NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TESTING AND EVALUATION OF THE SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS)

by

Suat Arslan

March 2000

Thesis Advisor: Xiaoping Yun
Co-Advisor: Eric Bachmann

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
TESTING AND EVALUATION OF THE SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS).

5. FUNDING NUMBERS

6. AUTHOR(S) Arslan, Suat

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)

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

13. ABSTRACT ***(maximum 200 words)***

At the Naval Postgraduate School (NPS), a small AUV navigation system (SANS) was developed for research in support of shallow-water mine countermeasures and coastal environmental monitoring. The objective of this thesis is to test and evaluate the SANS performance after tuning the filter gains through a series of testing procedures.

The new version of SANS (SANS III) used new hardware components which were smaller, cheaper, and more reliable. A PC/104 computer provided more computing power and, increased the reliability and compatibility of the system.

Implementing an asynchronous Kalman filter in the position and velocity estimation part of the navigation subsystem improved the navigation accuracy significantly. To determine and evaluate the overall system performance, ground vehicle testing was conducted. Test results showed that the SANS III was able to navigate within $\pm$ 15 feet of Global Positioning track with no Global Positioning update for three minutes.

14. SUBJECT TERMS
INS, GPS, AUV,SANS, Navigation, Kalman Filter

13. NUMBER OF PAGES
109

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited

TESTING AND EVALUATION OF THE SMALL AUTONOMOUS UNDERWATER VEHICLE NAVIGATION SYSTEM (SANS)

Suat Arslan
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2000

Author: _____
Suat Arslan

Approved by: _____
Xiaoping Yun, Thesis Advisor

_____
Eric R. Bachmann, Co-Advisor

_____
Jeffrey B. Knorr, Chairman
Department of Electrical and Computer Engineering

ABSTRACT

At the Naval Postgraduate School (NPS), a small AUV navigation system (SANS) was developed for research in support of shallow-water mine countermeasures and coastal environmental monitoring. The objective of this thesis is to test and evaluate the SANS performance after tuning the filter gains through a series of testing procedures.

The new version of SANS (SANS III) used new hardware components which were smaller, cheaper, and more reliable. A PC/104 computer provided more computing power and, increased the reliability and compatibility of the system.

Implementing an asynchronous Kalman filter in the position and velocity estimation part of the navigation subsystem improved the navigation accuracy significantly. To determine and evaluate the overall system performance, ground vehicle testing was conducted. Test results showed that the SANS III was able to navigate within ± 15 feet of Global Positioning track with no Global Positioning update for three minutes.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

During the process of this research, I received assistance from many individuals who made this a tolerable and quite often a pleasurable experience.

I would like to express my sincerest appreciation to my advisors, Prof. Xiaoping Yun and Eric Bachmann, for providing the opportunity for me to work with them. They have been not only advisors, but collaborators, partners, and friends. Thank you very much for giving your support, advice, and guidance to my efforts during this research.

I wish to give special thanks to Prof. Robert McGhee for his contributions. His guidance kept us on the track to accomplish our goal.

I want to express my appreciation to George Ginetti and Jeff Knight, lab technicians in Bullard Hall, for their help and endurance. George, you are a good friend that I will never forget.

I am also much appreciative of all the advice, enduring faith and continuous support that my parents have given me from miles away.

Finally, special thanks are due my wife, Lale Arslan. She excused my absence with patience and understanding. Her sacrifices were numerous and are graciously acknowledged and greatly appreciated. I love you.

I. INTRODUCTION

A. BACKGROUND

Autonomous Underwater Vehicles (AUVs) are capable of a variety of overt and clandestine missions. Such vehicles have been used for inspection, mine countermeasures, survey, and observation [Ref. 1]. One of the most important and difficult aspects of an AUV mission is navigation. Due to intermittent submergence, the vehicle does not have continuous access to external navigational aids such as the Global Positioning System (GPS) or Loran. Thus the vehicle itself must be capable of estimating the current position from onboard sensors that measure speed, heading, velocity and acceleration. Previous studies have shown that integrating GPS and INS into a single system makes possible production of continuously accurate navigational information even when using relatively low-cost components [Ref. 11]. GPS is capable of supplying accurate positioning when the AUV is surfaced. It is integrated with an INS to compensate for the loss of GPS signals when the AUV is submerged.

At the Naval Postgraduate School (NPS), a Small AUV Navigation System (SANS) was developed for research in support of shallow-water mine countermeasures and coastal environmental monitoring. The goal was to demonstrate the feasibility of using a small, low-cost Inertial Measurement Unit (IMU) to navigate between Differential Global Positioning System (DGPS) fixes.

The first prototype of the SANS system (called SANS I) was separated into two subsystems [Ref. 12]. The IMU, water speed sensor, compass, GPS antenna, and data logging computer were housed in one package and placed in a towfish. The GPS

receiver, DGPS antenna, and data processing computer were in the towing vessel. The data collected by the towfish subsystem were transmitted to the processing computer through a modem cable.

The second generation of the SANS system, or SANS II, was totally contained in a single package. The software of SANS I and SANS II was based on a twelve-state constant gain complementary filter. The values of these gains were initially selected based on bandwidth considerations, and later tuned based on results from bench testing and ground vehicle testing.

SANS III, the current version of SANS, is composed of the state-of-the-art components, which include an IMU, GPS/DGPS Receiver, magnetic compass, water speed sensor, and data processing computer. The 486 based ESP computer used in the previous versions of SANS was replaced by an AMD 586DX133 based PC/104 computer to provide more computing power, to increase reliability and provide compatibility with PC/104 industrial standards [Ref. 4].

In the current version of SANS, the position and velocity estimation part of the constant-gain filter used in SANS I and SANS II is replaced by an asynchronous Kalman filter. Kalman filtering is a method of combining all available sensor data, regardless of their precision, to estimate the current position of a vehicle. Use of a Kalman filter improves performance, decreases navigational errors, and improves reliability relative to filters that depend only on one sensor input.

B. RESEARCH OBJECTIVES

The objectives of this thesis research are the following:

- to implement Kalman filtering within the SANS III,

- to improve navigation accuracy of the SANS III by properly tuning filter gains,

-to test and evaluate the overall performance of the SANS III through ground vehicle testing in preparation for the at-sea testing.

C. SCOPE, LIMITATIONS, AND ASSUMPTIONS

This thesis reports part of the findings of more than eight years of research in an ongoing project. The main objective of this thesis is to test and evaluate the current configuration of the SANS through bench and ground vehicle tests, and tune the Kalman filter gains to increase reliability and accuracy of the system before at-sea testing.

D. ORGANIZATION OF THESIS

Chapter II provides an overview and description of the current hardware and software components of the system.

Chapter III describes the Kalman filter for the SANS III.

Chapter IV describes system test procedures and results.

Chapter V presents the thesis conclusions and provides recommendations for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. SYSTEM OVERVIEW

A. INTRODUCTION

References [13 and 14] include detailed information about the current SANS hardware and software components. The purpose of this chapter is to present an overview of the configuration of SANS III hardware and software units.

B. SANS III HARDWARE OVERVIEW

Most of the previous SANS hardware parts have been replaced with more powerful, flexible, and reliable components which are faster, smaller, and cheaper. Figure 2.1 shows the SANS III hardware configuration. The main components are a Real Time Devices 133 MHz AMD 586 PC/104 module, a Sealevel C4-104 four-port PC/104 compatible RS-232 module, a Real Time Devices DM406/DM5406 analog to digital (A/D) converter, a Real Time Devices CMT104 IDE hard drive module and data sensors which include the Crossbow DMU-VG Six Axis IMU, the Motorola Oncore/McKinney Technology GPS/DGPS Receiver unit, the Precision Navigation TCM2 Electronic Compass, and the B&G Sonic Water Speed Sensor.

1. Real Time Devices AMD 586DX133 Based PC/104

The Real Time Devices AMD 586DX133 based PC/104 is being used as a data acquisition and processing unit in SANS III system. PC/104 is an industrial standard for creating embedded computer applications. It fulfills the basic needs of embedded systems such as low power consumption, modularity, small foot print, high reliability, good noise immunity, high-speed operation, and expandability [Ref. 13].

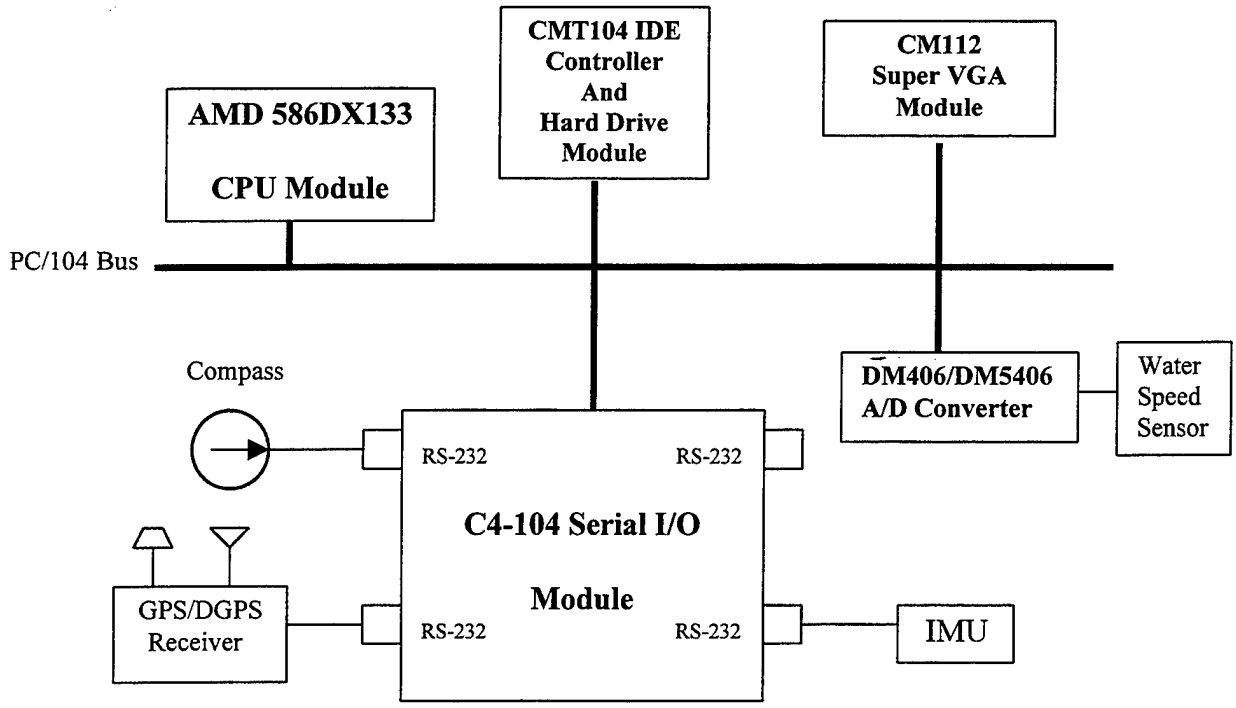

Figure 2.1 Current SANS Hardware Configuration [After Ref. 13]

The PC/104 can be easily customized by stacking PC/104 modules that are compliant with the PC/104 bus architecture, such as video controllers, network interfaces, analog and digital data acquisition modules, sound I/O modules, etc [Ref. 4].

The SANS system is equipped with five PC/104 modules. These are the CPU module, the hard drive module, the VGA module, A/D converter and the serial I/O module.

2. Sealevel C4-104 Serial I/O Module

The PC/104 compatible Sealevel C4-104 serial I/O module provides four RS-232 serial I/O ports for connecting the four sensors of SANS III system. Each serial port can be configured to have its own base memory address and Interrupt Request (IRQ) assignment.

The C4-104 is compliant with PC/104 specification regarding both mechanical and electrical specifications. The C4-104 utilizes Universal Asynchronous Receiver/Transmitters (UART) with programmable baud rates, data format, interrupt control and a 16-byte input and output FIFO [Ref. 7].

3. Real Time Devices DM406/DM5406 Analog to Digital (A/D) Converter

The DM406/DM5406 A/D Converter can receive up to 8 differential or 16 single -ended analog inputs. It converts these inputs into 12-bit digital data words. The analog input voltage range is jumper-selectable for bipolar ranges of –5 to +5 volts, –10 to +10 volts, or a unipolar range of 0 to +10 volts. The module is provided with overvoltage protection to ± 35 volts.

A/D conversions are performed in 5 microseconds, and the maximum throughput rate is 100 kHz. Conversions can be controlled through software, by an on-board pacer clock, or by an external trigger brought onto the board through the I/O connector.

The converted data can be transferred to PC memory through the PC data bus or by using direct memory access (DMA). Detailed information for settings and programming the module is supplied in [Ref. 8].

4. Crossbow DMU-VG Six Axis Inertial Measurement Unit

The DMU-VG is a six-axis measurement system designed to measure linear acceleration along three orthogonal axes, and rotation rates around three orthogonal axes. It is designed to provide stabilized pitch and roll in dynamic environments [Ref. 10]. The IMU has both analog output and RS-232 serial port output.

The DMU-VG is designed to operate in one of two modes, a scaled sensor mode and a voltage mode. It was configured to operate in voltage mode for the SANS.

5. Motorola Oncore/McKinney Technology GPS/DGPS Receiver

The GPS receiver unit used in the SANS system is based upon the Motorola ONCORE Receiver [Ref. 9]. It is capable of tracking eight satellites simultaneously. The GPS receiver incorporates a DGPS capability. The DGPS receiver unit was specially designed by McKinney Technology to operate in the Monterey Bay area. Its data port interface is RS-232 compatible. The output message consists of latitude, longitude, height, velocity, heading, time, and satellite tracking status. The test results conducted show that the GPS unit with differential corrections can provide a positional accuracy of 15 ft.

6. Precision Navigation TCM2 Electronic Compass

The Precision Navigation model TCM2 Electronic Compass Module is used in the current SANS hardware configuration [Ref. 3]. The TCM2 consists of a three-axis magnetometer, a two-axis tilt sensor and a small A/D board. The tilt angle compensation of the sensor depends on the TCM2 model (20, 50, and 80 degrees). Output may include heading, roll, pitch, temperature, and a three dimensional magnetic field measurement. The TCM2 standard output format can be configured to provide those parameters required by the user. Precision Navigation literature lists accuracy as within one half of a degree in level operation. The TCM2 can be calibrated for local magnetic field distortions by the user. It provides an alarm in the output data when local magnetic anomalies are present, and gives out-of-range warnings when the unit is being tilted too far. The calibration of the compass and its error characteristics are described in [Ref 3].

7. B&G Microsonic Speed Sensor

The B&G Water Speed Sensor (Figure 2.2) has an accuracy of 0.05 Knots. It can measure the speeds up to 30-40 Knots depending on the conditions of the signal path.

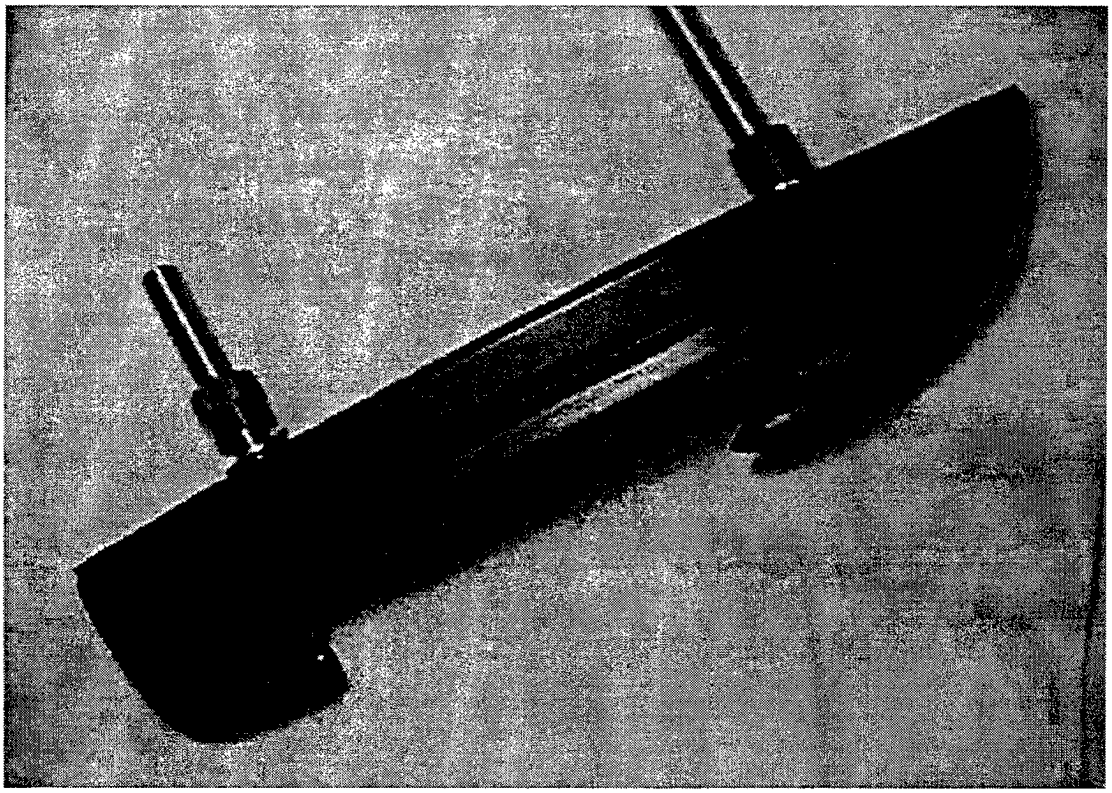

Figure 2.2 Water Speed Sensor

The sensor transmits groups of ultrasonic pulses (short burts of 500 kHz) from one transducer to the other, swapping transducers after each group. Each received pulse precisely triggers the next transmitted pulse. The total time taken for each group depends on the propagation time in the sonic path, hence variations in group time caused by water flow between the transducers can be measured and converted to the speed of water flow.

The microprocessor continually monitors the received pulses, adjusting the gain of the receiver in response to changes in propagation conditions (e.g. air bubbles) and also detects faulty sonic conditions.

The size of the sensor is also appropriate for the concept of the SANS. It has an overall length of sixteen inches, and a height of three inches.

C. SANS III SOFTWARE OVERVIEW

The software of the SANS III was written in C++ and compiled using the Borland 5.0. It is designed to utilize IMU, heading, and water-speed information to implement an INS based on an asynchronous Kalman filter. It also integrates GPS information with INS information to obtain continuously accurate navigation information in real time. The main flow between modules of the system is shown in Figure 2.3.

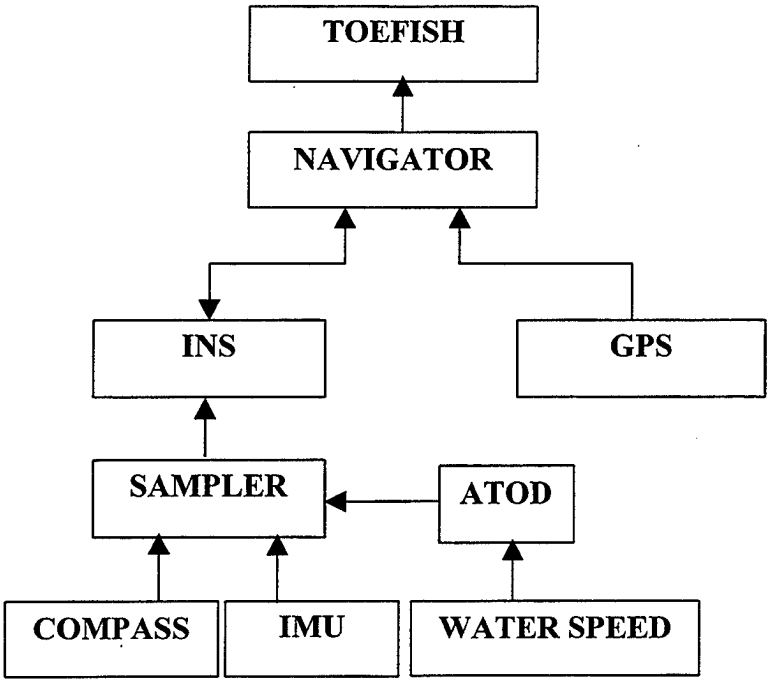

Figure 2.3 SANS Software Main Modules

The modules, with the exception of the water speed sensor, take in the data through the serial ports and supply it to the main program, toefish, for screen set up and output. The Sealevel C4-104 module provides serial port connections for the devices. Water speed data is supplied to the system through the A/D converter.

The GPS class object obtains GPS position messages over a RS-232 interface via the COM1 serial port. The GPS data message length is 76 bytes long. The message should have proper header and a valid checksum in order to be recognized as a valid message.

The compass data is received over a RS-232 interface via the COM2 serial port. The message length for compass data is 60 bytes long. The TCM2 compass is configured to supply magnetic heading information. Magnetic declination [Ref. 18] is added to determine the true north. For Monterey Bay, this value is 15 degrees and 6 minutes [Ref. 17]. It depends on location (latitude, longitude) and year.

IMU data is received over a RS-232 interface via the COM3 serial port. The Crossbow IMU runs in VG mode. The IMU message has 22 bytes of data. The data packet consists of stabilized pitch and roll angles along with angular rate and linear acceleration information [Ref. 13].

The water speed sensor data is received through an A/D converter. The output of the sensor is voltage.

The INS class implements the inertial navigation portion of the SANS. It is based on an asynchronous Kalman filter. The INS class instantiates a Sampler object from which it obtains heading, speed, linear acceleration data, and angular rate data. GPS

information is also passed to the INS class via the Navigator object. The INS produces accurate navigation information by integrating IMU data and DGPS data [Ref 13].

The Sampler prepares raw IMU, heading and water speed data for use by the INS object. The Sampler controls the data formatting and returns a formatted sample if valid raw data is available. Otherwise, a negative response is returned.

The navigator object interfaces with both the GPS and INS objects to determine if they have an updated estimate of the current position, and provides an estimate of the current position in hours, minutes, seconds and milliseconds of latitude and longitude. If GPS information is available, the navigator object converts a latitude and longitude expressed in milliseconds to a grid position in feet and passes it to INS class, so that the INS object can calculate the current position estimate with new GPS information. If no GPS information is available, the INS object calculates the current position estimate by using Kalman Filter equations.

The software description for configuring the Sealevel C4-104 four RS-232 serial ports can be found in [Ref. 13].

D. SUMMARY

The components in the current SANS hardware configuration were chosen based on size, cost, power, and ease of operation. The new components reduced the size of the system by 52% [Ref. 14]. The C4-104 Serial I/O Module provided four RS-232 serial ports for the PC/104 application. The various test results of the SANS III showed that the AMD 586DX133 based PC/104 computer provided more computing power and, more importantly, increased reliability and compatibility with PC/104 industrial standards.

The technological advances in sensors make it possible to change out the current components of the system for the newer versions in future years. With this feature, more accurate navigation information can be acquired, as well as smaller size advantages.

All additions and updates to the SANS software were compiled under the Borland version 5.0, C^{++} compiler. The software runs on a DOS (standard) platform with an AMD 586DX/133 MHz processor.

Since DGPS information is available aperiodically due to asynchronous reacquisition time of satellite signals and asynchronous submergence and surfacing duration of the AUV, an asynchronous Kalman filter is needed to optimally integrate IMU and DGPS data. A Kalman filter was implemented in the INS object to improve the accuracy of SANS.

Minor changes have been made in the SANS software. The changes were mostly related to the Kalman filter part of the system. The gains and constants were tuned to obtain improve accuracy.

Most of the SANS software is still same as in Appendices A and B of [Ref. 13]. The modified part of the SANS software is presented in Appendix A.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SANS ASYNCHRONOUS KALMAN FILTER

A. INTRODUCTION

The objective of the SANS is to produce real time navigation information by integrating an Inertial Navigational System with a Differential Global Positioning System (DGPS). The navigation software in the previous versions of SANS was based on a twelve-state constant gain filter. Filter gains were initially selected based on bandwidth considerations and later tuned based on tilt table and bench testing. This filter is being replaced by an asynchronous Kalman filter in SANS III. This chapter discusses the developed discrete time asynchronous Kalman filter (Figure 3.1) that estimates the eight states of the SANS III model after attitude estimation. A brief discussion will be provided on Kalman filter basics leading into a derivation of the SANS III Kalman filter equations.

B. KALMAN FILTER BASICS

The Kalman filter is a recursive predictive update technique used to estimate the states of a process which can be defined as a discrete-time or continuous-time model. For our purpose, we will study the discrete-time model of the Kalman filter. This state-space filter method has two main features (1) vector modeling of the random processes under consideration, and (2) recursive processing of the noisy measurement (input) data [Ref. 2]. Given some initial estimates, it allows the states of a model to be predicted and adjusted with each new measurement. It also provides an estimate of error at each update. Kalman filters incorporate the effects of measurement noise and modeling noise in their computational structure.

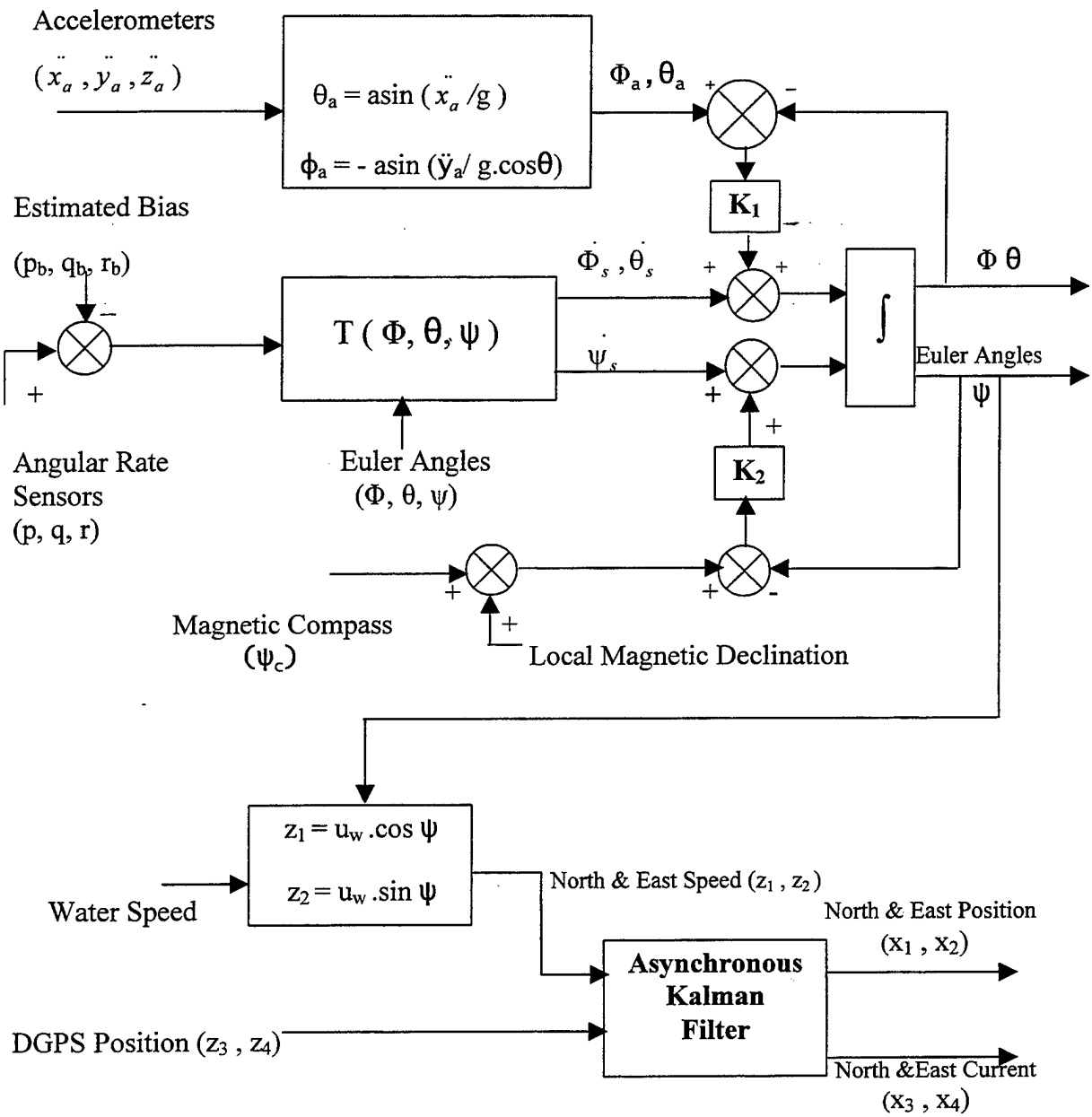

Figure 3.1 SANS Navigation Filter [From Ref. 13]

We assume that the random process to be estimated can be modeled in the form :

$$\mathbf{x}_{k+1} = \phi \mathbf{x}_k + \mathbf{w}_k \tag{3.1}$$

In cases where the relationship between model and its measurements is linear, the observation (measurement) of the random process model (Eq. 3.1) becomes:

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{3.2}$$

Throughout the remainder of this chapter, various terms are used to better explain the Kalman filter. The following notations are consistent with [Ref. 2]:

$\mathbf{x}_k$ (n x 1) Vector containing actual states of a process model at time t_k.

$\hat{\mathbf{x}}_k$ (n x 1) Vector containing the current estimated states at t_k.

$\hat{\mathbf{x}}_k^-$ (n x 1) Vector containing the estimated states at time t before updating with measurement.

$\hat{\mathbf{x}}_{k+1}^-$ (n x 1) Vector containing the estimated states at the next time sample.

$\mathbf{P}_k$ (n x n) Matrix containing the error covariance for $\hat{\mathbf{x}}_k$.

$\mathbf{P}_k^-$ (n x n) Matrix containing the error covariance for $\hat{\mathbf{x}}_k^-$.

$\mathbf{P}_{k+1}^-$ (n x n) Matrix containing the error covariance for $\hat{\mathbf{x}}_{k+1}^-$.

$\mathbf{z}_k$ (m x 1) Vector containing the measurement at time t_k.

$\mathbf{H}_k$ (m x n) Matrix giving the ideal (noiseless) connection between the measurement and the state vector at time t_k.

$\mathbf{R}_k$ (m x n) Matrix denoting the measurement error covariance, which must be known or estimated *a priori.*

ϕ_k (n x n) Matrix containing the model relating $\hat{\mathbf{x}}_k$ and $\hat{\mathbf{x}}_{k+1}^-$ in the absence of a forcing function (if $\hat{\mathbf{x}}_k$ is a sample of a continuous process, ϕ_k is the usual state transition matrix).

$\mathbf{w}_k$ (n x 1) Vector whose elements are white sequences with known covariance structure to be used with $\hat{\mathbf{x}}_k$.

$\mathbf{v}_k$ (m x 1) Vector whose elements are the measurement errors associated with $\mathbf{z}_k$ - assumed to be a white sequence with known covariance structure and having zero cross-correlation with the $\mathbf{w}_k$ sequence.

$\mathbf{Q}_k$ (n x n) Matrix containing the covariance of $\mathbf{w}_k$.

$\mathbf{K}_k$ (n x n) Kalman Gain matrix that relates the amount of influence that the error between $\hat{\mathbf{x}}_k^-$ and $\mathbf{z}_k$ has in deriving $\hat{\mathbf{x}}_k$.

τ_i time constant of a given state vector $\mathbf{x}_k$

q_i white noise variable with zero mean and variance of element being modeled.

$\mathbf{D}_i$ zero frequency white noise density, (magnitude of q_i)

The basic discrete-time Kalman filter contains five recursive equations. Beginning with a prior estimate, the noisy measurement $\mathbf{z}_k$ is used with a blending factor $\mathbf{K}_k$ to improve the the estimate as follows;

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-) \tag{3.3}$$

$\mathbf{K}_k$ (known as the Kalman gain) is now needed to find the optimal estimate $\hat{\mathbf{x}}_k$. It takes the error covariance (mean-square error) between the current state $\mathbf{x}_k$ and the estimated state $\hat{\mathbf{x}}_k$ and applies it with $\mathbf{H}_k$ and $\mathbf{R}_k$ resulting in

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{3.4}$$

Once the Kalman gain $\mathbf{K}_k$ minimizes the mean-square estimation error, a new covariance matrix can be computed for $\hat{\mathbf{x}}_k$ (Eq. 3.3) with

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k^- \tag{3.5}$$

The updated estimate $\hat{\mathbf{x}}_k$ is projected ahead using the state transition matrix ϕ_k. Unlike Equation 3.1 the noise vector $\mathbf{w}_k$ does not affect the projected states because it has zero mean and is white (zero correlation with any previous $\mathbf{w}_k$). Thus,

$$\hat{\mathbf{x}}_{k+1}^- = \phi_k \hat{\mathbf{x}}_k \tag{3.6}$$

Finally, the projected error covariance for $\hat{\mathbf{x}}_{k+1}^-$ uses the updated error covariance $\mathbf{P}_k$ from Equation 3.5, the covariance of $\mathbf{w}_k$ in Equation 3.1, denoted as $\mathbf{Q}_k$, and ϕ_k the state transition matrix to form the following:

$$\mathbf{P}_{k+1}^- = \phi_k \mathbf{P}_k \phi_k^T + \mathbf{Q}_k \tag{3.7}$$

Equations 3.3 through 3.7 can be placed into an algorithm that can loop indefinitely. This Kalman filter loop is shown in Figure 3.2.

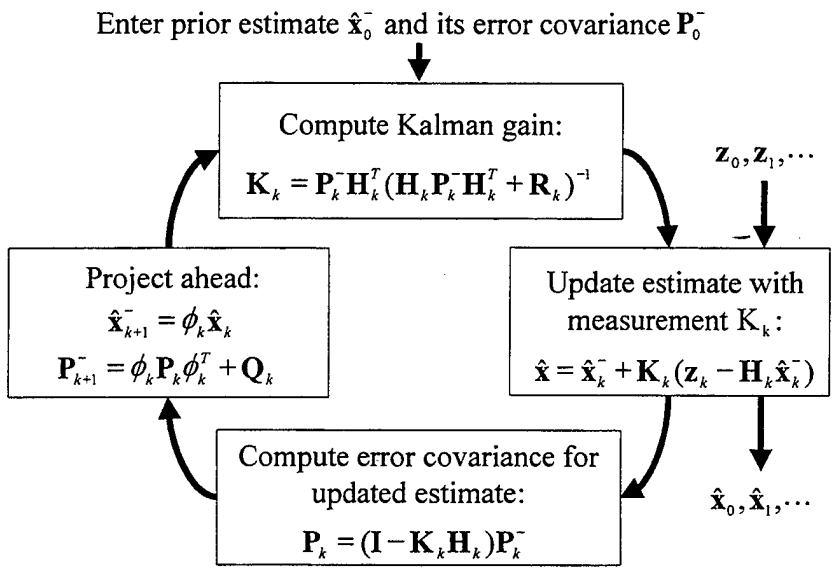

Figure 3.2 Kalman Filter Loop [From Ref. 2]

C. DERIVATION OF SANS III KALMAN FILTER EQUATIONS

The SANS III asynchronous Kalman filter has six states for orientation estimation (still constant gain), and eight states for position estimation. The part of the filter for orientation estimation remains the same as reported in [Ref. 11]. Figure 3.3 shows the process model developed by the SANS team [Ref. 11]. In this model, the velocity relative to water, ocean current, and GPS bias (state variables x_1 through x_6) are modeled as colored signals generated by white noises $q_1,...,q_6$ through first-order systems. The velocity relative to the ground is obtained by summing the velocity relative to water (x_1 and x_2) and the ocean current (x_3 and x_4). This result is integrated to obtain position estimation (x_7 and x_8).

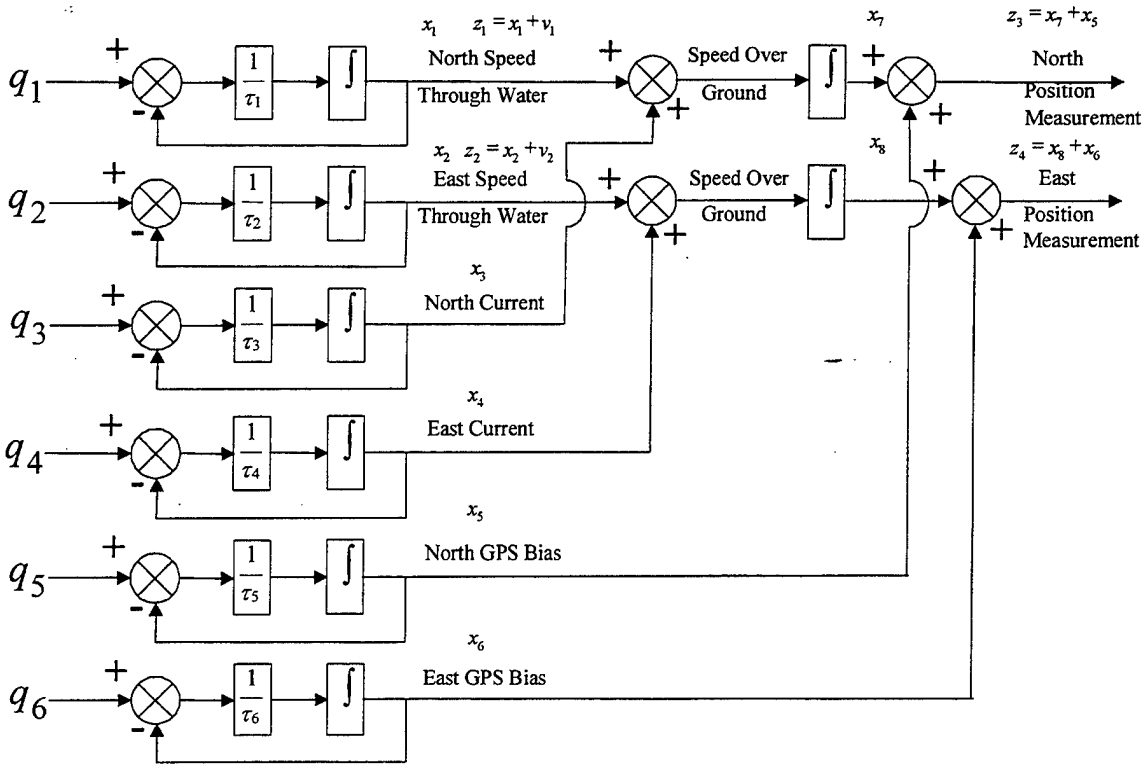

Figure 3.3 SANS Process Model (after attitude estimation) [From Ref. 11]

Low-frequency DGPS noise is explicitly modeled based on an experimentally obtained autocorrelation function. Ocean currents are modeled as a low-frequency random process. Finally, the asynchronous nature of the DGPS measurements resulting from AUV submergence or wave splash on the DGPS antenna is also taken into account by adopting an asynchronous Kalman Filter as the basis for the SANS position estimation software.

Therefore, the Kalman filter state equations are characterized by:

$$\dot{x}_1 = -\frac{1}{\tau_1} x_1 + \frac{1}{\tau_1} q_1 \tag{3.8}$$

$$\dot{x}_2 = -\frac{1}{\tau_2} x_1 + \frac{1}{\tau_2} q_2 \tag{3.9}$$

$$\dot{x}_3 = -\frac{1}{\tau_3} x_3 + \frac{1}{\tau_3} q_3 \tag{3.10}$$

$$\dot{x}_4 = -\frac{1}{\tau_4} x_4 + \frac{1}{\tau_4} q_4 \tag{3.11}$$

$$\dot{x}_5 = -\frac{1}{\tau_5} x_5 + \frac{1}{\tau_5} q_5 \tag{3.12}$$

$$\dot{x}_6 = -\frac{1}{\tau_6} x_6 + \frac{1}{\tau_6} q_6 \tag{3.13}$$

$$\dot{x}_7 = x_1 + x_3 \tag{3.14}$$

$$\dot{x}_8 = x_2 + x_4 \tag{3.15}$$

τ_i represents the time constant of the state vector $\mathbf{x}_i$. τ_1 and τ_2 are the speed through water time constants, τ_3 and τ_4 are the water current time constants, and τ_5 and τ_6 are the GPS time constants.

Acceleration signals from the attitude estimation part of the SANS III filter were not used for velocity estimation because they were judged to be too noisy to provide useful information in comparison to values for velocity obtained directly from an accurate water speed sensor. With the process model shown in Figure 3.3, the measurements used for position estimation are the velocity relative to water provided by the water speed sensor and position information provided by DGPS. The velocity measurements are synchronous and available at every sampling time. DGPS information is asynchronous and is only available when the AUV is surfaced. The two synchronous measurement equations are:

$$z_1 = x_1 + v_1 \tag{3.16}$$

$$z_2 = x_2 + v_2 \tag{3.17}$$

where v_i, i=1,2 are white noise. That is, it is assumed that the velocity measurements contain additive white noise. The two asynchronous measurement equations are:

$$z_3 = x_7 + x_5 \tag{3.18}$$

$$z_4 = x_8 + x_6 \tag{3.19}$$

From these equations, ϕ_k (state transition matrix) was developed.

$$\dot{x}_1 = -\frac{1}{\tau_1}x_1 + \frac{1}{\tau_1}q_1 \tag{3.20}$$

By taking Laplace transform of Equation 3.20:

$$sX_1(s) - x_1(0) = -\frac{1}{\tau_1}x_1(s) + \frac{1}{\tau_1}q_1(s)$$

$$(s + \frac{1}{\tau_1})X_1(s) = x_1(0) + \frac{1}{\tau_1}q_1(s)$$

$$X_1(s) = \frac{1}{(s + \frac{1}{\tau_1})}x_1(0) + \frac{1}{\tau_1(s + \frac{1}{\tau_1})}u_1(s) \tag{3.21}$$

By taking inverse Laplace transform of Equation 3.21:

$$x_1(t) = e^{-\frac{1}{\tau_1}t}x_1(0) + \frac{1}{\tau_1}\int_0^{t_t} e^{-\frac{1}{\tau_1}(t-\tau)}q_1(\tau)d\tau$$

So the discrete time model can be written as:

$$x_1(t_k + 1) = e^{-\frac{1}{\tau_1}\Delta t}x_1(t_k) + \frac{1}{\tau_1}\int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\tau)}q_1(\tau)d\tau,\ \Delta t = t_{k+1} - t_k$$

Similarly, the discrete time model for Equations 3.8 through 3.13 can be developed as:

$$x_i(t_k+1)=e^{-\frac{1}{\tau_i}\Delta t}x_i(t_k)+\frac{1}{\tau_i}\int_{t_k}^{t_{k+1}}e^{-\frac{1}{\tau_i}(t_{k+1}-\tau)}q_i(\tau)d\tau \qquad i=2,3,4,5,6 \qquad (3.22)$$

Next, we need to develop the discrete time model for Equations 3.14 and 3.15.

Taking the Laplace Transform of $\dot{x}_7 = x_1 + x_3$;

$$sX_7(s)-x_7(0)=X_1(s)+X_3(s)$$

$$=\frac{x_1(0)}{s+\frac{1}{\tau_1}}+\frac{u_1(s)}{\tau_1(s+\frac{1}{\tau_1})}+\frac{x_3(0)}{s+\frac{1}{\tau_3}}+\frac{u_3(s)}{\tau_3(s+\frac{1}{\tau_3})}$$

$$X_7(s)=\frac{1}{s}x_7(0)+\frac{x_1(0)}{s(s+\frac{1}{\tau_1})}+\frac{u_1(s)}{\tau_1 s(s+\frac{1}{\tau_1})}+\frac{x_3(0)}{s(s+\frac{1}{\tau_3})}+\frac{u_3(s)}{\tau_3 s(s+\frac{1}{\tau_3})} \qquad (3.23)$$

Finding inverse Laplace transform of $\frac{1}{s(s+\frac{1}{\tau})}$

$$\frac{1}{s(s+\frac{1}{\tau})}=\frac{K_1}{s}+\frac{K_2}{s+\frac{1}{\tau}}=\frac{K_1(s+\frac{1}{\tau})+K_2 s}{s(s+\frac{1}{\tau})}$$

$$\left.\begin{array}{l} K_1=-K_2 \\ K_1\frac{1}{\tau}=1 \end{array}\right\} \Rightarrow K_1=\tau \text{ and } K_2=-\tau$$

$$\frac{1}{s(s+\frac{1}{\tau})}=\frac{\tau}{s}-\frac{\tau}{s+\frac{1}{\tau}} \Rightarrow \tau-\tau e^{\frac{1}{\tau}t}=\tau(1-e^{\frac{1}{\tau}t}) \qquad (3.24)$$

By using the Equation 3.24, we can take the inverse Laplace Transform of Equation 3.23:

$$x_7(t) = x_7(0) + \tau_1(1 - e^{-\frac{1}{\tau_1}t})x_1(0) + \int_0^t (1 - e^{-\frac{1}{\tau_1}(t-\tau)})q_1(\tau)d\tau$$

$$+ \tau_3(1 - e^{-\frac{1}{\tau_3}t})x_3(0) + \int_0^t (1 - e^{-\frac{1}{\tau_3}(t-\tau)})q_3(\tau)d\tau \qquad (3.25)$$

Likewise, solution for Equation 3.15 can be found as:

$$x_8(t) = x_8(0) + \tau_2(1 - e^{-\frac{1}{\tau_2}t})x_2(0) + \int_0^t (1 - e^{-\frac{1}{\tau_2}(t-\tau)})q_2(\tau)d\tau$$

$$+ \tau_4(1 - e^{-\frac{1}{\tau_4}t})x_4(0) + \int_0^t (1 - e^{-\frac{1}{\tau_4}(t-\tau)})q_4(\tau)d\tau \qquad (3.26)$$

and the discrete time model for Equations 3.14 and 3.15 can be modeled by using Equations 3.25 and 3.26:

$$x_7(t_{k+1}) = x_7(t_k) + \tau_1(1 - e^{-\frac{\Delta t}{\tau_1}})x_1(t_k) + \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_1}(t_{k+1}-\tau)})q_1(\tau)d\tau$$

$$+ \tau_3(1 - e^{-\frac{\Delta t}{\tau_3}})x_3(t_k) + \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_3}(t_{k+1}-\tau)})q_3(\tau)d\tau \quad (3.27)$$

$$x_8(t_{k+1}) = x_8(t_k) + \tau_2(1 - e^{-\frac{\Delta t}{\tau_2}})x_2(t_k) + \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_2}(t_{k+1}-\tau)})q_2(\tau)d\tau$$

$$+ \tau_4(1 - e^{-\frac{\Delta t}{\tau_4}})x_4(t_k) + \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_4}(t_{k+1}-\tau)})q_4(\tau)d\tau \quad (3.28)$$

Therefore, the discrete process model of the system is given by

$$x(t_{k+1}) = \Phi_k x(t_k) + w(t_k) \qquad (3.29)$$

Where the state transition matrix Φ_k has the form of

$$\phi_k = \begin{bmatrix} e^{\frac{-|\Delta t|}{\tau_1}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{\frac{-|\Delta t|}{\tau_2}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & e^{\frac{-|\Delta t|}{\tau_3}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & e^{\frac{-|\Delta t|}{\tau_4}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & e^{\frac{-|\Delta t|}{\tau_5}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{\frac{-|\Delta t|}{\tau_6}} & 0 & 0 \\ \tau_1\left(1-e^{\frac{-|\Delta t|}{\tau_1}}\right) & 0 & \tau_3\left(1-e^{\frac{-|\Delta t|}{\tau_3}}\right) & 0 & 0 & 0 & 1 & 0 \\ 0 & \tau_2\left(1-e^{\frac{-|\Delta t|}{\tau_2}}\right) & 0 & \tau_4\left(1-e^{\frac{-|\Delta t|}{\tau_4}}\right) & 0 & 0 & 0 & 1 \end{bmatrix}$$

And the discrete white noises are given by

$$w_i(t_k) = \frac{1}{\tau_i}\int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_i}(t_{k+1}-\tau)} q_i(\tau)d\tau, \quad \text{i=1,2,...,6}$$

$$w_7(t_k) = \int_{t_k}^{t_{k+1}}\left[1-e^{-\frac{1}{\tau_1}(t_{k+1}-\tau)}\right]q_1(\tau)d\tau + \int_{t_k}^{t_{k+1}}\left[1-e^{-\frac{1}{\tau_3}(t_{k+1}-\tau)}\right]q_3(\tau)d\tau \quad (3.30)$$

$$w_8(t_k) = \int_{t_k}^{t_{k+1}}\left[1-e^{-\frac{1}{\tau_2}(t_{k+1}-\tau)}\right]q_2(\tau)d\tau + \int_{t_k}^{t_{k+1}}\left[1-e^{-\frac{1}{\tau_4}(t_{k+1}-\tau)}\right]q_4(\tau)d\tau \quad (3.31)$$

It is noted that $q_i(t)$, i=1,2,3,4,5,6 are continuous independent white noise sources with zero mean and variance D_i. Thus;

$$E[q_i(t)q_j(\tau)] = 0\,,\ \mathrm{i} \neq \mathrm{j}$$

$$E[q_i(t)q_i(\tau)] = D_i\delta(t-\tau)$$

Next, the covariance matrix Q_k of $\mathrm{w}(t_k)$ is computed:

$$Q_k = E[w(t_k)w^T(t_k)]$$

By noting the expression of $\mathrm{w}_i(\mathrm{t}_k)$ from the previous page, Q_k should have the following form :

$$\mathbf{Q}_k = \begin{bmatrix} q_{11} & 0 & 0 & 0 & 0 & 0 & q_{17} & 0 \\ 0 & q_{22} & 0 & 0 & 0 & 0 & 0 & q_{28} \\ 0 & 0 & q_{33} & 0 & 0 & 0 & q_{37} & 0 \\ 0 & 0 & 0 & q_{44} & 0 & 0 & 0 & q_{48} \\ 0 & 0 & 0 & 0 & q_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_{66} & 0 & 0 \\ q_{71} & 0 & q_{73} & 0 & 0 & 0 & q_{77} & 0 \\ 0 & q_{82} & 0 & q_{84} & 0 & 0 & 0 & q_{88} \end{bmatrix}$$

$$q_{11} = E[w_1(t_k)w_1(t_k)]$$

$$= E\left\{\frac{1}{\tau_1}\int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} u_1(\xi)d\xi . \frac{1}{\tau_1}\int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\eta)} u_1(\eta)d\eta\right\}$$

$$= \left\{\frac{1}{\tau_1^{\,2}}\int_{t_k}^{t_{k+1}}\int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} E[u_1(\xi)u_1(\eta)] e^{-\frac{1}{\tau_1}(t_{k+1}-\eta)} d\xi d\eta\right\}$$

$$= \left\{ \frac{1}{\tau_1^{\,2}} \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} e^{-\frac{1}{\tau_1}(t_{k+1}-\eta)} D_1 \delta(\xi-\eta) d\xi d\eta \right\}$$

$$= \frac{D_1}{\tau_1^{\,2}} \int_{t_k}^{t_{k+1}} e^{-\frac{2}{\tau_1}(t_{k+1}-\xi)} d\xi$$

$$= \frac{D_1}{2\tau_1}(1 - e^{-\frac{2\Delta t}{\tau_1}})$$

Likewise;

$$q_{ii} = \frac{D_i}{2\tau_i}(1 - e^{-\frac{2\Delta t}{\tau_i}}) \text{ , i} = 2,3,\ldots,6$$

Deriving equations for the q_{77} and q_{88}:

$$q_{77} = E[w_7(t_k) w_7(t_k)\ ]$$

$$= E\left\{ \left[\int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)}) u_1(\xi) d\xi + \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_3}(t_{k+1}-\xi)}) u_3(\xi) d\xi \right] \right.$$

$$\left. \cdot \left[\int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_1}(t_{k+1}-\eta)}) u_1(\eta) d\eta + \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_3}(t_{k+1}-\eta)}) u_3(\eta) d\eta \right] \right\}$$

$$= \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)}) E[u_1(\xi) u_1(\eta)] (1 - e^{-\frac{1}{\tau_3}(t_{k+1}-\eta)}) d\xi d\eta$$

$$+ \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} (1 - e^{-\frac{1}{\tau_3}(t_{k+1}-\xi)}) E[u_3(\xi) u_3(\eta)] (1 - e^{-\frac{1}{\tau_3}(t_{k+1}-\eta)}) d\xi d\eta$$

$$= D_1 \int_{t_k}^{t_{k+1}} \left[1 - e^{-\frac{t_{k+1}-\xi}{\tau_1}} \right]^2 d\xi \ + \ D_3 \int_{t_k}^{t_{k+1}} \left[1 - e^{-\frac{t_{k+1}-\xi}{\tau_3}} \right]^2 d\xi$$

$$= D_1 \int_{t_k}^{t_{k+1}} \left[1 - 2e^{-\frac{t_{k+1}-\xi}{\tau_1}} + e^{-\frac{2(t_{k+1}-\xi)}{\tau_1}} \right] d\xi + D_3 \int_{t_k}^{t_{k+1}} \left[1 - 2e^{-\frac{t_{k+1}-\xi}{\tau_3}} + e^{-\frac{2(t_{k+1}-\xi)}{\tau_3}} \right] d\xi$$

$$q_{77} = \mathrm{D}_1 \left[\Delta t - 2\tau_1 \left(1 - e^{-\frac{|\Delta t|}{\tau_1}} \right) + \frac{\tau_1}{2} \left(1 - e^{-\frac{2|\Delta t|}{\tau_1}} \right) \right] + \mathrm{D}_3 \left[\Delta t - 2\tau_3 \left(1 - e^{-\frac{|\Delta t|}{\tau_3}} \right) + \frac{\tau_3}{2} \left(1 - e^{-\frac{2|\Delta t|}{\tau_3}} \right) \right]$$

Similarly;

$$q_{88} = \mathrm{D}_2 \left[\Delta t - 2\tau_2 \left(1 - e^{-\frac{|\Delta t|}{\tau_2}} \right) + \frac{\tau_2}{2} \left(1 - e^{-\frac{2|\Delta t|}{\tau_2}} \right) \right] + \mathrm{D}_4 \left[\Delta t - 2\tau_4 \left(1 - e^{-\frac{|\Delta t|}{\tau_4}} \right) + \frac{\tau_4}{2} \left(1 - e^{-\frac{2|\Delta t|}{\tau_4}} \right) \right]$$

For the terms q_{17} and q_{71}:

$$q_{17} = q_{71} = E[w_1(t_k) w_7(t_k)]$$

$$= E\left\{ \frac{1}{\tau_1} \int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} u_1(\xi) d\xi \left[\int_{t_k}^{t_{k+1}} \left(1 - e^{-\frac{t_{k+1}-\eta}{\tau_1}} \right) u_1(\eta) d\eta + \int_{t}^{t_{k+1}} \left(1 - e^{-\frac{t_{k+1}-\eta}{\tau_3}} \right) u_3(\eta) d\eta \right] \right\}$$

$$= \frac{1}{\tau_1} \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} E[u_1(\xi) u_1(\eta)] \left(1 - e^{-\frac{t_{k+1}-\eta}{\tau_3}} \right) d\xi d\eta$$

$$= \frac{D_1}{\tau_1} \int_{t_k}^{t_{k+1}} e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} \left(1 - e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} \right) d\xi$$

$$= \frac{D_1}{\tau_1} \int_{t_k}^{t_{k+1}} \left(e^{-\frac{1}{\tau_1}(t_{k+1}-\xi)} - e^{-\frac{2}{\tau_1}(t_{k+1}-\xi)} \right) d\xi$$

$$= D_1 \left(1 - e^{-\frac{\Delta t}{\tau_1}} - \frac{1}{2} \left(1 - e^{-\frac{2\Delta t}{\tau_1}} \right) \right)$$

$$= D_1\left(\frac{1}{2} - e^{-\frac{\Delta t}{\tau_1}} + \frac{1}{2}e^{-\frac{2\Delta t}{\tau_1}}\right)$$

Therefore;

$$q_{37} = q_{73} = E[w_3(t_k)w_7(t_k)] = D_3\left(\frac{1}{2} - e^{-\frac{\Delta t}{\tau_3}} + \frac{1}{2}e^{-\frac{2\Delta t}{\tau_3}}\right)$$

$$q_{28} = q_{82} = E[w_2(t_k)w_8(t_k)] = D_2\left(\frac{1}{2} - e^{-\frac{\Delta t}{\tau_2}} + \frac{1}{2}e^{-\frac{2\Delta t}{\tau_2}}\right)$$

$$q_{48} = q_{84} = E[w_4(t_k)w_8(t_k)] = D_4\left(\frac{1}{2} - e^{-\frac{\Delta t}{\tau_4}} + \frac{1}{2}e^{-\frac{2\Delta t}{\tau_4}}\right)$$

and $\mathbf{v}_k$ is measurement noise with covariance $\mathbf{R}$,

$$\mathbf{R} = \begin{bmatrix} .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \text{with DGPS signal}$$

$$\mathbf{R} = \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix} \text{without DGPS signal}$$

The **H** matrix is the noiseless connection between measurement and the state vector at time *t*. For the SANS asynchronous Kalman filter, two **H** matrices describe this connection, one for samples with DGPS the other for samples without DGPS.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \text{with DGPS signal}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \text{without DGPS signal}$$

The SANS Kalman filter Model that was developed above, was tested in MATLAB. The model was assumed stationary throughout the simulation. Simulation results of the SANS Kalman filter are presented in the following chapter.

The MATLAB source code for the SANS Kalman filter is presented in Appendix B.

IV. SYSTEM CALIBRATION AND TESTING

A. INTRODUCTION

This chapter presents the calibration of system components, and the testing results of the current SANS configuration. After integrating the new hardware and implementing the new software, calibration of individual devices was required to achieve better results before testing the overall system. Bench testing was performed to determine the functionality and accuracy of the entire system. Ground vehicle testing, with the SANS system was mounted on a moving golf cart, was conducted to demonstrate the feasibility of the SANS system and to observe system performance before at-sea testing. These tests were also conducted to further tune the gains and constants of the SANS Kalman filter.

Matlab simulation results and hardware bench testing results of the SANS Kalman filter are also presented in this chapter.

B. CALIBRATION OF IMU AND COMPASS

1. Tilt Table Calibration of IMU

In order to calibrate the scale factors of accelerometers and determine K1 of the 12-state complementary filter, the IMU unit was placed on a Haas rotary tilt table, model TRT-7 (Figure 4.1). The table has two degrees of freedom (DOF) and is capable of positioning to an accuracy of 0.001 degrees at rates ranging from 0.001 to 80 degrees/s [Ref. 16].

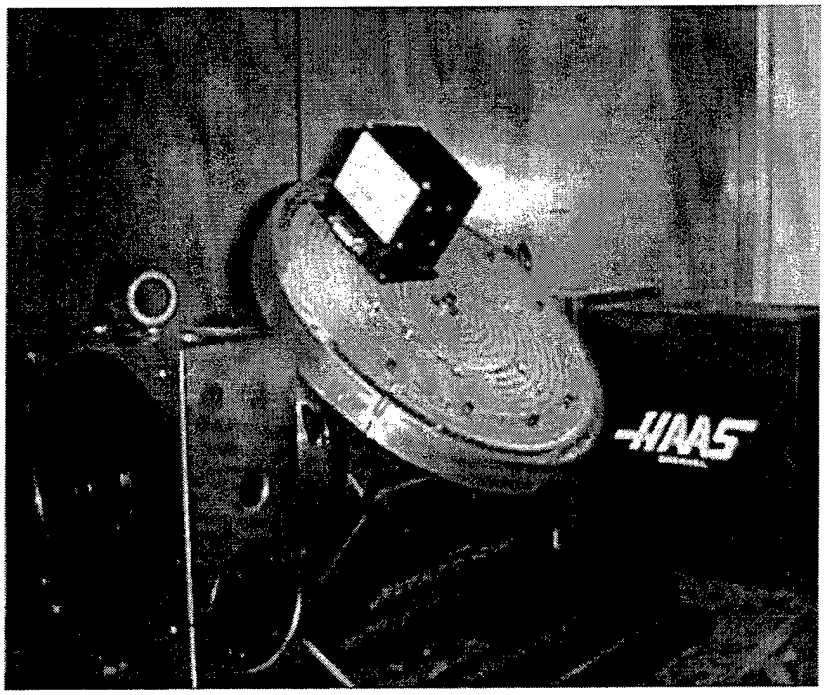

Figure 4.1 Haas Tilt Table

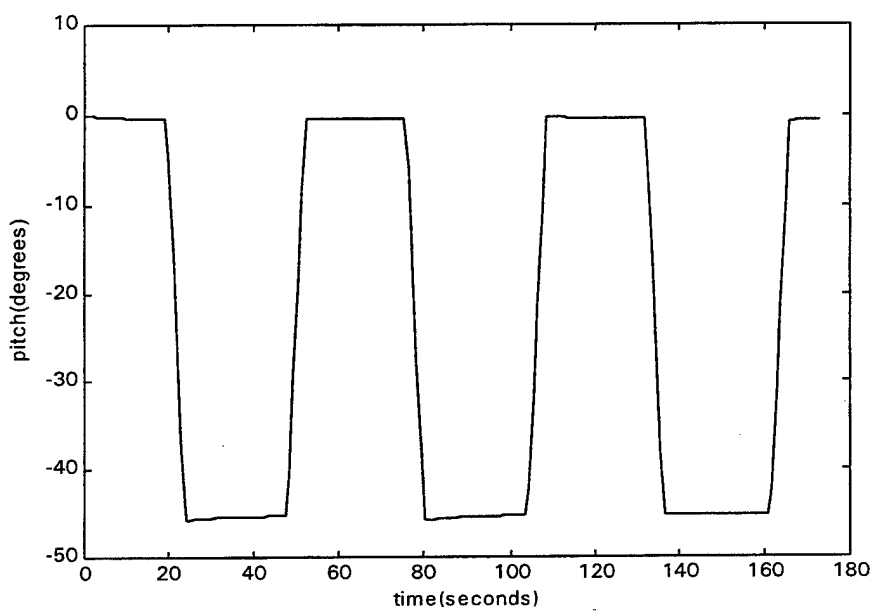

Figure 4.2 IMU Response to 45 Degree Tilt Table Input

Tuning data for the IMU was obtained by moving the unit through each DOF at varying rates within a 45 degree range. The attitude as determined by the unit was then plotted and compared with the actual motion of the table. Through this comparison, it was possible to determine initial gain values and scale factors. The tilt table data was then postprocessed using these initial values and once again compared to the actual motion of the table. This process was repeated several times until the attitude determined by the filter "matched" the true motion of the table.

Figure 4.2 shows an example of the results obtained during the tuning process. The trajectory of the tilt table is not available to plot. Only the response of the IMU unit to 45 degree tilt table input is plotted. However, given the 10 degree per second pitch rate of the tilt table, it can be seen that the system accurately recorded each attitude change as taking 4.5 seconds to complete. The slight overshoots following each motion may indicate that the scale factor for the y- axis angular rate sensor is slightly high. This effect may also be due to undersampling problems. The flatness of the curve following stabilization after each motion indicates that a reasonable gain value has been determined as does the distance between the tails of each step [Ref. 11].

As exemplified by Figure 4.2, tilt table tests of the IMU unit show that attitude sensing is achieved to an accuracy of one degree or better under demanding circumstances.

2. Magnetic Compass Calibration

To obtain the heading information in the SANS, the angular rate sensor is used as a high frequency input source, and the Precision Navigation Electronic Digital Compass (model TCM2) is used as low frequency data source. In a series of ground vehicle tests,

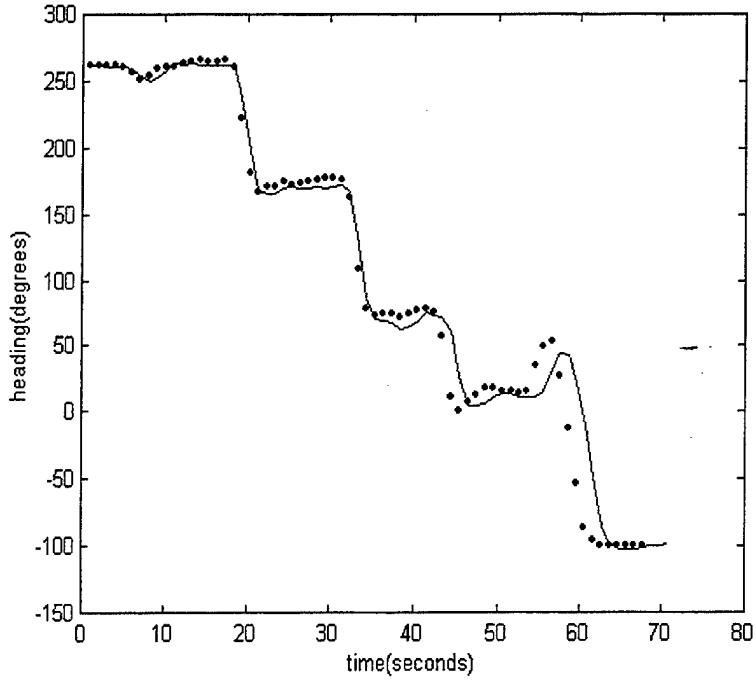

Figure 4.3 Compass Heading without Correction vs. DGPS Track

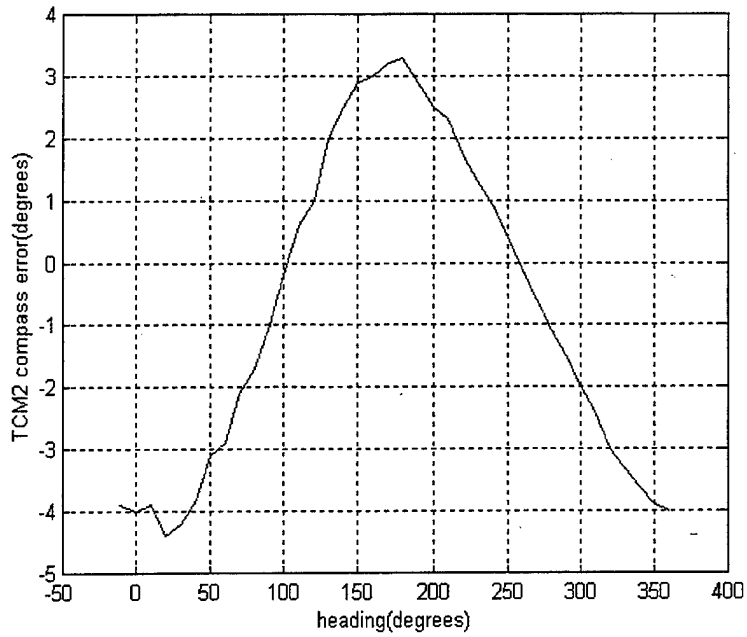

Figure 4.4 TCM2 Compass Error

with continuous GPS update (solid line). It can be seen, particularly during the north-south run, that the filter was not receiving accurate directional information.

The investigation of compass error focused on three areas: possible interference produced by the golf cart electric motor, vibration, and error of the compass itself.

The tests performed by Knapp [Ref. 15] showed that, golf cart electric motor caused interference, but its magnitude appeared to be limited to approximately half of a degree and could be compensated for with an appropriate value for the filter gain. It was also found that the vibration played a role in compass error, but still could be filtered out with an appropriate filter gain.

A final test needed to determine the heading dependent compass errors, because of a different TCM2 was being used by SANS III. The TCM2 compass has a self-calibration [Ref. 3] routine which is designed to remove the effects of static magnetic fields caused by ferrous materials in the vicinity of the compass. The calibration routine is not capable of compensating for dynamic magnetic field distortions. From the ground vehicle testing results, the heading-dependent compass error can be seen. They are especially noticable on north-south runs. A transit (W. and L.E. Gurley) with an accuracy of 0.5 degrees was used as a reference to determine the error of TCM2 compass that was caused by the dynamic magnetic field of the golf cart. The TCM2 compass was mounted in line with the transit on the golf cart facing a distant object with a known magnetic bearing. By taking this object as a starting point, the compass was swung through the entire 360 degrees, taking measurements every 10 degrees by using the optical scale of the transit. A comparison was made between the two indicated headings, the one from the optical scale of transit and the reading from TCM2 compass. Figure 4.4

optical scale of the transit. A comparison was made between the two indicated headings, the one from the optical scale of transit and the reading from TCM2 compass. Figure 4.4 shows the difference between measurements of the transit and those of the TCM2 compass. Using this data, a new lookup table and a linear interpolating function were added to the SANS code to compensate for heading-dependent errors.

The major sensing component of the TCM2 compass is a set of magnetic coils used to sense the magnetic field of the earth. These are not affected by acceleration or inertia as is the "card" in a mechanical compass. The TCM2 does use a "bubble sensor" to determine the attitude. This sensor is affected by linear acceleration to some degree. However, the only purpose of the TCM2 in the SANS system is to provide drift correction for the high frequency heading information obtained from the rate-sensors in the IMU. The magnitude and duration of any accelerations experienced in SANS system are relatively low and TCM2 data is low-pass filtered. Thus, the static accuracy of the TCM2 compass is investigated and not its dynamic accuracy in the presence of extended large magnitude accelerations.

Figure 4.5 shows the data from the TCM2 after compensation using the table data.

C. SYSTEM TESTING RESULTS

1. Simulation and Hardware Bench Testing Results

The asynchronous Kalman filter was simulated in Matlab, and implemented in SANS III. Simulation and hardware bench testing results are presented in this subsection. In simulation, asynchronism is coded as follows. Every 20 to 30 milliseconds, the filter takes measurements, to update the position estimates at the next (synchronous) sample time.

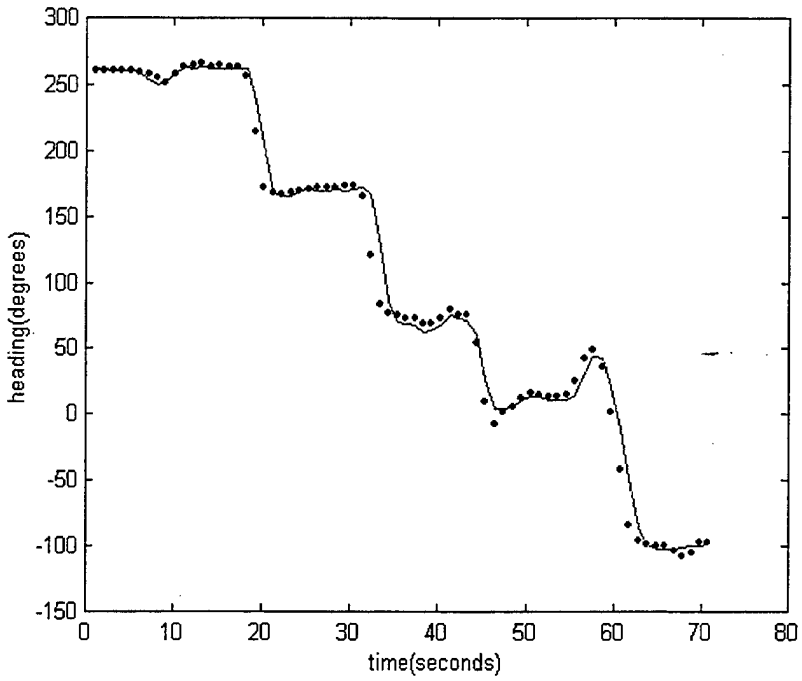

Figure 4.5 Compass Heading with Correction vs. DGPS Track

In hardware testing, the system is fixed on a laboratory bench. The IMU, compass, and DGPS are operational. Water speed information is taken from a fixed voltage source in place of the water speed sensor.

Simulation results are shown in Figures 4.6 and 4.7. The simulation is designed to simulate the hardware system in the stationary bench testing condition. Thus the actual position of the system is fixed at (0,0). It is seen in Figure 4.6 that the accuracy of position estimation is about 15 feet. In Figure 4.7, a north water speed of 10.0 feet/second was entered. Since the system was stationary, the filter was able to estimate that there was a north current at about –10.0 feet/second as shown in Figure 4.7. The estimated east current stayed zero, as expected.

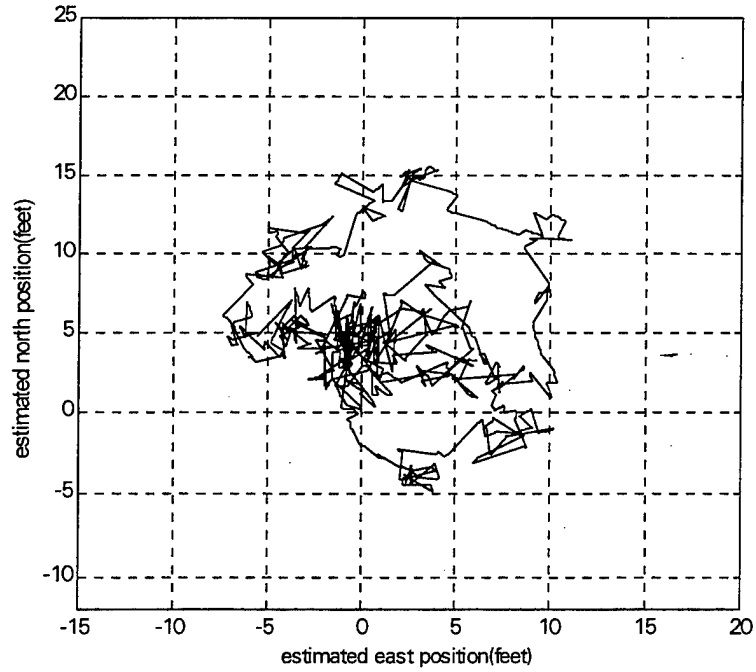

Figure 4.6 Simulation Result: Estimated North Position Versus Estimated East Position

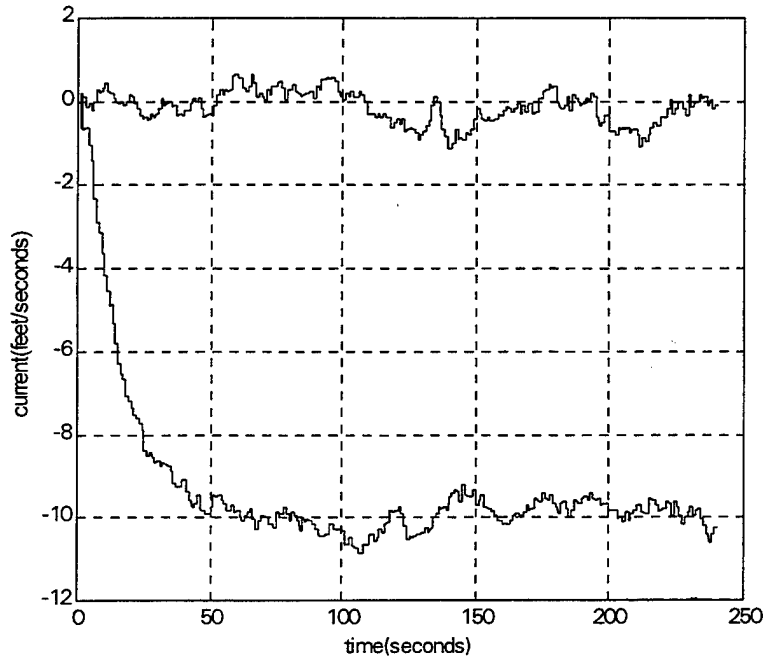

Figure 4.7 Simulation Result: Estimated North (lower curve) and East Current (upper curve)

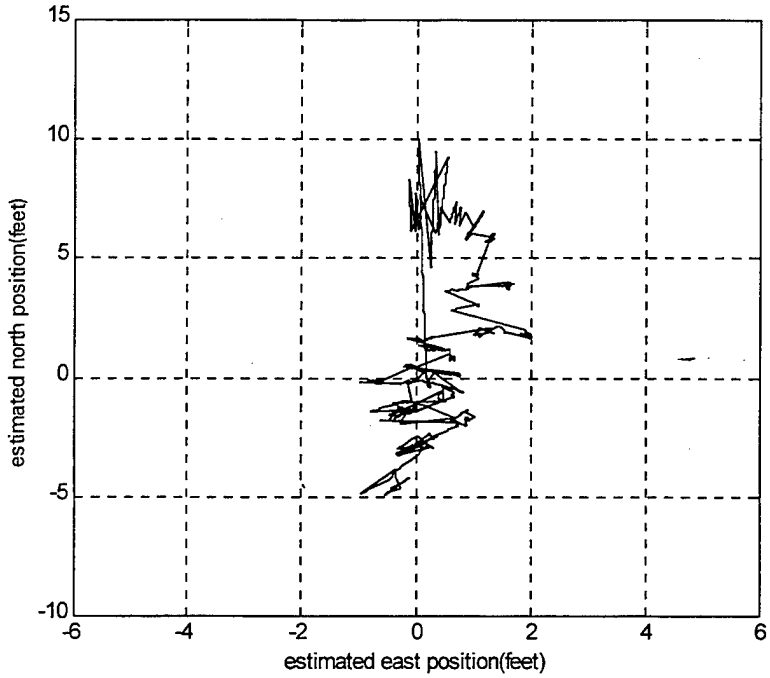

Figure 4.8 Hardware Bench Test Result: Estimated North Position vs. Estimated East Position

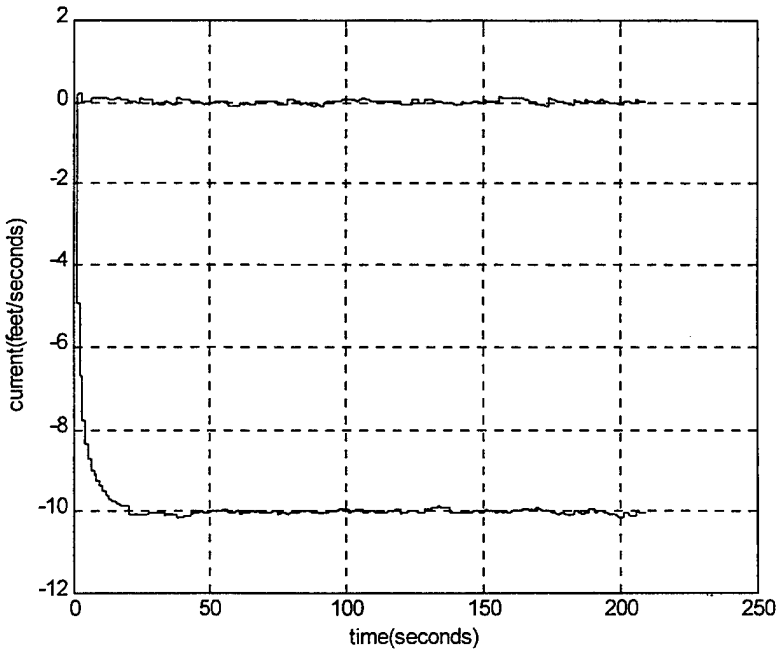

Figure 4.9 Hardware Bench Test Result: Estimated North (lower curve) And East Currents(upper curve)

Examples of hardware bench testing results are shown in Figures 4.8 and 4.9. In Figure 4.8, it is especially interesting to note that there is a long straightline segment starting from (0.2,0.0) ending at (0.0,10.0). This is exactly the estimated motion trajectory of the system during the first second of operation. During this time period, the filter relied entirely on inertial data. Since the water speed sensor indicated a water speed of 10.0 feet/second in the north direction, the filter estimated a northerly motion. As soon as a GPS fix was obtained, the filter updated position and current estimates. After about 20 seconds, the estimated north current converges to 10.0 feet/second.

Matlab simulation code for Kalman filtering is presented in Appendix B.

2. Ground Vehicle Testing Results

In addition to the bench testing, a series of ground vehicle tests were performed in preparation for at-sea testing. The SANS III system was placed on a golf cart (Figure 4.10) and was driven along a predetermined path. Most of the tests were conducted around a surveyed course in the parking lot next to the Mechanical Engineering Auditorium at Naval Postgraduate School. Some additional tests were also performed in the parking lot at the Navy Golf Course to observe the behavior of the system in a different location.

Tests were conducted with continuous DGPS and without DGPS. The path of the parking lot (at school) taken with continuous DGPS is shown in Figure 4.11. The cart started from the (0,0) position and traveled northward (at 350 degrees) for about 450 feet, made a U-turn, traveled southward 248 feet (at 160 degrees) and about 210 feet (at 150 degrees), made a last turn to westward and returned to the starting point along the last leg

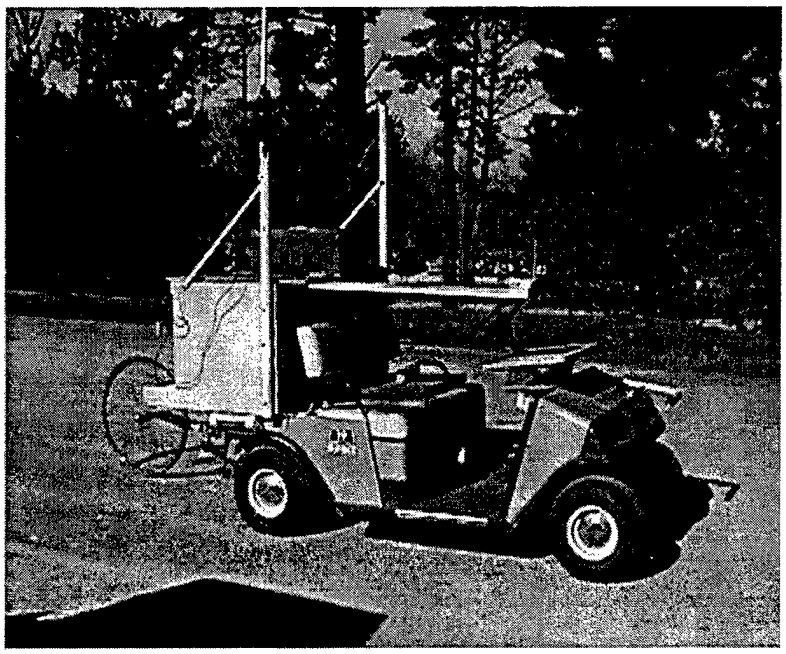

Figure 4.10 Golf Cart

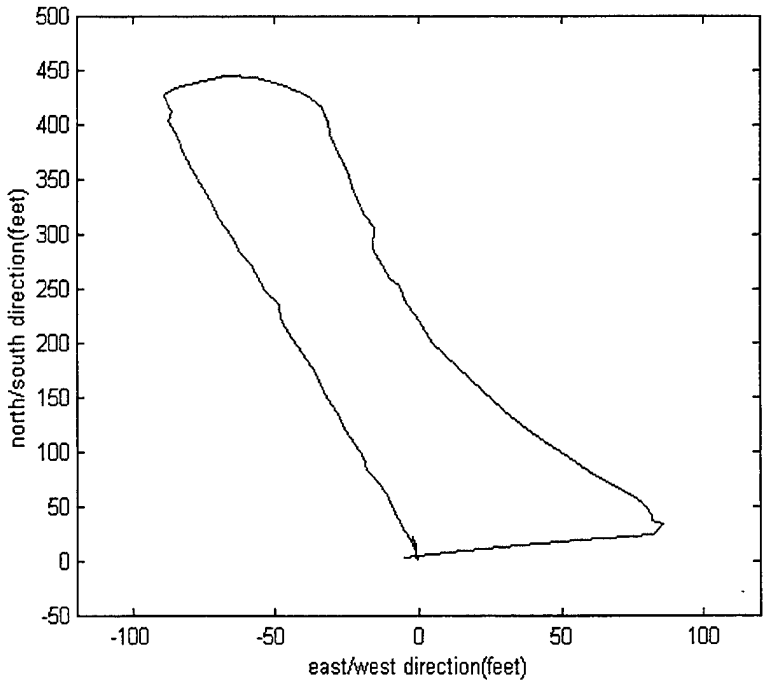

Figure 4.11 Parking Lot Test Track (at School)

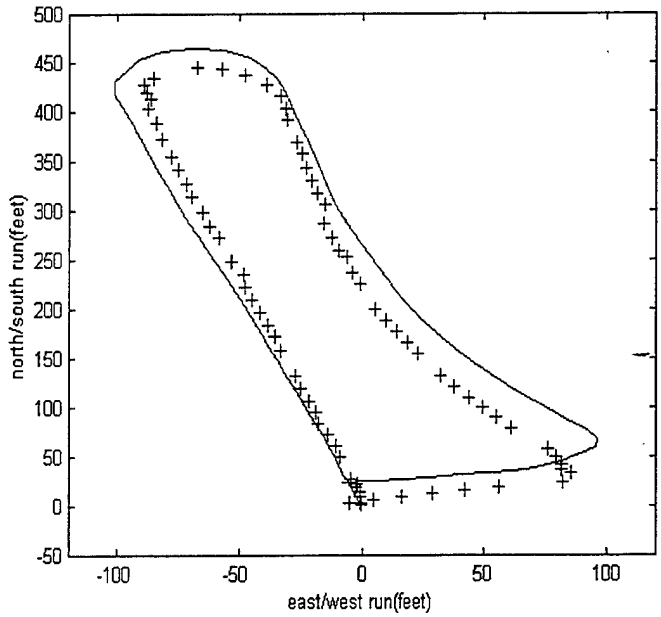

Figure 4.12 Parking Lot Track with Continuous DGPS vs. without DGPS (without compass correction)

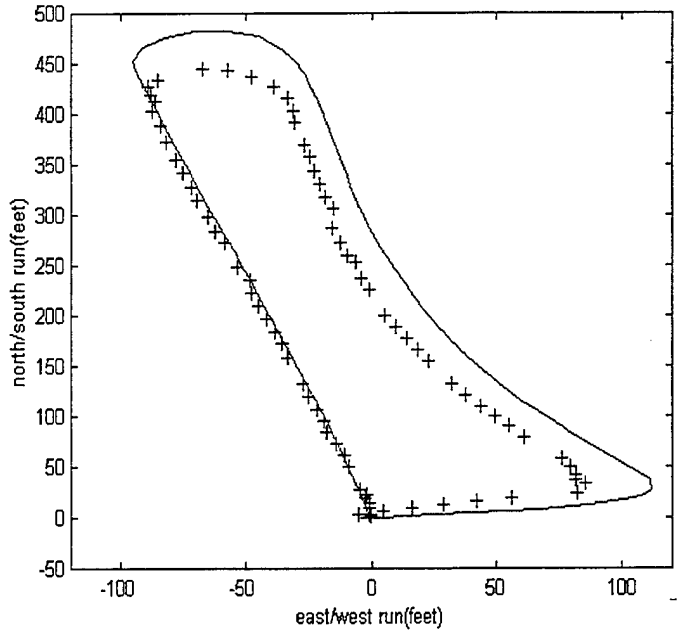

Figure 4.13 Parking Lot Track with Continuous DGPS vs. without DGPS (with compass correction)

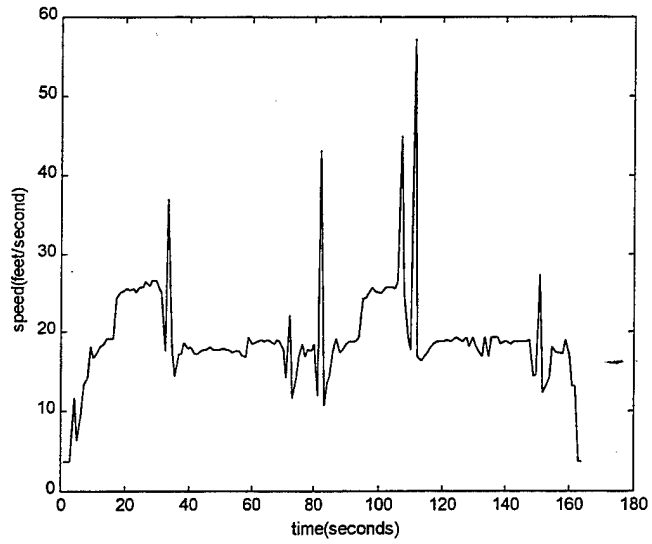

Figure 4.14 Speed Data Through First Runs

$$newSpeed = n*measuredSpeed + (1-n)*previousSpeed \quad (4.2)$$

$$previousSpeed = newSpeed \quad (4.3)$$

The speed value was also scaled, because the later test results showed that the calculated speed value from the circuit output was slightly higher than the actual value.

These corrections improved the speed data significantly (Figure 4.15).

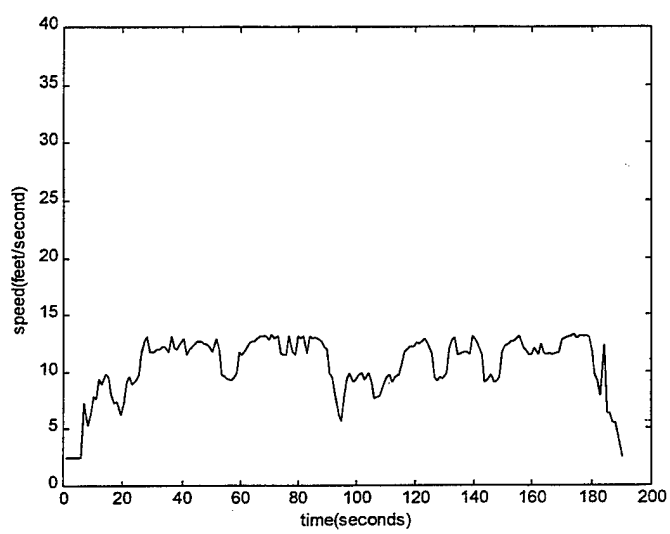

Figure 4.15 Speed Data After Filtering

After compensating heading-dependent compass error and fixing the problems with speed sensor, other ground vehicle tests were performed. These results are depicted in Figures 4.16 and 4.17. They were taken at the Navy Golf Course parking lot. The plus signs represent the DGPS measurements of the vehicle trajectory. The solid line is the trajectory of the vehicle following the same path computed by the SANS III filter without using DGPS data during the entire run. Taking the DGPS data as the reference, it is seen that the result of the SANS III without DGPS is accurate to within 15 feet throughout the run.

D. OBSERVATIONS

The following observations were made while conducting these tests. It is necessary to receive a proper DGPS signal to run the SANS software. If a valid differential correction is received about every five seconds, the SANS program will be able to be run in DGPS mode. It took a considerably long time to get a valid DGPS signal for ground vehicle testing. These are several things that could be the reason for not getting a proper signal.

One of the reasons could be the differential antenna. It was replaced with a more powerful one to eliminate this possibility.

The signal itself could have been the cause the problem. Two ground stations in Monterey Bay area are transmitting DGPS signals for our specially designed unit at a certain frequency. If the testing area is open to both stations, the differential unit sometimes receives both signals back to back which causes the program not to run in DGPS mode. The signal was tested with a hand receiver, and it was hypothesized that there is a synchronization problem between the ground stations. Most of the time, the

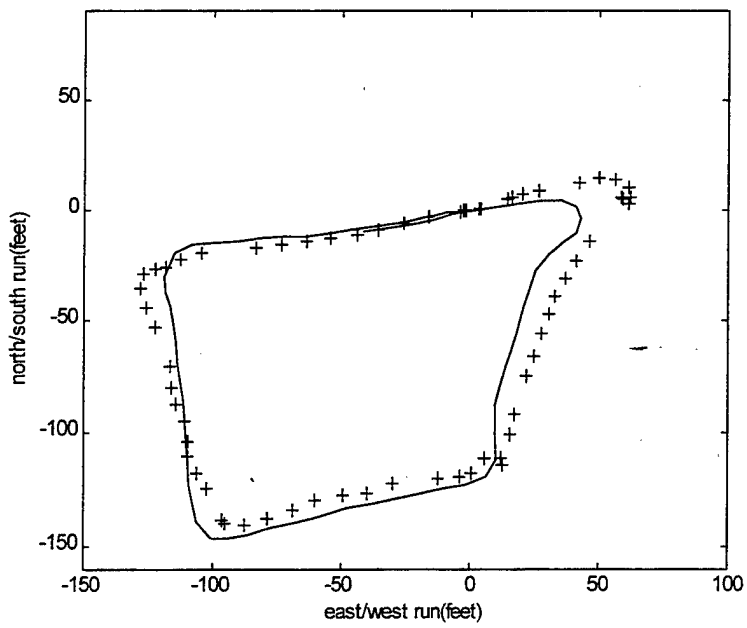

Figure 4.16 Continuous DGPS vs. without DGPS Track (without compass correction)

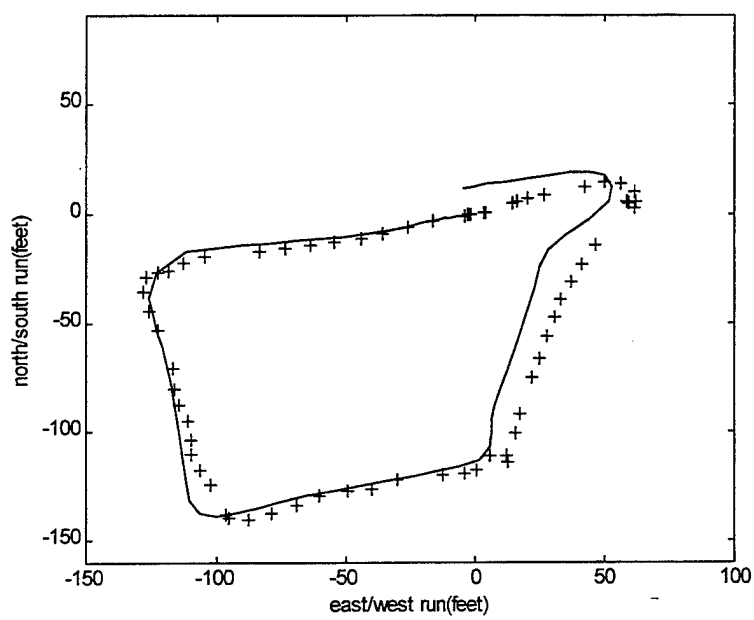

Figure 4.17 Continuous DGPS vs. without DGPS Track (with compass correction)

signal was reasonably good and strong enough to use.

Eliminating these two possible problems didn't solve the problem with the DGPS signal. It turned out that when the computer of SANS was turned on, the DGPS signal went away. The SANS computer interfered with the DGPS signal. Everything on the golf cart was grounded to eliminate the noise causing interference and a metal shield for the SANS computer was fabricated to eliminate any frequency related interference. This helped, but didn't solve the problem completely.

E. CONCLUSIONS AND SUMMARY

The purpose of this chapter was to present the latest testing results of SANS III and explain the calibration procedures for the current hardware components. These results showed that the Crossbow IMU had an accuracy of one degree with proper gains and scale factors, and the TCM2, after heading dependent error correction, could supply information with an accuracy of one degree.

The simulation and the hardware bench test results demonstrated that the SANS Kalman filter was working properly. The ground vehicle tests proved that the overall SANS III was able to navigate within ± 15 feet of Global Positioning track with no Global Positioning update for three minutes.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

A. SUMMARY

The purpose of this thesis was to improve the navigation accuracy of the latest version of SANS by properly tuning Kalman filter gains. It also presents the test and calibration results.

The research objectives of this thesis were: (1) to implement the asynchronous Kalman filter within the SANS system, (2) to tune filter gains by comparing the simulation and real hardware bench test results and, (3) to calibrate the components and test the overall system.

The objective of the SANS system is to demonstrate the feasibility of using low-cost, small components to navigate inertially between DGPS fixes. To achieve the first research objective, the previous SANS constant-gain filter was replaced by an asynchronous Kalman filter, which has six states for orientation estimation (still constant gain), and eight states for position estimation. The asynchronous nature of DGPS measurements due to asynchronous reacquisition time of satellite signals and asynchronous submergence and surfacing of the AUV, made the selection of an asynchronous Kalman filter algorithm a logical choice. The results in Chapter 4 confirmed that using a Kalman filter improved the accuracy of the system significantly.

Bench test results indicated that the newly designed system provided a higher level of performance than the previous versions of SANS. Examination of the experimental data indicated that the new IMU used in this research was capable of meeting all SANS requirements. The new data acquisition and processing unit increased

the speed, reliability, and compatibility of the system. Testing the new asynchronous Kalman filter with different speed and heading data indicated that the new navigation filter worked properly.

Ground vehicle tests demonstrated that the SANS III was able to navigate with an accuracy of ± 15 feet. The current effort was directed toward at-sea testing.

B. FUTURE RESEARCH

The current hardware components seem to be working at a very reasonable sampling rate, but technology advances, software development, and the amount of research put into test and evaluation showed that the future of SANS will be subject to many changes. New SANS components should be chosen to reduce the size, while improving performance and decreasing cost.

For the near future, the system is ready for at-sea testing. In order to acquire more accurate navigation information, the interference between the SANS computer and the DGPS unit should be solved. For future versions, it is suggested that the DGPS unit should be replaced with a new one which utilizes the broadcast Coast Guard signal.

The performance of the new water speed sensor should be tested. The accuracy of the sensor seems acceptable for the SANS objective. It might need calibration prior to at-sea testing.

The asynchronous Kalman filter has been implemented as a navigation filter. The filter gains were adjusted during the test and calibration period. While conducting at-sea testing, the filter constants and gains might need to be adjusted further.

During the ground vehicle testing, it appeared that magnetic compasses are very sensitive to environmental changes and hard to work with. Today's laser gyros are getting smaller and cheaper. They are more robust than a magnetic compass. These should be considered for the future versions of SANS.

Adding a local area network card to the system would improve SANS portability and could potentially change the way the sensors are integrated and utilized for applications other than AUVs.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. MODIFIED PART OF NAVIGATION CODE(C++)

A. INS.H

```
#ifndef _INS_H
#define _INS_H
#include <time.h>
#include <math.h>
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <fstream.h>
#include <iostream.h>
#include <assert.h>

#include "toetypes.h"
#include "globals.h"
#include "sampler.h"
#include "MatrixCls.h"

/*****************************************************************
   CLASS:          insClass
   AUTHOR:         Eric Bachmann, Dave Gay, Kadir Akyol, Suat Arslan
   DATE:           11 July 1995 last modified March 2000
   FUNCTION:  Takes in linear accelerations, angular rates, speed and
    heading information and uses Kalman filtering techniques to return a
    dead reconing position.
*****************************************************************/

class insClass {

   public:

      insClass();          // Constructor, initializes gains
      ~insClass() {}       // destructor

      Boolean insPosition(stampedSample&); // returns ins estimated
position

      // Updates the x, y and z of the vehicle posture
      void correctPosition(stampedSample&, double);

      // Sets posture to the origin and develops initial biases
      void insSetUp(double, stampedSample&);

   private:

      // file for filter data
      ofstream kalmanData;

      Matrix h, h_transpose, p, p_minus, r1, k1, k,  x_hatMinus, x_hat,
           z, i, phi,phi_transpose, q, h1, h1_transpose, k2, r2, k3, z3,
           zMat;
```

```
// ins estimated posture (x y z phi theta psi)
float posture[6];
// ins estimated linear and angular velocities
double velocities[6];

// time of last gps position fix
float lastGPStime;
  // Accumulates deltaT between GPS data
double gpstimeCounter;
  //accumulates deltaT
double cumtime;

// filter time constant
int tau;

samplerClass sam1;   // sampler instance

matrix rotationMatrix; // body to euler transformation matrix

float current[3];    // ins estimated error current

// Software bias corrections for IMU rate sensors
double biasCorrection[3];

// Complementary filter gains for orientation.
float Kone1, Kone2, Ktwo, speed;

// Kalman Filter constants
double tau_1, tau_2, tau_3, D1, D2, D3;

// Transforms body coords to earth coords, removes gravity comp.
void transformAccels (double[]);

// Transforms water speed reading to x and y components
void transformWaterSpeed (double, double[]);

// Tranforms body euler rates to earth euler rates.
void transformBodyRates (double[]);

// Euler integrates the accelerations and updates the velocities
void updateVelocities (stampedSample&);

// Euler integrates the velocities and updates the posture
void updatePosture (stampedSample&);

// Builds the body to euler rate matrix
matrix buildBodyRateMatrix();

// Builds the body to earth rotation matrix
void buildRotationMatrix();

// Calculates the imu bias correction during set up
void calculateBiasCorrections(stampedSample&);
```

```
        // Applies bias corrections to a sample
        void applyBiasCorrections(stampedSample&);

        // Reads filter constants from 'ins.cfg'
        void readInsConfigFile();

        //constructs h(4*8) matrix
        void constructHmatrix();

        //constructs P_minus(8*8) matrix
        void constructPminusMatrix();

        //constructs r(4*4) matrix
        void constructR1matrix();

        //constructs h(2*8) matrix (h matrix without GPS)
        void constructH1matrix();

        //constructs r(2*2) matrix (r matrix without GPS)
        void constructR2matrix();

        //constructs phi(8*8) matrix
        void constructPhiMatrix(stampedSample&);

        //constructs q(8*8) matrix
        void constructqMatrix(stampedSample&);
};
// Post multiply a matrix times a vector and return result.
vector operator* (matrix&, double[]);

#endif
```

B. INS.CPP

```
#include <iostream.h>
#include <signal.h>
#include <assert.h>
#include <math.h>

#include "ins.h"
#include "matrix.h"

#define SIGFPE 8            // Floating point exception

/*****************************************************************
   PROGRAM:  insClass (constructor)
   AUTHOR:   Eric Bachmann,Dave Gay,Rick Roberts, Kadir Akyol,
             Suat Arslan
   DATE:     11 July 1995 last modified March 2000
   FUNCTION: Constructor initializes kalman filter gains and  linear
    and angular velocities
    RETURNS:  nothing
   CALLED BY: navigator class
   CALLS:     none
*****************************************************************/
#ifndef _NO_NAMESPACE
using namespace std;
using namespace math;
#define STD std
#else
#define STD
#endif

#ifndef _NO_TEMPLATE
typedef matrix<double> Matrix;
#else
typedef matrix Matrix;
#endif

#ifndef _NO_EXCEPTION
#  define TRYBEGIN()     try {
#  define CATCHERROR()   } catch (const STD::exception e) { \
                                  cerr << "Error: " << e.what() <<
endl; }
#else
#  define TRYBEGIN()
#  define CATCHERROR()
#endif
void myInverse(MatrixCls &);

insClass::insClass():h("h matrix",4,8),h_transpose("h transpose", 8,4),
            p_minus("p minus",8,8),r1("r1 matrix",4,4), k1("k1", 4, 4),
            k("k matrix", 8, 4), x_hatMinus("x_hatmin", 8, 1),
            x_hat("x hat", 8,1), i("unit mat", 8, 8),
            phi_transpose("phitranspose", 8, 8), h1("h1",2,8),
            h1_transpose("h1 transpose", 8, 2), r2("r2 matrix",2,2),
            k2("k2", 2, 2), k3("k mat no  gps", 8, 2),
```

```
            phi("phi matrix", 8, 8), q("q matrix", 8, 8),
            p("p matrix", 8, 8),z3("z3 matrix",2,1),
            zMat("zMat",4,1),kalmanData("xhat.dat"),gpstimeCounter(1.0)
{
   cerr << "\nconstructing ins1" << endl;

   readInsConfigFile();            // Read the config file

   constructHmatrix();             //constructs 4*8 h matrix

   constructPminusMatrix();        //constructs 8*8 P_minus matrix

   constructR1matrix();                //constructs 4*4 R1 matrix

   constructH1matrix();                      //constructs 2*8 h matrix

   constructR2matrix();                //constructs 2*2 R2 matrix

   velocities[0] = 0.0;             // x dot
   velocities[1] = 0.0;             // y dot
   velocities[2] = 0.0;             // z dot
   velocities[3] = 0.0;             // phi dot
   velocities[4] = 0.0;             // theta dot
   velocities[5] = 0.0;             // psi dot

   // Set posture to straight and level at the origin
   posture[0] = 0.0;    // x
   posture[1] = 0.0;    // y
   posture[2] = 0.0;    // z
   posture[3] = 0.0;    // phi
   posture[4] = 0.0;    // theta
   posture[5] = 0.0;    // psi

   cerr << "\nins construction complete" << endl;
}

/*****************************************************************
   PROGRAM: insPosit
   AUTHOR:  Eric Bachmann, Dave Gay, Kadir Akyol, Suat Arslan
   DATE:    11 July 1995 last modified March 2000
    FUNCTION:Make dead reckoning position estimation using kalman
    filtering.Inputs are linear accelerations, angular rates, speed and
    heading.Primary input data is obtained from a sampler object via the
    getSample method. This data is stored in the sample filed of a
    stampedSample structure called newSample. The sample field is then
    used as a working variable as the linear accelerations and angular
    rates it contains are converted to earth coordinates and integrated
    to determine current velocities and posture. The data is
    asynchronous kalman filtered against itself, speed and magnetic
    heading.
    RETURNS:      position in grid coordinates as estimated by the INS
   CALLED BY:     navPosit (nav.cpp)
   CALLS:         getSample (sampler.cpp)
                  findDeltaT (ins.cpp)
                              transformBodyRates (ins.cpp)
```

```
                    buildRotationMatrix (ins.cpp)
                    transformAccels (ins)
                    transformWaterSpeed (ins)
*****************************************************************/

void fpeInsPosit(int sig)
{if (sig == SIGFPE) cerr << "floating point error in insPosit\n";}

Boolean insClass::insPosition(stampedSample& newSample)
{
   signal (SIGFPE, fpeInsPosit);
    // Working variables
   double thetaA, phiA, xIncline, yIncline;

   if (sam1.getSample(newSample)) {

      applyBiasCorrections(newSample);

        newSample.rawSample[0] = newSample.sample[0];
        newSample.rawSample[1] = newSample.sample[1];
        newSample.rawSample[2] = newSample.sample[2];
        newSample.rawSample[3] = newSample.sample[3];
        newSample.rawSample[4] = newSample.sample[4];
        newSample.rawSample[5] = newSample.sample[5];
        newSample.rawSample[6] = newSample.sample[6];
        newSample.rawSample[7] = newSample.sample[7];

        gpsTimeCounter+=newSample.deltaT;
        cumtime+=newSample.deltaT;

        xIncline = newSample.sample[0] / GRAVITY;
        yIncline = (newSample.sample[1] -
                  (newSample.sample[5] * newSample.sample[6]))
                  / (GRAVITY * cos(posture[4]));

        if (fabs(yIncline) > 1.0 || fabs(xincline)>1.0) {
              static int inclineCount(0);
              gotoxy(1,24);
          cerr << "Inclination errors: " << ++inclineCount << endl;
        return FALSE;
        }

      // Calculate low freq pitch and roll
        thetaA = asin(xIncline);
        phiA = -asin(yIncline);

        // Transform body rates to euler rates.
        transformBodyRates(newSample.sample);

        // Calculate estimated roll rate (phi-dot).
        velocities[3] = newSample.sample[3] + Kone1 * (phiA -
posture[3]);
        // Calculate estimated pitch rate (theta-dot).
        velocities[4] = newSample.sample[4] + Kone2 * (thetaA-
posture[4]);
```

```
        // Calculate estimated heading rate (psi-dot).
        velocities[5] =
        newSample.sample[5] + Ktwo * (newSample.sample[7] -
posture[5]);

        // integrate estimated angular rates to obtain angles
      // pitch rate to angle
      posture[3] += newSample.deltaT * velocities[3];
      // roll rate to angle
      posture[4] += newSample.deltaT * velocities[4];
        // yaw rate to angle
      posture[5] += newSample.deltaT * velocities[5];

       // constructs Phi matrix
     constructPhiMatrix(newSample);

     //constructs Q matrix (8*8)
     constructqMatrix(newSample);

     //calculate x_hatMinus
     x_hatMinus = ( phi * x_hat );

     //calculate phi_transpose
     phi.transpose(phi_transpose);

     //calculate P_minus
     p_minus = ((( phi * p ) * phi_transpose ) +  q );

     // Beginning Kalman Filter loops
     if (newSample.gpsFlag && gpsTimeCounter>=1.0) {

        zMat.copy(0,0,(newSample.sample[6] * cos (posture[5])));
        zMat.copy(1,0,(newSample.sample[6] * sin (posture[5])));
        zMat.copy(2,0,newSample.est.x);
        zMat.copy(3,0,newSample.est.y);

        //transpose of matrix h
        h.transpose(h_transpose);
        k1 = (((h*p_minus)*h_transpose)+r1);

        //take inverse of matrix k1
        myinverse(k1);

        //calculate matrix Kalman gain
        k = ((p_minus * h_transpose)* k1);

        //calculate x_hat
        x_hat = ( x_hatMinus + (k * (zMat - (h * x_hatMinus))));

        //calculate I matrix
        i = i.unitMatrix (8);

        //calculate P matrix
        p = ((i - (k * h)) * p_minus);
```

```
        // Writing filter output to file
        kalmanData  << cumtime << ' '
                    << newSample.gpsFlag << ' '
                    << x_hat.getElement(0,0) << ' '
                              << x_hat.getElement(1,0) << ' '
                          << x_hat.getElement(2,0) << ' '
                          << x_hat.getElement(3,0) << ' '
                          << x_hat.getElement(4,0) << ' '
                    << x_hat.getElement(5,0) << ' '
                              << x_hat.getElement(6,0) << ' '
                          << x_hat.getElement(7,0)
                    << endl;
        newSample.gpsFlag = FALSE;
        gpsTimeCounter = 0.0;

    }
    else {
            z3.copy(0,0, (newSample.sample[6] * cos (posture[5])));
        z3.copy(1,0, (newSample.sample[6] *      sin (posture[5])));

        //h1 is the h matrix without GPS
        h1.transpose(h1_transpose);

        k2 = (((h1*p_minus)*h1_transpose)+r2);

        // take inverse of matrix k2
        myinverse(k2);

        //kalman gain matrix without gps
        k3 = ((p_minus * h1_transpose)* k2);
        x_hat = ( x_hatMinus + (k3 * (z3 - (h1 * x_hatMinus))));

        //calculate I matrix
        i = i.unitMatrix (8);

        //calculate P matrix
        p = ((i - (k3 * h1)) * p_minus);    }

        // Writing filter output to file
        kalmanData  << cumtime << ' '
                    << newSample.gpsFlag << ' '
                    << x_hat.getElement(0,0) << ' '
                              << x_hat.getElement(1,0) << ' '
                          << x_hat.getElement(2,0) << ' '
                          << x_hat.getElement(3,0) << ' '
                          << x_hat.getElement(4,0) << ' '
                    << x_hat.getElement(5,0) << ' '
                              << x_hat.getElement(6,0) << ' '
                          << x_hat.getElement(7,0)
                    << endl;
    }

    // estimated north and east positions
    posture[0] = x_hat.getElement(6,0);
    posture[1] = x_hat.getElement(7,0);
```

```
    // estimated current values
    newSample.sample[0] = posture[0] ;
    newSample.sample[1] = posture[1] ;
    newSample.sample[2] = posture[2] ;
    newSample.sample[3] = posture[3];
    newSample.sample[4] = posture[4];
    newSample.sample[5] = posture[5];
    return TRUE;
  }
  else {
    return FALSE;  // New IMU information was unavailable.
}
}
/*****************************************************************
    PROGRAM:        insSetUp
    AUTHOR:       Eric Bachmann, Dave Gay
    DATE:           11 July 1995
    FUNCTION:             Initializes the INS system. Sets the posture to
     the origin.Initializes the heading using magnetic compass
     information. Initializes the last GPS fix and last IMU information
     times.
    RETURNS:        void
    CALLED BY:      initializeNavigator (nav)
    CALLS:          calculateBiasCorrections (ins)
                              getSample (sampler)
                              buildRotationMatrix (ins)
                              transformWaterSpeed (ins)
*****************************************************************/

void fpeInsSetUp(int sig)
{if (sig == SIGFPE) cerr << "floating point error in inSetUp\n";}

void insClass::insSetUp(double originTime, stampedSample& posit)
{
    cerr << "   Initializing INS." << endl;
    signal (SIGFPE, fpeInsSetUp);

    saml.initSampler();        // Initialize the sampler
    saml.getSample(posit);

    cerr << "   X accel = " << posit.sample[0]
           << ", Y accel = " << posit.sample[1]
           << ", Z accel = " << posit.sample[2] << endl;

    calculateBiasCorrections(posit);     // set imu biases

    posture[5] = posit.sample[7]; //set initial true heading

    buildRotationMatrix();               //set initial speed
    transformWaterSpeed(posit.sample[6], velocities);

    posit.current[0] = current[0];
    posit.current[1] = current[1];
    posit.current[2] = current[2];
```

```
   lastGPStime = originTime;            // initialize times

   cerr << "   INS initialization complete." << endl;
}

/*********************************************************
   PROGRAM: transformAccels
   AUTHOR:        Eric Bachmann, Dave Gay
   DATE:      11 July 1995
   FUNCTION:      Transforms linear accelerations from body coordinates
    to earth coordinates and removes the gravity component in the z
    direction.
   RETURNS:   void
   CALLED BY:     navPosit
   CALLS:     none
*********************************************************/
void insClass::transformAccels (double newSample[])
{
   vector earthAccels;

   newSample[0] -= GRAVITY * sin(posture[4]);
   newSample[1] += GRAVITY * sin(posture[3]) * cos(posture[4]);
   newSample[2] += GRAVITY * cos(posture[3]) * cos(posture[4]);

   earthAccels = rotationMatrix * newSample;

   newSample[0] = earthAccels.element[0];
   newSample[1] = earthAccels.element[1];
   newSample[2] = earthAccels.element[2];
}

/*****************************************************************
   PROGRAM:     transformWaterSpeed
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:    Transforms water speed into a vector in earth
   coordinates and returns them in the speedCorrection variable.
   RETURNS:     void
   CALLED BY:   navPosit
   CALLS:       none
*****************************************************************/

void insClass::transformWaterSpeed (double waterSpeed, double
speedCorrection[])
{
   double water[3] = {waterSpeed, 0.0, 0.0};
   vector waterVelocities = rotationMatrix * water;

   speedCorrection [0] = waterVelocities.element[0];
   speedCorrection [1] = waterVelocities.element[1];
   speedCorrection [2] = waterVelocities.element[2];
}

/*****************************************************************
```

```
   PROGRAM:     transformBodyRates
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
   FUNCTION:    Tranforms body euler rates to earth euler rates
   RETURNS:     none
   CALLED BY:   insPosit
   CALLS:       buildBodyRateMatrix
*****************************************************************/

void insClass::transformBodyRates (double newSample[])
{
   matrix bodyRateMatrix = buildBodyRateMatrix( );
   vector earthRates = bodyRateMatrix * &(newSample[3]);

   newSample[3] = earthRates.element[0];
   newSample[4] = earthRates.element[1];
   newSample[5] = earthRates.element[2];
}

/*****************************************************************
   PROGRAM:    buildBodyRateMatrix
   AUTHOR:     Eric Bachmann, Dave Gay
   DATE:       11 July 1995
   FUNCTION:   Builds body to Euler rate translation matrix.
   RETURNS:    rate translation matrix
   CALLED BY:  insPosit
   CALLS:      none
*****************************************************************/

matrix insClass::buildBodyRateMatrix()
{
   matrix rateTrans;

   float    tth = tan(posture[4]),
           sphi = sin(posture[3]),
                  cphi = cos(posture[3]),
                  cth = cos(posture[4]);

   rateTrans.element[0][0] = 1.0;
   rateTrans.element[0][1] = tth * sphi;
   rateTrans.element[0][2] = tth * cphi;
   rateTrans.element[1][0] = 0.0;
   rateTrans.element[1][1] = cphi;
   rateTrans.element[1][2] = -sphi;
   rateTrans.element[2][0] = 0.0;
   rateTrans.element[2][1] = sphi / cth;
   rateTrans.element[2][2] = cphi / cth;

   return rateTrans;
}

/*****************************************************************
   PROGRAM:     buildRotationMatrix
   AUTHOR:      Eric Bachmann, Dave Gay
   DATE:        11 July 1995
```

```
   FUNCTION:        Sets the body to earth coordinate rotation matrix.
   RETURNS:     void
   CALLED BY:       insPosit, insSetUp
   CALLS:       none
*****************************************************************/

void insClass::buildRotationMatrix()
{
   float spsi = sin(posture[5]),
         cpsi = cos(posture[5]),
         sth = sin(posture[4]),
         sphi = sin(posture[3]),
         cphi = cos(posture[3]),
         cth = cos(posture[4]);

   rotationMatrix.element[0][0] = cpsi * cth;
   rotationMatrix.element[0][1] = (cpsi * sth * sphi) - (spsi * cphi);
   rotationMatrix.element[0][2] = (cpsi * sth * cphi) + (spsi * sphi);
   rotationMatrix.element[1][0] = spsi * cth;
   rotationMatrix.element[1][1] = (cpsi * cphi) + (spsi * sth * sphi);
   rotationMatrix.element[1][2] = (spsi * sth * cphi) - (cpsi * sphi);
   rotationMatrix.element[2][0] = -sth;
   rotationMatrix.element[2][1] = cth * sphi;
   rotationMatrix.element[2][2] = cth * cphi;
}

/*****************************************************************
   PROGRAM:    postmultiplication operator *
   AUTHOR:   Eric Bachmann, Dave Gay
   DATE:     11 July 1995
   FUNCTION:   Post multiply a 3 X 3 matrix times a 3 X 1 vector and
   return the result
   RETURNS:  3 X 1 vector
   CALLED BY:
   CALLS:    None1
*****************************************************************/

vector operator* (matrix& transform, double state[])
{
   vector result;

   for (int i = 0; i < 3; i++) {

      result.element[i] = 0.0;

      for (int j = 0; j < 3; j++) {

            result.element[i] += transform.element[i][j] * state[j];
      }
   }
   return result;
}

/*****************************************************************
   PROGRAM:      calculateBiasCorrections
```

```
    AUTHOR:         Eric Bachmann, Dave Gay, Rick Roberts
    DATE:           11 July 1995
    FUNCTION:       Calculates the initial imu bias by averaging a number
    of imu readings.
    RETURNS:        none
    CALLED BY:      insSetup
    CALLS:          none
*****************************************************************/

void fpeCalculateBiasCorrections(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
CalculateBiasCorrections\n";}

void insClass::calculateBiasCorrections(stampedSample& biasSample)
{
   signal (SIGFPE, fpeCalculateBiasCorrections);

   int biasNumber(tau/10);

   biasCorrection[0] = 0.0;          // p roll rate
   biasCorrection[1] = 0.0;          // q pitch rate
   biasCorrection[2] = 0.0;          // r yaw rate

   for (int i = 0; i < biasNumber; i++) {

      // Find the average of the biasNumber packets
      while(!sam1.getSample(biasSample)) {/* */};

      // roll-rate/b#
      biasCorrection[0] += biasSample.sample[3]/biasNumber;
      // pitch-rate/b#
      biasCorrection[1] += biasSample.sample[4]/biasNumber;
      // yaw-rate/b#
      biasCorrection[2] += biasSample.sample[5]/biasNumber;
   }

   // set biasSample correction fields to new bias correction values
   // negative biasCorrection value is taken so biases are added to
   // sensor values
   biasSample.bias[0] = biasCorrection[0] = -(biasCorrection[0]);
   biasSample.bias[1] = biasCorrection[1] = -(biasCorrection[1]);
   biasSample.bias[2] = biasCorrection[2] = -(biasCorrection[2]);
}

/*****************************************************************
   PROGRAM:     applyBiasCorrections
   AUTHOR:      Eric Bachmann, Dave Gay, Rick Roberts
   DATE:        11 July 1995
   FUNCTION:    Applies updated bias corrections to a sample.
   RETURNS:     void
   CALLED BY:   insPosit
   CALLS:       none
*****************************************************************/

void insClass::applyBiasCorrections(stampedSample& posit)
```

```
{
   const float sampleWght(posit.deltaT/tau);
   const float biasWght(1 - sampleWght);

   //Calculate updated bias values
   biasCorrection[0] = (biasWght * biasCorrection[0])
                              - (sampleWght * posit.sample[3]);
   biasCorrection[1] = (biasWght * biasCorrection[1])
                              - (sampleWght * posit.sample[4]);
   biasCorrection[2] = (biasWght * biasCorrection[2])
                              - (sampleWght * posit.sample[5]);

   //Apply the bias to the sample
   posit.sample[3] += biasCorrection[0];
   posit.sample[4] += biasCorrection[1];
   posit.sample[5] += biasCorrection[2];

   //Save the bias to the sample
   posit.bias[0] = biasCorrection[0];
   posit.bias[1] = biasCorrection[1];
   posit.bias[2] = biasCorrection[2];
}


/*****************************************************************
   PROGRAM:  readInsConfigFile
   AUTHOR:   Rick Roberts, Eric Bachmann, Suat Arslan
   DATE:     02 Nov 96 last modified march 2000
   FUNCTION: Reads filter constants from 'ins.cfg'
   RETURNS:  void
   CALLED BY:ins class constructor
   CALLS:    none
*****************************************************************/

void insClass::readInsConfigFile()
{
   cerr << "Reading ins configuration file." << endl;
   ifstream insCfgFile("ins.cfg", ios::in);

   if(!insCfgFile) {
      cerr << "could not open ins configuration file!" << endl;
   }
   else {
      char comment[128];
      insCfgFile
      >> Kone1 >> comment
            >> Kone2 >> comment
            >> Ktwo >> comment
            >> tau_1 >> comment
            >> tau_2 >> comment
            >> tau_3 >> comment
            >> D1 >> comment
            >> D2 >> comment
            >> D3 >> comment
```

```
            >> tau >> comment
            >> speed >> comment
            >> current[0] >> comment
            >> current[1] >> comment
            >> current[2] >> comment;

      cout << "\nKone1: " << Kone1 << "\nKone2: " << Kone2
           << "\nKtwo:  " << Ktwo << "\ntau_1: " << tau_1
           << "\ntau_2: " << tau_2 << "\ntau_3: " << tau_3
           << "\nD1: " << D1 << "\nD2: " << D2
           << "\nD3: " << D3 << "\ntau: " << tau
           << "\nx Current: " << current[0] << "\ny Current: "
           << current[1] << "\nz Current: "<< current[2] << endl;
   }

   insCfgFile.close( );
}

/*****************************************************************
   PROGRAM:  constructHmatrix()
   AUTHOR:   Kadir Akyol
   DATE:     01 March 1999
   FUNCTION: constructs h matrix
   RETURNS:  none
   CALLED BY:ins class constructor
   CALLS:    none
*****************************************************************/



void fpeconstructHmatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructHmatrix\n";}

void insClass::constructHmatrix()
{
   signal (SIGFPE, fpeconstructHmatrix);

   h.copy(0,0,1.0);
   h.copy(1,1,1.0);
   h.copy(2,4,1.0);
   h.copy(2,6,1.0);
   h.copy(3,5,1.0);
   h.copy(3,7,1.0);

   return ;
}//end constructHmatrix()

/*****************************************************************
   PROGRAM: constructPminusMatrix()
   AUTHOR:    Kadir Akyol, Suat Arslan
   DATE:      01 March 1999 last modified march 2000
   FUNCTION:  constructs P_minus matrix
   RETURNS:   none
   CALLED BY: ins class constructor
```

```
   CALLS:     none
*****************************************************************/

void fpeconstructPminusMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructPminusMatrix\n";}

void insClass::constructPminusMatrix()
{
   signal (SIGFPE, fpeconstructPminusMatrix);

   p_minus.copy(0,0,0.1);
   p_minus.copy(1,1,0.1);
   p_minus.copy(2,2,0.1);
   p_minus.copy(3,3,0.1);
   p_minus.copy(4,4,0.1);
   p_minus.copy(5,5,0.1);
   p_minus.copy(6,6,0.1);
   p_minus.copy(7,7,0.1);

   return ;
}//end constructPminusMatrix()

/*****************************************************************
   PROGRAM: constructPMatrix()
   AUTHOR:    Suat ARSLAN
   DATE:      28 May 1999
   FUNCTION:  constructs P matrix
   RETURNS:   none
   CALLED BY: ins
   CALLS:     None
*******************************************************************/

void fpeconstructPMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructPMatrix\n";}

void insClass::constructPMatrix()
{
      signal (SIGFPE, fpeconstructPMatrix);

      p.copy(0,0,0.5);
   p.copy(1,1,0.5);
   p.copy(2,2,1.0);
   p.copy(3,3,1.0);
   p.copy(4,4,10.0);
   p.copy(5,5,10.0);
   p.copy(6,6,15.0);
   p.copy(7,7,15.0);

   return ;

}//end constructPMatrix()

/*****************************************************************
```

```
    PROGRAM:        constructR1matrix()
    AUTHOR:         Kadir Akyol, Suat Arslan
    DATE:           01 March 1999 last modified march 2000
    FUNCTION:       constructs r1 matrix
    RETURNS:        none
    CALLED BY:      ins class constructor
    CALLS:     none
 ****************************************************************/

 void fpeconstructR1matrix(int sig)
 {if (sig == SIGFPE) cerr << "floating point error in
 constructR1matrix\n";}

 void insClass::constructR1matrix()
 {
    signal (SIGFPE, fpeconstructR1matrix);

    r1.copy(0,0,0.01);
    r1.copy(1,1,0.01);
    return ;
 }//end constructR1Matrix()

 /****************************************************************
    PROGRAM: constructH1matrix()
    AUTHOR:     Kadir Akyol
    DATE:       01 March 1999
    FUNCTION:   constructs h matrix
    RETURNS:    none
    CALLED BY:  ins class constructor
    CALLS:      None
 ****************************************************************/

 void fpeconstructH1matrix(int sig)
 {if (sig == SIGFPE) cerr << "floating point error in
 constructH1matrix\n";}

 void insClass::constructH1matrix()
 {
    signal (SIGFPE, fpeconstructH1matrix);

    h1.copy(0,0,1.0);
    h1.copy(1,1,1.0);
    return ;
 }//end constructH1matrix()

 /****************************************************************
    PROGRAM:        constructR2matrix()
    AUTHOR:     Kadir Akyol, Suat Arslan
    DATE:       01 March 1999 last modified march 2000
    FUNCTION:   constructs r2 matrix
    RETURNS:    none
    CALLED BY:  ins class constructor
    CALLS:      None
 ****************************************************************/
```

```
void fpeconstructR2matrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
constructR2matrix\n";}

void insClass::constructR2matrix()
{
   signal (SIGFPE, fpeconstructR2matrix);

   r2.copy(0,0,0.01);
   r2.copy(0,1,0.0);
   r2.copy(1,0,0.0);
   r2.copy(1,1,0.01);

   return ;

}//end constructR2matrix()

/*****************************************************************
   PROGRAM:    constructPhiMatrix()
   AUTHOR:     Kadir Akyol, Suat Arslan
   DATE:       01 March 1999 last modified march 2000
   FUNCTION:   constructs phi matrix
   RETURNS:    none
   CALLED BY:  insPosit
   CALLS:      None
*****************************************************************/

void fpeconstructPhiMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
cunstructPhiMatrix\n";}

void insClass::constructPhiMatrix(stampedSample& delta)
{
   signal (SIGFPE, fpeconstructPhiMatrix);

   double xx, yy, zz;

   xx = - (delta.deltaT)/tau_1;
   xx = exp(xx);
   yy = - (delta.deltaT)/tau_2;
   yy = exp(yy);
   zz = - (delta.deltaT)/tau_3;
   zz = exp(zz);

   phi.copy(0,0,xx);
   phi.copy(1,1,xx);
   phi.copy(2,2,yy);
   phi.copy(3,3,yy);
   phi.copy(4,4,zz);
   phi.copy(5,5,zz);
   phi.copy(6,0,((1-xx)*tau_1));
   phi.copy(6,2,((1-yy)*tau_2));
   phi.copy(7,1,((1-xx)*tau_1));
   phi.copy(7,3,((1-yy)*tau_2));
```

```
    return ;

}//end constructPhiMatrix()

/*****************************************************************
    PROGRAM:        constructqMatrix()
    AUTHOR:         Kadir Akyol, Suat Arslan
    DATE:           01 March 1999
    FUNCTION:       constructs Q matrix
    RETURNS:        none
    CALLED BY: insPosit
    CALLS:      None
*****************************************************************/

void fpeconstructqMatrix(int sig)
{if (sig == SIGFPE) cerr << "floating point error in
cunstructqMatrix\n";}

void insClass::constructqMatrix(stampedSample& delt)
{
   signal (SIGFPE, fpeconstructqMatrix);

   double uu, ww, vv,zz,yy,Q661,Q662,Q60,Q62,Q71,Q73,Q66;

   uu =  -(2.0 * delt.deltaT)/tau_1;
   uu = exp(uu);
   ww =  -(2.0 * delt.deltaT)/tau_2;
   ww = exp(ww);
   vv =  -(2.0 * delt.deltaT)/tau_3;
   vv = exp(vv);
   zz = -(delt.deltaT)/tau_1;
   zz = exp(zz);
   yy = -(delt.deltaT)/tau_2;
   yy = exp(yy);

   Q661=D1*((tau_1/2)*(1-uu)-((2*tau_1)*(1-zz))+delt.deltaT);
   Q662=D2*((tau_2/2)*(1-ww)-((2*tau_2)*(1-yy))+(delt.deltaT));
   Q66=Q661+Q662;

   Q60=D1*(0.5+(0.5*uu)-zz);
   Q62=D2*(0.5+(0.5*ww)-yy);
   Q71=Q60;
   Q73=Q62;

   q.copy(0,0,((uu)*(D1/(2.0*tau_1))));
   q.copy(1,1,((uu)*(D1/(2.0*tau_1))));
   q.copy(2,2,((ww)*(D2/(2.0*tau_1))));
   q.copy(3,3,((ww)*(D2/(2.0*tau_1))));
   q.copy(4,4,((vv)*(D3/(2.0*tau_1))));
   q.copy(5,5,((vv)*(D3/(2.0*tau_1))));
   q.copy(6,6,(Q66));
   q.copy(7,7,(Q66));
   q.copy(0,6,(Q60));
   q.copy(6,0,(Q60));
   q.copy(2,6,(Q62));
```

```
   q.copy(6,2,(Q62));
   q.copy(1,7,(Q71));
   q.copy(7,1,(Q71));
   q.copy(3,7,(Q73));
   q.copy(7,3,(Q73));

   return ;

}//end constructqMatrix()

/*****************************************************************
   PROGRAM: myInverse(MatrixCls&)
   AUTHOR:     Suat ARSLAN
   DATE:       29 july 1999
   FUNCTION:   invert the matrix
   RETURNS:    none
   CALLED BY:  ins
   CALLS:      None

*****************************************************************/
void myInverse(MatrixCls& k1) {
      Matrix k3;
      k3.SetSize(k1.getCol(), k1.getRow());
      for(int i = 0; i < k1.getCol(); i++) {
            for(int j = 0; j < k1.getCol(); j++) {
                  k3(i, j) = k1.getElement(i, j);
            }
      }
      k3 = !k3;

      for(int i = 0; i < k1.getCol(); i++) {
            for(int j = 0; j < k1.getCol(); j++) {
                  k1.copy(i, j, k3(i, j));
            }
   }
}// end of function
// end ins.cpp
```

C. ATOD.H

```
#define _ATOD_H

#include <stdio.h>
#include <fstream.h>
#include <iostream.h>
#include <dos.h>


#define ENABLED  1
#define DISABLED  0

#define INPUT  1
#define OUTPUT  0

//#define TRUE 1
//#define FALSE 0

#define STATUS_BYTE  0
#define START_CONVERSION  0
#define READ_DATA  1
#define DAC_UPDATE  1
#define CLEAR_BOARD  2
#define SCAN_BURST_SLCT 3
#define CHANNEL_SLCT  5
#define GAIN_SLCT  5
#define IRQ_ENABLE  5
#define EXT_TRIG_ENABLE  5
#define PPI_A  4
#define PPI_B  5
#define PPI_C  6
#define PPI_CTRL  7
#define TIMER_A  8
#define TIMER_B  9
#define TIMER_C  10
#define TIMER_CTRL  11
#define DAC1_LSB  12
#define DAC1_MSB  13
#define DAC2_LSB  14
#define DAC2_MSB  15
#define CLEAR_IRQ_STAT 14
#define CLEAR_DMA_DONE 15

#define PPI_PORT_A  0
#define PPI_PORT_B  1
#define PPI_PORT_C  2

#define DISABLE_BOTH 0
#define SCAN_ENABLE 1
#define BURST_ENABLE 2

#define SOURCE_AD_START 0
#define SOURCE_DMA_DONE 1
#define SOURCE_TRIGGER 2
```

```
#define SOURCE_PACER_CLOCK 3

class atodClass{

      public:

      atodClass(){};
      ~atodClass(){};

      void Initatod();

      void InitializeBoardSettings(unsigned , float );

      float DigitalToReal(int DigitalValue);

      void ResetBoard(void);

      void ClearBoard(void);

      void ClearIrqStat(void);

      void ClearDmaDone(void);

      void SetChannel(unsigned char);

      void SetGain(unsigned char);

      void SetIRQStatus(char);

      void SetExternalTrigger(unsigned char);

      void SetInterruptSource(unsigned char);

      void ScanBurstEnable(unsigned char);

      void SetScanChannels(int);

      void StartConversion(void);

      int ConversionDone();

      int ReadData(void);

      void ClockMode(unsigned char, unsigned char);

      void ClockDivisor(unsigned char, unsigned int);

      void SetUserClock(float);

      void SetPacerClock(float);

      //Boolean ClockDone(unsigned char);

      unsigned char ReadDigitalIO(unsigned char);
```

```
    void WriteDigitalIO(unsigned char, unsigned char);

    void ConfigureIOPorts(unsigned char, unsigned char);

    void UpdateDAC(unsigned char, float);

};
#endif
//end atod.h
```

D. ATOD.CPP

```
#include "atod.h"

unsigned BaseAddress;

float VoltageRange,
      DACSlope,
      ConversionFactor,
      Baseline ;

int   DACOffset;

void atodClass::Initatod()
{
      InitializeBoardSettings(768,10.0);
   ResetBoard();
   ClearBoard();
   SetChannel(1);
   SetExternalTrigger(DISABLED);
   SetGain(1);
}


/*********************************************************************

 InitializeBoardSettings

 The InitializeBoardSettings procedure is used to set the Base address
 variable and calculate the conversion factor for converting between
digital
 values and volts.  Since the base address variable is not set until
this
 procedure is called, make certain that you call this procedure before
 any others in this file.

*********************************************************************/

void atodClass::InitializeBoardSettings(unsigned BA, float Range)

{
  BaseAddress = BA;
  VoltageRange = Range;
  ConversionFactor = VoltageRange / 4096.0;
  Baseline = 0;
}

/*********************************************************************

 The Function DigitalToReal converts a digitized value to a real world
 value.  In these sample programs the conversion factor and baseline
 correspond to converting between digitized values and volts.

*********************************************************************/
```

```
float atodClass::DigitalToReal(int DigitalValue)
{
 return(DigitalValue * ConversionFactor + Baseline);
}

/*********************************************************************

 ResetBoard

 The ResetBoard procedure is used to reset the DM5406.  The 8255 PPI is
 configured so that ports A and C are input and port B is output, a
 dummy A-D conversion is performed and then the clear board command is
 sent.

*********************************************************************/

void atodClass::ResetBoard(void)

{
  outportb(BaseAddress + PPI_CTRL, 0x99);       /* Set PPI Port B for
output */
  outportb(BaseAddress + PPI_B, 0);             /* Channel 0, Gain 1,
Ext Trig =
      ..................................................................
      ..........................            Disabled, IRQ = Disabled */
  outportb(BaseAddress + SCAN_BURST_SLCT, 0);
  outportb(BaseAddress + CLEAR_BOARD, 0);

  outportb(BaseAddress + START_CONVERSION, 0); /* Start a Dummy
conversion */

  delay(5);

  outportb(BaseAddress + CLEAR_BOARD,  0);      /* Any value will do */

}

/*********************************************************************

 ClearBoard

 The clear board procedure writes to the CLEAR BOARD port on the
DM5406.

*********************************************************************/

void atodClass::ClearBoard(void)

{
  outportb(BaseAddress + CLEAR_BOARD,  0);      /* Any value will do */
}

/*********************************************************************
 ClearIrqStatus
```

The ClearIrqStat procedure reads from the CLEAR_IRQ_STAT port on the DM5406.

```
*********************************************************************/

void atodClass::ClearIrqStat(void)

{
  inportb(BaseAddress + CLEAR_IRQ_STAT);
}

/*********************************************************************

 ClearDmaDone

 The ClearDmaDone procedure reads from the CLEAR_DMA_ONE port on the
DM5406.

*********************************************************************/

void atodClass::ClearDmaDone(void)

{
  inportb(BaseAddress + CLEAR_DMA_DONE);
}

/*********************************************************************

 SetChannel

 The SetChannel procedure is used to set the channel bits, B0..B3, in
 the CHANNEL SELECT register (BA + 5).  Note how this procedure sets
only
 the channel select bits; it does not change the other bits in this
register.
 This is important because if you unintentionally clear the other bits
it
 can cause unexpected behavior of the DM5406.

*********************************************************************/

void atodClass::SetChannel(unsigned char ChannelNumber)
{
 unsigned char B;

 B = inportb(BaseAddress + CHANNEL_SLCT);  /* read current byte */
 B = B & 240;                              /* clear B0 - B3 */
 B = B | (ChannelNumber - 1);              /* set channel bits */
 outportb(BaseAddress + CHANNEL_SLCT, B);  /* write new byte */
}

/*********************************************************************

 SetGain
```

```
 The SetGain procedure is used to set the Gain bits on the DM5406.
 Note how this procedure sets only the channel select bits; it does not
 change the other bits in this register. This is important because if
 you unintentionally clear the other bits it can cause unexpected
behavior
 of the DM5406.

*********************************************************************/

void atodClass::SetGain(unsigned char Gain)
{
 unsigned char B;

 switch (Gain)
  {
   case 1 : Gain = 0; break;
   case 2 : Gain = 1; break;
   case 4 : Gain = 2; break;
   case 8 : Gain = 3; break;
   default : Gain = 0; break;
  }

 B = inportb(BaseAddress + GAIN_SLCT);    /* read current byte */
 B = B & 207;                             /* clear B4,B5*/
 B = B | (Gain * 16);                     /* set gain bits */
 outportb(BaseAddress + GAIN_SLCT, B);    /* write new byte */
}

/*********************************************************************

 SetIRQStatus

 The SetIRQStatus procedure is used to set or clear the IRQ Enable bit
 on the DM5406.  A value of 1 passed to this procedure enables
interrupts
 a value of 0 disables interrupts.

*********************************************************************/

void atodClass::SetIRQStatus(char IRQStatus)
{
 unsigned char B;

 B = inportb(BaseAddress + IRQ_ENABLE);    /* read current byte */
 B = B & 127;                             /* clear B5 */
 B = B | IRQStatus * 128;                 /* set IRQ select bit */
 outportb(BaseAddress + IRQ_ENABLE, B);    /* write new byte */
}

/*********************************************************************

 SetExternalTrigger

 The SetExternalTrigger routine is used to set the external trigger bit
 on the DM5406.  A value of 1 passed to this procedure enables the
```

```
 external trigger, a value of 0 disables the external trigger.

*********************************************************************/

void atodClass::SetExternalTrigger(unsigned char TriggerStatus)
{
 unsigned char B;

 B = inportb(BaseAddress + EXT_TRIG_ENABLE); /* read current byte */
 B = B & 191;                                /* clear B6 */
 B = B | TriggerStatus * 64;                 /* set Trigger bit */
 outportb(BaseAddress + EXT_TRIG_ENABLE, B); /* write new byte */
}

/*********************************************************************

 SetInterruptSource

 The SetInterruptSource procedure determines the source used for
generating
 interrupts for the DM5406.  Passing SOURCE_AD_START selects the A/D
start
 convert;  passing SOURCE_DMA_DONE selects DMA done;  passing
SOURCE_TRIGGER
 selects the trigger;  finally, passing SOURCE_PACER_CLOCK
 to this procedure selects the pacer clock.

*********************************************************************/

void atodClass::SetInterruptSource(unsigned char Source)

{
 unsigned char B;

      B = inportb(BaseAddress + SCAN_BURST_SLCT);     /*read current
byte*/
      B = B & 207;                                    /*clear bits 4 and
5*/
      B = B | Source * 16;                            /*set new bits*/
      outportb(BaseAddress + SCAN_BURST_SLCT, B);     /*write new byte*/
}

/*********************************************************************

 ScanBurstEnable

 The ScanBurstEnable procedure allows the user to enable either scan
mode
 or burst mode.  A value of 0 passed to this procedure disables both
modes;
 a value of 1 enables scan mode;  a value of 2 enables burst mode.

*********************************************************************/

void atodClass::ScanBurstEnable(unsigned char Enable)
```

```
{
 unsigned char B;

 switch (Enable)
      {
       case DISABLE_BOTH: Enable = 0; break;
       case SCAN_ENABLE: Enable = 2; break;
       case BURST_ENABLE: Enable = 3; break;
       default: Enable = 0;
      }
 B = inportb(BaseAddress + SCAN_BURST_SLCT);    /*read current byte*/
 B = B & 63;                                    /*clear bits 6 and 7*/
 B = B | Enable * 64;                           /*set new bits*/
 outportb(BaseAddress + SCAN_BURST_SLCT, B);    /*write new byte*/
}

/*********************************************************************

 SetScanChannels

 The SetScanChannels procedure determines how many channels are scanned
or
 bursted while in the appropriate mode, selected using ScanBurstEnable.
 The parameter passed in this procedure is simply the number of
channels the
 user wishes to include.

*********************************************************************/

void atodClass::SetScanChannels(int NumChannels)

{
 unsigned char B;

      B = inportb(BaseAddress + SCAN_BURST_SLCT);    /*read current
byte*/
      B = B & 240;                                   /*clear bits 0
through 3*/
      B = B | (NumChannels - 1);                     /*set new bits*/
      outportb(BaseAddress + SCAN_BURST_SLCT, B);    /*write new byte*/
}


/*********************************************************************

 StartConversion

 The StartConversion procedure is used to start conversions.

*********************************************************************/

void atodClass::StartConversion(void)

{
```

```
  outportb(BaseAddress + START_CONVERSION, 0);
}

/*********************************************************************

 ConversionDone

 The ConversionDone function returns TRUE if a conversion is complete,
FALSE
 if a conversion is in progress.

*********************************************************************/

int atodClass::ConversionDone()
{
 unsigned char Status;

 Status = inportb(BaseAddress + STATUS_BYTE);    /* read board status
*/
 if ( (Status & 1) == 1)
  return 1;                          /* if B0 is set return TRUE */
 else
  return 0;                     /* if B0 is not set return FALSE */
}

/*********************************************************************

 ReadData

 The ReadData function retrieves two bytes from the converter and
combines
 them into an integer value.

*********************************************************************/

int atodClass::ReadData(void)
{
 unsigned char MSB, LSB;
 int DigitalValue;

 MSB = inportb(BaseAddress + READ_DATA);
 LSB = inportb(BaseAddress + READ_DATA);

 DigitalValue = (MSB * 256 + LSB);

 return(DigitalValue);
}

/*********************************************************************

 ClockMode

 The ClockMode procedure is used to set the mode of a designated
counter
 on the 8254 programmable interval timer (PIT).
```

```
*********************************************************************/

void atodClass::ClockMode(unsigned char Clock, unsigned char Mode)
{
 unsigned char StatusByte;

 StatusByte = (Clock * 64) + (Mode * 2) + 48;
 outportb(BaseAddress + TIMER_CTRL, StatusByte);
}

/*********************************************************************

 ClockDivisor

 The ClockDivisor procedure is used to set the divisor of a designated
 counter on the 8254 programmable interval timer (PIT).  This procedure
 assumes that the counter has already been set to receive the least
 significant byte (LSB) of the divisor followed by the most significant
 byte (MSB).

*********************************************************************/

void atodClass::ClockDivisor(unsigned char Clock, unsigned int Divisor)
{
 unsigned char MSB, LSB;
 unsigned int PortID;

 PortID = BaseAddress + TIMER_A + Clock;
 LSB = Divisor % 256;
 MSB = Divisor / 256;
 outportb(PortID, LSB);
 outportb(PortID, MSB);
}

/*********************************************************************

 SetUserClock

 The SetUserClock procedure is used to set the programmable interval
 timer (PIT) on the DM5406 such that the output of counter 2 goes high
 at the specified rate.  The maximum attainable rate this procedure, as
 written, is 250,000 Hz although you can easily change this by
adjusting
 the divisors accordingly.

*********************************************************************/

void atodClass::SetUserClock(float Rate)
{
  ClockMode(0, 2);
  ClockDivisor(0, 8);

  ClockMode(1, 2);
  ClockDivisor(1, (500000.0 / Rate));
```

```
  ClockMode(2, 2);
  ClockDivisor(2, 2);
}

/*********************************************************************

 SetPacerClock

 The SetPacerClock procedure is used to set the programmable interval
 timer (PIT) on the DM5406 such that the output of counter 1 goes high
 at the specified rate.  The maximum attainable rate this procedure, as
 written, is 250,000 Hz although you can easily change this by
adjusting
 the divisors accordingly.

 Note that the Pacer and User clocks are really the same device and
 cannot be used independently.

*********************************************************************/

void atodClass::SetPacerClock(float Rate)

{
  ClockMode(0, 2);
  ClockDivisor(0, 16);

  ClockMode(1, 2);
  ClockDivisor(1, (500000.0 / Rate));
}

/*********************************************************************

 Read DigitalIO

 The ReadDigitalIO function returns the value of the specified digital
 input port.  Each digital input line is represented by a bit of the
 return value. Digital in 0 is bit 0, digital in 1 is bit 1, and so on.

*********************************************************************/

unsigned char atodClass::ReadDigitalIO(unsigned char InputPort)
{
 return(inportb(BaseAddress + PPI_A + InputPort));
}

/*********************************************************************

 WriteDigitalIO

 The WriteDigitalIO function sets the value of the digital output port
to
 equal the value passed as parameter v.  Each digital output line is
 represented by a bit of v.  Digital out 0 is bit 0, digital out 1 is
```

```
bit 1,
 and so on.

*********************************************************************/

void atodClass::WriteDigitalIO(unsigned char OutputPort, unsigned char
v)
{
 outportb(BaseAddress + PPI_A + OutputPort, v);
}
/*********************************************************************

 ConfigureIOPorts

 The ConfigureIOPorts procedure is used to configure the ports A and C
on
 the 8255 PPI for either input or output.  A value of 1 means input, a
value
 of 0 is for output.  It is advisable to use the INPUT and OUTPUT
constants
 defined in this file.

 Port B remains set for output.

*********************************************************************/

void atodClass::ConfigureIOPorts(unsigned char PortA, unsigned char
PortC)
{
 unsigned char ControlByte;

 ControlByte = 128 + (PortA * 16) + (PortC * 9);
 outportb(BaseAddress+PPI_CTRL, ControlByte);
}

/*********************************************************************
 UpdateDAC

 The UpdateDAC procedure outputs the specified voltage to the specifed
 DAC.  The DACSlope and DACOffset variables must be set to the values
 required for the output range of the DACs.

*********************************************************************/

void atodClass::UpdateDAC(unsigned char DAC, float Volts)
{
 int Value;

 Value = (int) (Volts * DACSlope) + DACOffset;
 outportb(BaseAddress + DAC1_LSB + (DAC - 1) * 2, Value % 256);
 outportb(BaseAddress + DAC1_MSB + (DAC - 1) * 2, Value / 256);
 outportb(BaseAddress + DAC_UPDATE, 0);
}
//end atod.cpp
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. MATLAB CODE FOR SANS KALMAN FILTER

```
clear all
format long

gps_flag_time=1;
cum_time=0;
x_hat=zeros(8,1);
simulation_time=5.0;    % in minutes
samples=(simulation_time * 60) / 0.025; % 0.025 is the average delta_t
heading=6.2832; % for north
speed=10;
x_hat_plot=zeros(samples,9);
temp=zeros(1,9);

% Time Constants
tau_1 = 10;    % seconds for velocity
tau_2 = 3600;  % seconds for current
tau_3 = 60;  % seconds for GPS

%R1 matrix
r1=[.01 0 0 0 ; 0 .01 0 0
     0 0 0 0 ; 0 0 0 0];
%R2 matrix
r2=[.01 0 ; 0 .01];

% P_minus matrix
p_minus = [.1  0  0  0  0  0  0  0 ;
            0 .1  0  0  0  0  0  0 ;
            0  0 .1  0  0  0  0  0 ;
            0  0  0 .1  0  0  0  0 ;
            0  0  0  0 .1  0  0  0 ;
            0  0  0  0  0 .1  0  0 ;
            0  0  0  0  0  0 .1  0 ;
            0  0  0  0  0  0  0 .1];

% P matrix
       p = [.5  0  0  0  0  0  0  0 ;
            0 .5  0  0  0  0  0  0 ;
            0  0 1.0  0  0  0  0  0 ;
            0  0  0 1.0  0  0  0  0 ;
            0  0  0  0 10.0  0  0  0 ;
            0  0  0  0  0 10.0  0  0 ;
            0  0  0  0  0  0 15.0  0 ;
            0  0  0  0  0  0  0 15.0];


%H1 matrix
h1=[1 0 0 0 0 0 0 0 ;
    0 1 0 0 0 0 0 0 ;
    0 0 0 0 1 0 1 0 ;
    0 0 0 0 0 1 0 1];
```

```
%H2 matrix
h2=[1 0 0 0 0 0 0 0 ;
    0 1 0 0 0 0 0 0];

D1=0.01;
D2=0.01;
D3=0.5;

D1_p = 0.01;
D2_p = 0.01;
D3_p = 0.5;

% initial process state
x_previous = [0; 0; 0; 0; 0; 0; 0; 0];

%for plotting only.
gps_plot = [ 0; 0];

'beginning kalman filter loops'

for i=1:samples

   delta_t=0.02; % 0.01*rand
   gps_flag_time=gps_flag_time+delta_t;
   cum_time=cum_time+delta_t;
   % construct Phi matrix
   uu=exp(-delta_t/tau_1);
   vv=exp(-delta_t/tau_2);
   ww=exp(-delta_t/tau_3);

   phi = [uu 0 0 0 0 0 0 0 ;
          0 uu 0 0 0 0 0 0 ;
          0 0 vv 0 0 0 0 0 ;
          0 0 0 vv 0 0 0 0 ;
          0 0 0 0 ww 0 0 0 ;
          0 0 0 0 0 ww 0 0 ;
         tau_1*(1-uu) 0 tau_2*(1-vv) 0 0 0 1 0 ;
         0 tau_1*(1-uu) 0 tau_2*(1-vv) 0 0 0 1];

   %construct Q matrix for Kalman filter
   xx=exp((-2*delta_t)/tau_1);
   yy=exp((-2*delta_t)/tau_2);
   zz=exp((-2*delta_t)/tau_3);
   Q00=((D1/(2*tau_1))*(xx));
   Q11=((D1/(2*tau_1))*(xx));
   Q22=((D2/(2*tau_1))*(yy));
   Q33=((D2/(2*tau_1))*(yy));
   Q44=((D3/(2*tau_1))*(zz));
   Q55=((D3/(2*tau_1))*(zz));
   Q661=D1*(delta_t-2*tau_1*(1-uu)+(tau_1/2)*(1-xx));
   Q662=D2*(delta_t-2*tau_2*(1-vv)+(tau_2/2)*(1-yy));
   Q66=Q661+Q662;
   Q77=D1*(delta_t-2*tau_1*(1-uu)+(tau_1/2)*(1-xx))+D2
         *(delta_t-2*tau_2*(1-vv)+(tau_2/2)*(1-yy));
```

```
Q06=D1*((1/2)-(uu)+(1/2)*xx);
Q60=Q06;
Q26=D2*((1/2)-(vv)+(1/2)*yy);
Q62=Q26;
Q17=Q06;
Q71=Q17;
Q37=Q62;
Q73=Q37;
%Q matrix
q = [Q00 0 0 0 0 0 Q06 0 ;
     0 Q11 0 0 0 0 0 Q17 ;
     0 0 Q22 0 0 0 Q26 0 ;
     0 0 0 Q33 0 0 0 Q37 ;
     0 0 0 0 Q44 0 0 0 ;
     0 0 0 0 0 Q55 0 0 ;
     Q60 0 Q62 0 0 0 Q66 0;
     0 Q71 0 Q73 0 0 0 Q77];

%construct Q matrix for process model
xx=exp((-2*delta_t)/tau_1);
yy=exp((-2*delta_t)/tau_2);
zz=exp((-2*delta_t)/tau_3);
Q00=((D1_p/(2*tau_1))*(xx));
Q11=((D1_p/(2*tau_1))*(xx));
Q22=((D2_p/(2*tau_1))*(yy));
Q33=((D2_p/(2*tau_1))*(yy));
Q44=((D3_p/(2*tau_1))*(zz));
Q55=((D3_p/(2*tau_1))*(zz));
Q661=D1_p*(delta_t-2*tau_1*(1-uu)+(tau_1/2)*(1-xx));
Q662=D2_p*(delta_t-2*tau_2*(1-vv)+(tau_2/2)*(1-yy));
Q66=Q661+Q662;
Q77=D1_p*(delta_t-2*tau_1*(1-uu)+(tau_1/2)*(1-xx))+D2_p
        *(delta_t-2* tau_2*(1-vv)+(tau_2/2)*(1-yy));
Q06=D1_p*((1/2)-(uu)+(1/2)*xx);
Q60=Q06;
Q26=D2_p*((1/2)-(vv)+(1/2)*yy);
Q62=Q26;
Q17=Q06;
Q71=Q17;
Q37=Q62;
Q73=Q37;

q_process = [Q00 0 0 0 0 0 Q06 0 ;
             0 Q11 0 0 0 0 0 Q17 ;
             0 0 Q22 0 0 0 Q26 0 ;
             0 0 0 Q33 0 0 0 Q37 ;
             0 0 0 0 Q44 0 0 0 ;
             0 0 0 0 0 Q55 0 0 ;
             Q60 0 Q62 0 0 0 Q66 0;
             0 Q71 0 Q73 0 0 0 Q77];

% process model noise
white_noise = randn(8,1);
w = sqrtm(q_process) * white_noise;
w(1) = w(1) + 0.02;   % input to generate some speed
```

```
x = phi * x_previous + w;
x_previous = x;

% measurement noise from water speed sensor
v = 0.2*randn(2,1);

% those two equations are here for smoothing rather than prediction.
x_hat_minus=phi*x_hat;
p_minus= (phi*p*phi')+q;

tmp(1) = cum_time;

if (gps_flag_time>=1)
   z_with_gps = h1 * x;   %measurement with GPS
   z_with_gps(1) = z_with_gps(1) + v(1);
   z_with_gps(2) = z_with_gps(2) + v(2);

   z_with_gps(3) = 0.0 + x(5);
   z_with_gps(4) = 0.0 + x(6);

   % k1=p_minus*h1'*inv((h1*p_minus)*h1'+r1);
   kinv=inv((h1*p_minus)*h1'+r1);
   k1=p_minus*h1'*kinv;

   ii=ii+4;

   x_hat=x_hat_minus+k1*(z_with_gps-(h1*x_hat_minus));
   p=(eye(8)-k1*h1)*p_minus;

   gps_plot(1) = z_with_gps(3);
   gps_plot(2) = z_with_gps(4);

   % for printing purpose
   for j=1:4
      tmp(j+1) = z_with_gps(j);
   end

   gps_flag_time=0;

else
   z_no_gps = h2 * x;     % measurement without GPS
   z_no_gps = z_no_gps + v;

   % k2=p_minus*h2'*inv((h2*p_minus)*h2'+r2);
   k2inv=inv((h2*p_minus)*h2'+r2);
   k2=p_minus*h2'*k2inv;
   k2invtmp=[k2inv(1,1) k2inv(1,2) 0 0;
             k2inv(2,1) k2inv(2,2) 0 0];

   x_hat=x_hat_minus+k2*(z_no_gps-(h2*x_hat_minus));
   p=(eye(8)-k2*h2)*p_minus;
   for j=4:5
      tmp(j) = gps_plot(j-3);
   end
```

```
      for j=2:3
         tmp(j) = z_no_gps(j-1);
      end

   end %end if statement

% collecting measurement for plotting purpose
z_plot(i,:) = tmp;

% collecting x_hat for plotting purpose
temp2(1)=cum_time;
temp3(1)=cum_time;
for j=2:9
   temp2(j)=x_hat(j-1);
   temp3(j)=x(j-1);
end

x_hat_plot(i,:)=temp2;
x_plot(i,:) = temp3;

end %end of Kalman Filter loop
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Yuh, J., *Underwater Robotic Vehicle: Design and Control,* TSI Press, Albuquerque, New Mexico, 1995.

2. Brown, R.G. and Hwang, P.Y.C., *Introduction to Random Signals and Applied Kalman Filtering*, Third Edition, John Wiley & Sons, Inc., 1997.

3. *TCM2 Electronic Compass Module User's Manual*, Precision Navigation Inc., December 1999.

4. *CMV586DX133 CPU Module User's Manual*, Real Time Devices, Inc., September 1997.

5. *CMT104 IDE Controller and Hard Drive Carrier utiliyModule,* Real Time Devices, Inc., September 1997.

6. *CM1122 Super VGA+Flat Panel Utility Module,* Real Time Devices, Inc., September 1997.

7. *C4-104 User Manual*, Sealevel Systems Inc., January 1997.

8. *DM406/DM5406 Analog/Digital Converter User's Manual*, Real Time Devices, Inc., November 1996.

9. *Oncore User's Guide*, Motorola Inc., August 1995.

10. *DMU User's Manual,* Crossbow Technology Inc., October 1998.

11. Yun, X., Bachmann, E.R., McGhee, R.B., Whalen, R.H., Roberts, R.L., Knapp, A.J., Healy, A.J., and Zyda, M.J., "Testing and Evaluation of an Integrated GPS/INS System for Small AUV Navigation," *IEEE Journal of Oceanic Engineering*, Vol. 24, No. 3, July 1999.

12. Yun, X., Bachmann, E. R. and Arslan, S., "An Inertial Navigation System for Small Autonomous Underwater Vehicles," *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, San Francisco, California, April 2000.

13. Akyol, K., "Hardware Integration of the Small Autonomous Underwater Vehicle Navigation System (SANS) Using a PC/104 Computer," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1999.

14. Hernandez, C.G., "An Integrated INS/GPS Navigation System for Small AUV Using an Asynchronous Kalman Filter," Master's Thesis, Naval Postgraduate School, Monterey, California, June 1998.

15. Knapp, R., "Design and Calibration of a Water Speed Sensor for a Small AUV Navigation System (SANS)," Master's Thesis, Naval Postgraduate School, Monterey, California, December 1997.

16. *Model TRT-7 Tilting Rotary Table*, Haas Automation, Inc., Chatsworth, CA, July 1992.

17. The Internet, [http://www.geolab.emr.ca/geomag/e-cgrf.html], November 1999.

18. The Internet, [http://sulu.lerc.nasa.gov/dictionary/m.html], March 2000.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. Prof. Xiaoping Yun, Code EC/Yx 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

5. Eric Bachmann, Instructor, Code CS/Bc 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5118

6. Prof. Robert McGhee, Code CS/Mz 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5118

7. Deniz Kuvvetleri Komutanligi 1
   Personel Tedarik ve Yetistirme Daire Baskanligi
   06100 Bakanliklar, Ankara
   TURKEY

8. Suat Arslan 2
   Bayraktar Mah. Zulfikar Sok.
   No=24/8 Kucukesat, Ankara
   TURKEY

9. Orta Dogu Teknik Universitesi 1
Department of Computer Engineering
06531 Ankara
TURKEY

10. Bogazici Universitesi 1
Department of Computer Engineering
80815 Bebek, Istanbul
TURKEY

11. Bilkent Universitesi 1
Department of Computer Engineering and Information Science
06533 Bilkent, Ankara
TURKEY

12. Deniz Harp Okulu Komutanligi 1
Kutuphane
81704 Tuzla, Istanbul
TURKEY