            // SPK 04/17/06
            P_1_8_out,
            P_2_8_out);

      ReadNavFlagShm(NavFlag_Shmid,&Nav_Kill);
      if(Nav_Kill) break;

      /* Don't Stop Navigator if ABORT */
      /*if(ABORT_FLAG) break; */

      if(Nav_Id>65534)
      {
         Nav_Id = 0;
      }
      ++Nav_Id;

      ++t_count;
      if(t_count == Hz) t_count = 0;
      t = (double) (t_count*dt);
   }

   /* Clear Pending Proxies */
   // SPK 010906: Neutrino equivalent is MsgReceive, preceeded
   // immediately by special timer code; but this requires
   // a channel ID vice a process ID; for now, just use
   // qnx_proxy_detach()
   // old:  while(Creceive(LoopTimerProxy,0,0) == LoopTimerProxy);
   // SPK 011106: No longer using old code (or mig4nto version)
   //replace: qnx_proxy_detach(LoopTimerProxy);
   ConnectDetach( timerConId );

   /* Get Rid of the Timer */
   timer_delete(LoopTimerId);

   CloseNavDataFile();
   fclose(AbortLogfp);
   CloseNavShm(Nav_Shmid);
   CloseNavFlagShm(NavFlag_Shmid);
   CloseNetFlagShm(NetFlag_Shmid);
   CloseRDIShm(RDI_Shmid);
   CloseMotPakShm(MP_Shmid);
   CloseGPSShm(GPS_Shmid);
   CloseAnalogShm(Analog_Shmid);
   CloseBobShm(Bob_Shmid);
   CloseHMRShm(HMR_Shmid);
}
```

LIST OF REFERENCES

Bar-Shalom, Y., Rong Li, X., and Kirubarajan, T., *Estimation with Applications to Tracking and Navigation,* John Wiley & Sons, 2001.

Busse, F., How, J, and Simpson, J., "Demonstration of Adaptive Extended Kalman Filter for Low Earth Orbit Formation Estimation Using CDGPS," paper presented at the Institute of Navigation GPS Meeting, Portland, OR, September 2002.

Department of the Navy (2004). *The Navy Unmanned Undersea Vehicle (UUV) Master Plan.* Retrieved March 26, 2006 from www.chinfo.**navy**.mil/navpalib/technology/**uuv**mp.pdf.

Julier, S. J. and Uhlmann, J. K., "A New Extension of the Kalman Filter to nonlinear systems," in Proceedings of AeroSense: The 11th International Symposium on Aerospace / Defence Sensing, Simulation and Controls, vol. Multi Sensor Fusion, Tracking and Resource Management II, 1997.

Healey, A. J., *Dynamics of Marine Vehicles (MA-4823),* Class Notes, Naval Postgraduate School, Monterey, CA, 1995.

Healey, A. J., *Center for AUV Research Presentation*, Naval Postgraduate School, Monterey, CA, 2006.

Kragelund, S., Re: Hardware Installation. Available e-mail: from spkragel@nps.edu, 13 November 2006.

Marco, D. B. and Healey, A.J., "Command, control, and navigation experimental results with the NPS ARIES AUV," *Oceanic Engineering, IEEE Journal of* , vol.26, no.4, pp. 466-476, October 2001.

Mehra, R. K., "On the Identification of Variances and Adaptive Kalman Filtering," *IEEE Transactions on Automatic Control*, Vol AC-15, No. 2, pp. 175-184, April 1970.

Myers, K. A. and Tapley, B. D., "Adaptive Sequential Estimation with Unknown Noise Statistics," IEEE *Transactions on Automatic Control*, pp. 520-523, August 1976.

Roth, F., "Strapdown Inertial Navigation for Ground Penetrating Radar Data Acquisition: Theory and Experiments," Master's Thesis, Colorado School of Mines, Golden, Colorado, 1999.

Siouris, G., *Aerospace Avionics Systems: A Modern Synthesis*, Academic Press, 1993.

Yakimenko, O. A., *Marine Navigation (MA-4821),* Class Notes, Naval Postgraduate School, Monterey, CA, 2006.

BIBLIOGRAPHY

Arulampalam, M. S., Maskell, S., Gordon, N., Clapp, T., "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *Signal Processing, IEEE Transactions on*, vol.50, no.2, pp.174-188, February 2002.

Cambrideg University Engineering Department Technical Report 380, *The Unscented Particle Filter*, by R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan, 16 April 2000.

Choi, J., Bouchard, M., Hin Yeap, T., Kwon, O., "A Derivative-Free Kalman Filter for Parameter Estimation of Recurrent Neural Networks and Its Applications to Nonlinear Channel Equalization" paper presented at the Fourth International ICSC Symposium on Engineering of Intelligent Systems, February 29 – March 2, 2004.

Healey, A. J., An, E. P., Marco, D.B., "Online compensation of heading sensor bias for low cost AUVs," *Autonomous Underwater Vehicles, 1998. AUV'98. Proceedings Of The 1998 Workshop on* , pp.35-42, 20-21 August 1998

Jetto, L., and Longhi, S., "Development and Experimental Validation of an Adaptive Extended Kalman Filter for the Localization of Mobile Robots," *IEEE Transactions on Robotics and Automation*, Vol. 15, No. 2, pp. 219-229, April 1999.

MIT Marine Robotics Laboratory Technical Memorandum 98-1, *Autonomous Underwater Vehicle Navigation,* by J. J. Leonard, A. A. Bennett, C. M. Smith, H. J. S. Feder, 1 August 1998.

Noriega, G., and Pasupathy, S., "Adaptive Estimation of Noise Covariance Matrices in Real-Time Preprocessing of Geophysical Data," *IEEE Transactions on GEOSCIENCE and Remote Sensing,* Vol. 35, No. 5, pp. 1146-1159, September 1997.

Ristic, B., Arulampalam, S., Gordon, N., *Beyond the Kalman Filter: Particle Filters for Tracking Applications*, Artech House Radar Library, 2004.

Valappil, J., and Georgakis, C., "Systematic Estimation of State Noise Statistics for Extended Kalman Filters," *American Institute of Chemical Engineers Journal*, Vol. 46, No. 2, pp. 292-308, February 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Mechanical Engineering Department Chairman, Code ME
   Naval Postgraduate School
   Monterey, California

4. Mechanical Engineering Curriculum, Code 34
   Naval Postgraduate School
   Monterey, California

5. Professor Anthony J. Healey, Code ME/HY
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, California

6. CAPT J. W. Nicholson
   Department of Weapons and Systems Engineering
   United States Naval Academy
   Annapolis, Maryland

7. Dr. Kwang Sub Song, Code ME/SO
   Department of Mechanical Engineering
   Naval Postgraduate School
   Monterey, California

8. Dr. Tom Swean, Code 32
   Office of Naval Research
   Arlington, Viginia

9. Mr. Scott Willcox
   Bluefin Robotics
   Cambridge, Massachusetts

10. Dr. Chris von Alt
    Hyrdoid, Inc.
    Pocasset, Massachusetts

11. LCDR Steve Vonheeder
Puget Sound Naval Shipyard
Bremerton, Washington