1995-09

Design and evaluation of an integrated GPS/INS system for shallow-water AUV Navigation

Bachmann, Eric R.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND EVALUATION OF AN INTEGRATED GPS/INS SYSTEM FOR SHALLOW-WATER AUV NAVIGATION (SANS)

by
Eric Bachmann and David Gay

September 1995

Thesis Co-Advisors:	Robert B. McGhee
	Don Brutzman
Second Reader:	James Clynch

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	3. REPORT TYPE AND DATES COVERED
	September 1995	Master's Thesis

4. TITLE AND SUBTITLE
DESIGN AND EVALUATION OF AN INTEGRATED GPS/INS SYSTEM FOR SHALLOW-WATER AUV NAVIGATION (SANS)

5. FUNDING NUMBERS

6. AUTHOR(S)
Bachmann, Eric R.
Gay, David L.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
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 authors and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The major problem addressed by this research is the large and/or expensive equipment required by a conventional navigation system to accurately determine the position of an Autonomous Underwater Vehicle (AUV) during all phases of an underwater search or mapping mission.

The approach taken was to prototype an integrated navigation system which combines Global Positioning System (GPS) and Inertial Measurement Unit (IMU), waterspeed and heading information using Kalman filtering techniques. Actual implementation was preceded by a computer simulation to test where the unit would fit into a larger hardware and software hierarchy of an AUV. The system was then evaluated in experiments which began with land based cart tests and progressed to open water trials where the unit was placed in a towed body behind a boat and alternately submerged and surfaced to provide periodic GPS updates to the Inertial Navigation System (INS).

Test results and qualitative error estimates indicate that submerged navigation accuracy comparable to that of differential GPS may be attainable for periods of 30 seconds or more with low cost components of a small physical size.

14. SUBJECT TERMS
Autonomous underwater vehicles, GPS/INS integration, navigation, virtual environment, kinematics, Robotics, NPS AUV, Jack, LISP simulations

15. NUMBER OF PAGES
241

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

Approved for public release; distribution is unlimited

DESIGN AND EVALUATION OF AN INTEGRATED GPS/INS SYSTEM FOR SHALLOW-WATER AUV NAVIGATION

Eric R. Bachmann
Lieutenant, United States Navy
B.A., University of Cincinnati, 1983

David L. Gay
Lieutenant Commander, United States Navy
B.S., University of New Mexico, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

September 1995

Authors: [redacted]

Eric R. Bachmann David L. Gay

Approved by: [redacted]

Robert B. McGhee, Thesis Co-Advisor

[redacted]

Don Brutzman, Thesis Co-Advisor

[redacted]

James Clynch, Second Reader

[redacted]

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The major problem addressed by this research is the large and/or expensive equipment required by a conventional navigation system to accurately determine the position of an Autonomous Underwater Vehicle (AUV) during all phases of an underwater search or mapping mission.

The approach taken was to prototype an integrated navigation system which combines Global Positioning System (GPS)and Inertial Measurement Unit (IMU), waterspeed and heading information using Kalman filtering techniques. Actual implementation was preceded by a computer simulation to test where the unit would fit into a larger hardware and software hierarchy of an AUV. The system was then evaluated in experiments which began with land based cart tests and progressed to open water trials where the unit was placed in a towed body behind a boat and alternately submerged and surfaced to provide periodic GPS updates to the Inertial Navigation System (INS).

Test results and qualitative error estimates indicate that submerged navigation accuracy comparable to that of differential GPS may be attainable for periods of 30 seconds or more with low cost components of a small physical size.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This research was only possible due to efforts of many people. Instrumental in the success of this research was the assistance of Frank Kelbe and Russ Whalen. Each volunteered his expertise, advice, and time unselfishly. Without Frank Kelbe the SANS would have lead a solitary existence, communicating with nothing, due to the mysteries of serial port communications. Russ Whalen's wide-ranging technical experience was singularly instrumental in providing the SANS with a warm and dry home in which endure the trials and tribulations of the at-sea tow tests. Throughout, the many frustrations of this research Russ remained cheerful and supportive.

Appreciation also goes to Dr. Robert McGhee and Dr. Donald Brutzman who served as co-advisors for this thesis. Both helped to make it an enjoyable and rewarding experience. The patience and enthusiasm of Dr. McGhee was truly amazing. Even after explaining concepts repeatedly, he remained ready and willing to do it all over again. The eagle-eye of Dr. Brutzman went far to ensure that the final results of this research were documented clearly and logically.

An expression of gratitude goes to the second reader, Dr. James Clynch. The technical assistance and experimental facilities provided by this professional scientist allowed the integrated SANS to get off the beach and into the water.

We would also like to thank Walter Schubert who was solely responsible for the electronic design, development and testing of the 10 Hz filter and most of the low-level tracing of 1's and 0's in the SANS hardware.

Most importantly, the authors like to thank their wives Cindy Gay and Dans Bachmann. Each morning they cheerfully and lovingly kicked their husbands out of the house and took care home, hearth, school and kids until their husbands returned sometime in the wee hours of the morning.

The hardships of the tow testing were greatly reduced by the valuable assistance provided by the Coast Guard Group, Monterey. This research was supported in part by

Grant BCS-9306252 from the National Science Foundation to the Naval Postgraduate School.

DEDICATION

For Mandy, Jeremy, and Carol

I. INTRODUCTION

A. BACKGROUND

An Autonomous Underwater Vehicle (AUV) can be capable of numerous missions both overt and clandestine. Such vehicles have been used for inspection, mine countermeasures, survey, observation, etc. [Yuh 95]. Recent research trends in underwater robotics have emphasized minimizing the need for human interaction by increasing the autonomy of such vehicles.

The NPS "Phoenix" AUV is an experimental vehicle designed primarily for research in support of shallow-water mine countermeasures and coastal environmental monitoring [Healey 93, 95]. In [Kwak 93], an approach is described for determining the position of submerged detected objects by executing a "pop-up" maneuver to obtain a GPS fix, and then extrapolating this fix backwards to the submerged object location using recorded inertial data. As explained in [Kwak 93], navigation accuracy during such a surfacing maneuver is strongly enhanced by the use of accurate depth information available from low-cost pressure cells. However, this form of "aided" inertial navigation [Brown 92], is not applicable to a surfaced AUV. Of course, inertial navigation is not needed in circumstances where continuous reliable reception of GPS satellite signals is possible. However, this does not apply to AUVs, unless perhaps they are fitted with a mast to extend a GPS antenna above the effects of wave action. Such a mast is not an attractive option for military operations, and in any event may be mechanically difficult.

Recognizing the problem of intermittent GPS satellite tracking for surfaced (or cruising near the surface) AUV navigation, an experimental system has been designed which uses a low-cost strapped-down inertial measurement unit (IMU) to enable inertial navigation between GPS fixes. This IMU also is appropriate to pop-up navigation, so finding a means of navigating near the surface provides a complete solution to the overall navigation problem associated with transiting an AUV to a shallow water work site,

recording the position of detected submerged objects, and then returning to a recovery site where stored mission data can be uploaded.

Many of the missions of the Phoenix class of vehicles can be separated into two distinct phases: transit and search. After being launched from an aircraft, submarine or surface vessel such an AUV would conduct the transit phase of the mission in order to arrive at the search area. After the search phase, the AUV would transit back to a recovery position. Neither of these transit phases require as high a degree of navigation accuracy as the search phase. Once established in the mission area, the Phoenix would enter the search phase which might include missions such as minehunting, mapping, or environmental data collection. Typically, the search phase would require more precise navigation which could be provided by more frequent GPS fixes or by using Differential GPS (DGPS) or post-processing, if available. Both mission phases may involve waypoint steering and (AI) artificial intelligence applications such as obstacle avoidance.

One of the most important and difficult aspects of an AUV mission is navigation. It is important that the navigation system be robust if the AUV is to be capable of a wide variety of missions. In order to achieve such robustness, the AUV should be capable of navigating with the Global Positioning System (GPS) and/or an Inertial Navigation System (INS). The GPS is capable of highly accurate positioning when the AUV is surfaced, while an INS can be used for submerged navigation. In order to ensure accurate navigation for the various missions, the GPS and INS components can be combined. A favorable analysis of this type of navigation system was conducted in [McKeon 92]. The hardware and software architecture required for a typical mapping scenario was evaluated in [Norton 94].

In order to implement intermediate testing of a prototype navigation subsystem prior to installation into the Phoenix, the navigation system reported in this thesis was broken into two subsystems in which a minimum number of components were placed in a towable device (towfish) which can be commanded to submerge and surface. The remainder of the components were placed in the towing vessel. This results in a smaller towfish, with

reduced power requirements, and also allows for human monitoring and interaction during the course of an experiment.

B. RESEARCH QUESTIONS

This thesis will examine the following research areas:

- Evaluate the hardware and software architecture for the GPS/INS installation in the Naval Postgraduate School (NPS) Phoenix AUV.

- Evaluate the feasibility of an AUV navigating from point to point using GPS/INS while conducting open-ocean transit.

- Develop the software interface which will communicate between the embedded GPS/ INS and the Phoenix controlling programs.

- Evaluate the capability of the Phoenix to obtain accurate navigation fixes by utilizing Differential GPS (DGPS).

C. SCOPE, LIMITATIONS AND ASSUMPTIONS

This thesis reports the findings of the fourth year of research in an ongoing research project. The scope of this thesis is to investigate the feasibility of an AUV navigating from point to point using a combination of GPS/INS. Also, the scope includes developing a simulation written in LISP which implements a GPS/INS navigation system and a Kalman filter. The requirements for a Small Autonomous Underwater Vehicle (AUV) Navigation System (SANS) described by [Kwak 93] which impact this project are:

- Low power consumption. Operation from an external battery pack for 24 hours is desirable.

- Limited exposure time. The amount of time that the GPS antenna is exposed in the search phase should be as short as possible. Up to 30 seconds of exposure is allowed, but time between exposures should be maximized.

- Maintain clandestine operation. The GPS antenna should present a very small cross section when exposed and ought not extend more than a few inches above the surface of the water.

- Maximize accuracy. During the search phase of the mission, system accuracy of 10 meters rms or better is required with postprocessing, both submerged and surfaced.

- Total volume not to exceed 120 cubic inches. Elongated, streamlined packaging is preferred.

- For the purposes of this research, DGPS will be used as ground truth data (without postprocessing) for determining appropriate Kalman filter gains. However, most real-world scenarios will only utilize the noncorrected GPS signal for real-time mission navigation and may require further tuning of Kalman filter coefficients.

D. ORGANIZATION OF THESIS

The purpose of this thesis is to present the interim development of a system meeting the mission requirements of the SANS. The term AUV is understood to include any small underwater vehicle (including human divers) which can easily carry such a compact device. This thesis provides an evaluation of the hardware and software used to provide accurate navigation for the NPS AUV. The major thrust of the thesis is twofold: first, evaluating the experimental results of a GPS/INS integrated unit (towfish) which was towed behind a boat in both submerged and surfaced states; second, developing a LISP computer simulation which implements a GPS/INS integrated system which also utilizes a Kalman filtering technique.

Chapter II reviews the previous work on this project as well as previous work on GPS navigation and AUV submerged navigation.

Chapter III is a detailed problem statement which includes the mission requirements, and the problems related to GPS navigation, submerged navigation, and the LISP navigation simulation.

Chapter IV is a detailed description of the hardware currently in use for this portion of the project. A description of each of the components used for the towfish experiment is included.

Chapter V is a detailed description of the software used for this portion of the project. The chapter includes a description of the C++ code required for the towfish experiment. This description provides an explanation of the class and object hierarchy used, as well as an outline of the functions of the major functions. The Kalman filter of the INS is also discussed here.

Chapter VI discusses the LISP simulation. It explains how the objects fit into Rational Behavior Model hierarchy and highlights the purpose each class serves in the simulation. In describes in some detail the mathematics behind the motion simulation. A sample display of the AUV tracking from point to point is included here.

Chapter VII is a description of the experiment design and analysis of the experimental results. This chapter covers the methodology of the design and implementation of the towfish experiment as well as the push cart and bench tests.

Chapter VIII presents the conclusions and recommendations of the hardware and software testing and provides recommendations for future research.

Throughout the research conducted as part of this thesis, tasks were divided between software and hardware. Eric Bachmann was principally responsible for the former. David Gay was principally responsible for the later. Initial authorship of chapters I, II, IV, and VI belongs to David Gay. Eric Bachmann authored the first drafts of Chapters III, V, VII, and VIII.

II. SURVEY OF RELATED WORK

A. INTRODUCTION

Autonomous Underwater Vehicles (AUVs) have the potential to be used in an efficient and cost effective manner in a variety of missions involving military and non-military applications. One of the most important aspects of the AUV mission is the ability to navigate accurately. Many possible missions, (such as minehunting), require a high degree of navigation accuracy. This chapter will discuss some of the solutions for navigating the AUV. Additionally, computer simulations of robot kinematics and dynamics can be used to simulate the actions of the AUV, thus helping to establish the requirements of the navigation system.

In general there are two categories of navigation systems: those that are based on external signals and those that are based on sensors. External-signal-based navigation systems such as, Loran, Omega, and Global Positioning System (GPS) are only able to determine position while the signal receiver is exposed to the signal. Loran and Omega are relatively inaccurate compared to GPS. While Loran covers almost the entire northern hemisphere, it has almost no coverage in the southern hemisphere [Bowditch 84]. GPS on the other hand is capable of world-wide coverage with a high degree of navigational accuracy.

Sensor-based navigation can be implemented as a self-contained unit which can be composed of various types of equipment such as inertial measuring units (IMUs), acoustic transponders, or geophysical map comparison. All sensors are subject to some amount of error and on long AUV missions this error may not allow for the accuracy required by some missions. Each of these components has its disadvantages. Acoustic beacons must be pre-deployed at precisely known locations and may require costly maintenance. Geophysical map interrogation requires a precise bottom contour map previously stored in the AUV's computer. IMU-based navigation is prone to sensor drift, which if left uncorrected, can become very large.

B. GPS NAVIGATION

The Navigation Satellite Timing and Ranging (NAVSTAR) Global Positioning System (GPS) is a space-based radio positioning, navigation and time-transfer system sponsored by the U.S. Department of Defense (DoD). It was originally intended to provide the military with precise navigation and timing capabilities [Parkinson 80]. The system is designed to provide 24-hour, all-weather navigation by providing total earth coverage using 24 satellites in 22,200 km orbits that are inclined 55^0, with 12 hour periods. The satellites broadcast two L-band frequencies: L1 (1575.4 MHz) and L2 (1227.6 MHz). Navigation and system data, predicted satellite position (ephemeris) information, atmospheric propagation correction data, satellite clock error information, and satellite health data are all superimposed on these two carrier frequencies [Logsdon 92, Wooden 85].

There are two different navigation services available from the GPS satellites depending on the type of receiver being used: the Standard Positioning Service (SPS) and the Precise Positioning Service (PPS). The SPS is achieved by receiving the L1 carrier signal which is broadcast with an intentional inaccuracy called Selective Availability (SA). SA limits world-wide navigation to 100 m horizontal accuracy with a 95% confidence level [Logsdon 92]. The PPS is limited to US and allied military, and specific non-military uses that are in the national interest. Access to PPS is restricted by use of special cryptographic equipment. PPS provides the highest stand alone accuracy: 15 m Spherical Error Probable (SEP), a velocity accuracy of 0.1 m/sec, and a timing accuracy of better than 100 nanoseconds [Logsdon 92, Wooden 85].

In order to take full advantage of GPS precision without having access to cryptographic equipment, the civilian market had to determine a way to improve the accuracy of the SPS. The two most common ways to work around the inaccuracies of the SPS are real-time Differential GPS (DGPS) and post-processing DGPS. The idea behind DGPS is to survey a receiver at a stationary site, allow the stationary site to determine the difference between its actual position and its GPS position, and broadcast the pseudorange

corrections to any DGPS capable receivers. Real-time differential processing can reduce the typical 100 m accuracy of the SPS to 2-4 m regardless of the status of SA [Logsdon 92]. In the case of post-processing, it is possible to have the AUV record the raw PPS or SPS GPS information for later comparison to a known geographical site. Precise post-processing procedures can be used to reconstruct extremely accurate positioning information, typically in the submeter range. Table 1 shows a comparison of expected GPS accuracies.

POSITIONING SERVICE	PPS (m)	SPS (m)
NON-DIFFERENTIAL	16	100
DIFFERENTIAL	2-4	2-4

TABLE 1: Expected RMS GPS accuracy levels [Logsdon 92]

As the GPS technology has matured, the size and cost of GPS receivers has decreased drastically. Not only is miniaturization improving, but GPS receivers have maintained or increased in performance capability. Since as early as 1992, the GPS industry has been able to produce receivers that are essentially a single printed circuit board. [Souen 92] reports that the Furuno GPS receiver module LGN-72 is an eight-channel receiver which is a single printed circuit board measuring 100 mm x 70 mm x 20 mm and requiring only 2 W of power.

Given the level of miniaturization and performance along with the excellent accuracy, GPS is an obvious choice for AUV navigation. One manner of using GPS to locate an AUV is to place buoys with GPS receivers at appropriate locations. These buoys would translate the GPS signal and retransmit an underwater acoustic signal. The AUV would determine its position via ranging and position fixes to the buoys. [Youngberg 91] suggests that the GPS antenna, receiver, processing and control subsystem, acoustic transmitter, battery power, and homing beacon could all be contained in a buoy measuring 123 mm diameter x 910 mm long and weighing 5-15 kg. A simulation which showed the feasibility of this

approach is presented in [Leu 93]. The simulation consisted of several sonobuoys spaced one kilometer apart. Due to uncertainties in buoy position caused by wave action and variations in altitude, the study proposed the use of Kalman filtering techniques to combine the outputs of an accelerometer and DGPS to enhance accuracy. Each GPS buoy would essentially act as a GPS satellite and broadcast its position via spread spectrum signals used by the AUV for ranging. This technique would significantly reduce the requirement to predeploy a surveyed transponder field.

Another possible use for using GPS to determine the AUV's position is to physically mount the GPS antenna and receiver onboard the AUV. One major concern would be that the GPS receiver would be unable to acquire satellites in a timely manner suitable to the mission due to splash effects on the antenna. [Norton 94] describes both static and dynamic test results which show that a submersible system is able to meet the accuracy and time requirements of the mission while being splashed by wave wash.

C. AUV SUBMERGED NAVIGATION

There are many techniques available for submerged navigation, including dead reckoning, inertial, electromagnetic and acoustic navigation. With acoustic navigation, time of arrival and direction of propagation of acoustic waves are the two principal measurements made. A wide variety of acoustic navigation systems have been developed for underwater vehicle use. They are typically divided into long, short, and ultrashort baseline systems (LBL, SBL, and USBL). All involve the use of an array of acoustic beacons or receivers whose positions must be known to an accuracy better than the desired vehicle localization accuracy [Tuohy 93]. Unfortunately, most acoustic navigation systems require major expeditions for their accurate set-up and periodic maintenance. This makes them expensive and in many ways reduces the level of autonomy achievable by an AUV. Also, acoustic navigation methods are affected by changes in the speed of sound in the ocean and suffer from refraction and multipath propagation problems in restricted shallow water coastal and ice-covered areas [Tuohy 93].

There are various other ways of determining a vehicle's velocity and position while submerged without the aid of external signals. An AUV could use Doppler sonar or side-scanning sonar to determine velocity. Charge Coupled Device (CCD) cameras, laser scanning or variations in the earth's magnetic field can also aid in determining position [Bergem 93]. Position could also be estimated by the double integration of acceleration as sensed by an Inertial Navigation System (INS).

Unless an AUV has access to acoustic information, it will not be able to refer to external signals while submerged. If this is the case, then the system must rely on some sort of dead reckoning. Dead reckoning is a form of navigation where position is determined by integrating estimated velocity over a time interval. Modern dead reckoning systems typically use magnetic or gyroscopic heading sensors and a bottom or water-locked velocity sensor [Grose 92]. The main problem is that the presence of an ocean current will add a velocity component to the vehicle which is not detected by the speed log. In the vicinity of the shore, ocean currents can exceed two knots [Tuohy 93]. Using dead reckoning with currents which are relatively large in relation to the typical 4-6 knot speed of an AUV can produce extremely inaccurate results [Tuohy 93].

Inertial navigation is basically a complex method of dead reckoning. In its purest form it involves no outside references to fix position. All position data is calculated based on a known starting point. An inertial navigation system continuously measures three mutually orthogonal acceleration components using various types of accelerometers. These measurements are taken in increments and multiplied by elapsed time in order to determine the instantaneous velocity. The three-dimensional change in position can then be determined by multiplying the velocity by an appropriate time increment.

There are many techniques for measuring accelerations and angular rates. These include using ring laser and fiber optic gyros, rotating mass gyros, vibratory rate sensors and high performance Inertial Measuring Units (IMUs). The inertial grade IMUs typically contain three angular rate sensors, three precision linear accelerometers and a three-axis magnetometer. The acceleration measurements required by an Inertial Navigation System

(INS) can be made by several types of Inertial Measurement Units. These can be divided into two fundamental categories: gimbaled and strapdown. In a strapdown unit, three mutually orthogonal accelerometers and three angular rate sensors are mounted parallel to the body axes of the vehicle. Changes in vehicle attitude, position and velocity can then be continuously measured. Strapdown systems are smaller and simpler than gimbaled systems, but necessitate much larger computational loads [Logsdon 92]. All of these sensors are subject to drift errors which relentlessly increase with time. High quality sensors are subject to less drift but can cost up to $100,000 [Tuohy 93] making them unattractive for small AUVs.

[McKeon 92] proposes a combination of GPS and INS to allow an AUV to determine position information. While submerged, the AUV is uses a low-cost inertial navigation system. However, when on the surface the vehicle has access to GPS information. GPS/INS information could be combined with a Kalman filter techniques to reduce the errors during the next dive sequence as simulated in [Nagengast 92] and demonstrated in [McGhee 95]. A more thorough discussion of Kalman filtering techniques can be found in [Brown 92].

D. ROBOT KINEMATICS AND DYNAMICS SIMULATION

The motion of rigid bodies, as described in physics and engineering, can be used to simulate movement by using robot kinematics and robot dynamics. Robot kinematics is a systematic approach of using vector and matrix algebra to represent the spatial geometry of an object with respect to a fixed frame of reference, all as a function of time. Robot dynamics uses the mathematical equations and physical laws describing motion such as Newton-Euler equations to represent motion from applied forces and moments [Fu 87].

Using robot kinematics and dynamics to model real-world motion of an AUV in a computer simulation is an extremely powerful tool. Simulation allows a system designer to establish the requirements of the system prior to proceeding to expensive fabrication and testing in a possible unfriendly environment. [Davidson 93] provides an explanation of the kinematics and graphical computer simulation code for an underwater walking machine

written in Common Lisp Object System (CLOS) and C++. The object-oriented programming approach used allows the programmer to easily make adjustments and modifications to individual components in complex system designs.

[Norton 94] describes a computer simulation of an AUV navigating with the SANS. The simulation was used in order to help determine the errors created by the various sensors. The code was written in CLOS and allowed the AUV and SANS to be represented as objects and classes were are linked together to create an entire system which was able to move as a single unit. Using robot kinematics and dynamics based on Newton-Euler-equations of motion, the system was able to simulate an actual mission.

Observing, communicating with, and testing underwater robots is difficult due to their operations in remote and hazardous environments. A great number of robotics-related simulations have been produced, but few involve mobile robots. These simulations are typically approached in a piecemeal and fragmented fashion. Thus simulation results remain susceptible to failure when deployed in the real world due to the untested complexity of unforgiving environments [Brutzman 94]. There is no safe and complete "practice" environment for AUVs, since test tanks cannot reproduce the variability of critical parameters found in the ocean and since any in-water failure may lead to vehicle damage or loss due to flooding. [Brutzman 94] develops a virtual world using 3D real-time graphics designed from the perspective of the robot, enabling realistic AUV evaluation and testing in the laboratory. A networked architecture enables multiple world components to operate collectively in real time, and also permits world-wide observation and collaboration with other scientists interested in the robot and virtual worlds. The real-time six-degree-of-freedom hydrodynamics model used in the virtual world provides an excellent opportunity for further testing of robot navigation algorithms in the presence of buoyancy effects and wave action.

E. SUMMARY

The previous survey has shown that there are many ways to overcome the challenges associated with AUV navigation. The choices range from simple dead reckoning to systems which use acoustic information from floating or stationary transponders to complex systems which use sophisticated IMUs and GPS receivers combined with Kalman filtering techniques. In order to reduce the hazards associated with the design of a physical system, it is possible to attempt to model the proposed AUV using robot kinematics and robot dynamics in a computer simulation. This simulation can be used to establish system requirements prior to placing the physical system in a dangerous real-world environment. The final choice of a navigation system is largely dependent on the mission profile which may have different requirements for navigation accuracy at different stages.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

There are many possible types of missions which can be carried out by an AUV. Many of these missions involve some form of underwater mapping or search for objects of interest. This implies that the locations of many of the objects encountered by the AUV need to be accurately known once the mission has been completed. Given this requirement, the ability of the AUV to accurately determine its position (either in real time or following the mission during postprocessing) is a key requirement. The exact nature of the mission (whether overt or clandestine) places limitations upon how this determination can be made. These limitations are manifested in both the maximum time the AUV can spend on the surface and the types of external navigational assistance available to it. In either case, some combination of GPS and inertial navigation may be able to provide the required accuracy.

The general profile of an AUV search or mapping mission can be divided into phases consisting of two types. The first type can be termed a transit phase. During this phase the AUV would navigate relatively great distances between search areas, transit from a launch point to an initial search area or transit from the final search area to a recovery point. These mission phases are be characterized by a need for only moderately accurate navigation information. For the most part this could be accomplished using periodic fixes together with dead reckoning. The second type of mission phase is search. During a search phase the AUV would attempt to locate objects of interest and record their locations. The accuracy of navigation required during the search phase is much greater, whether accomplished in real time or during mission postprocessing. Navigation during this phase must be continuously accurate, not just during short discrete time periods following GPS fixes.

[Norton 94] demonstrates the feasibility of using GPS to attain accurate positional fixes on the surface by using differential postprocessing. Whether due to errors or intermittent fixes, GPS positional information can be described as accurate only in the long

term. In comparison, inertial data is accurate in the short term but tends to drift over time. The data of the two systems can therefore be used to complement each other. The research of this thesis combines real-time differential corrected GPS data with the measurements taken from a low-cost Inertial Measurement Unit (IMU), all using Kalman filtering techniques. This provides a continuously accurate estimate of position in real time as required during the search phases of an AUV mission. In this refined configuration, the methods used will meet the requirements outlined for the SANS baseline system described in [Kwak 93].

B. GPS NAVIGATION

The Navstar Global Positioning System is a satellite-based navigation system. It provides continuous world-wide navigation information to an unlimited number of units equipped with receivers capable of processing the signals being broadcast by the satellites. There are two different levels of navigation service provided by the system: the Standard Positioning System (SPS) and the Precise Positioning System (PPS) [Logsdon 92]. The PPS utilizes an encrypted P-code (Precision code) which is reserved for high-precision military users. This code restricts the most accurate navigation information of the Global Position System to US and allied military and specific US non-military users. PPS provides a stand-alone accuracy of 15 m SEP (Spherical Error Probable), a velocity accuracy of 0.1 m/sec, and a timing accuracy of better than 100 nanoseconds [Van Dierendonck 80, Wooden 85]. In most cases it is not be desirable to provide an unmonitored AUV with the cryptographic keys needed to obtain PPS due to the fallible nature of any safe guarding system for the keys and the potential for compromise.

SPS information includes an intentional inaccuracy which is introduced into the satellite broadcast signal through a process called Selective Availability (SA). This limits SPS to a 100 m horizontal accuracy with a 95% confidence level. Differential GPS (DGPS) is a method which allows highly accurate information to be obtained from GPS without the cryptographic equipment required for access to the P-code of PPS. DGPS entails placing

one receiver at a known location, determining local satellite range errors, and broadcasting error corrections to nearby GPS users. Differential processing can be done in real-time or during mission postprocessing. This procedure improves GPS accuracy to 2-4 m regardless of whether SA is utilized [Logsdon 92].

SPS could be used to adequately perform both the transit and search phases of an AUV mission. During the transit phases, non-differential SPS and a magnetic compass would provide the primary source of navigation data. In order to utilize GPS as a meaningful correction to a low-cost INS system, periods between fixes during the transit phase must not exceed the time in which an AUV could travel a distance greater than the horizontal accuracy of SPS (100m). The mapping phases of an AUV mission would require the vehicle to maintain more accurate navigational picture both submerged and on the surface. This would necessitate the use of periodic differential corrected GPS information in order to keep the INS system accurate while submerged. This differential correction could be provided in real-time during overt missions along friendly shores providing a DGPS reference signal is provided or during mission postprocessing following a clandestine mission.

C. INERTIAL NAVIGATION

Inertial navigation is basically a complex method of dead reckoning. In its purest form it involves no outside references to fix position. All position data is calculated relative to a known starting point. An inertial navigation system (INS) continuously measures three mutually orthogonal acceleration components using accelerometers. These measurements are taken in short time increments and multiplied by elapsed time in order to determine an estimate of instantaneous velocity. The three-dimensional change in position can then be determined by integrating respective velocities over time.

The primary drawback of any INS is the tendency for small sensor drift rates to accumulate errors over time. Without outside references for correction, these errors grow relentlessly and eventually lead to large errors in the estimated position. Highly accurate

inertial navigation systems can be constructed but they are large, costly, and complex [Tuohy 93]. Size alone makes them unacceptable candidates for the SANS. In order to meet SANS physical requirements, a low-cost INS can be integrated with GPS. GPS will provide the INS with the periodic position fixes necessary to correct slowly building INS errors.

The acceleration measurements required by an INS can be made by several types of Inertial Measurement Units (IMUs). These can be divided into two fundamental categories: gimbaled and strapdown. Due to their large size and power requirements most gimbaled systems are not suitable for the SANS. In a strapdown unit, three mutually orthogonal accelerometers and three angular rate sensors are mounted parallel to the three body axes of the vehicle. Changes in linear and rotational velocities are continuously measured. Strapdown systems are smaller and simpler than gimbaled systems, but necessitate much larger computational capabilities [Logsdon 92].

D. INTEGRATED INS/GPS NAVIGATION

Integration of INS and GPS into a single system can produce continuously accurate navigational information even when using relatively low-cost components. This integration not only allows periodic reinitialization of the INS to correct accumulated errors but can also (with the aid of Kalman filtering techniques) improve the performance of the INS between fixes. Filtering the acceleration data with additional sensor information such as water speed and heading will further improve the quality of the integrated system. Overall, an integrated system will provide improved reliability, smaller navigation errors and superior survivability [Logsdon 92].

Kalman filtering is a method of combining all available sensor data regardless of their precision to estimate the current posture of a vehicle [Cox 90]. The filter is actually a data-processing algorithm which minimizes the error of this estimate statistically using currently available sensor data and prior knowledge of system characteristics. Each piece of data is weighted based upon the expected accuracy of the measurement it represents. In a complementary filter, low- frequency data which is trusted over the long term and high-

frequency data which is trusted only in the short term are used to "complement" each other providing a much better estimate than either can alone [Brown 92].

For this research the low-frequency data of the accelerometers and compass is combined with high-frequency angular rate and heading information using this complementary filter technique. Intermediate results are again filtered using high-frequency water-speed data. GPS data is used to reinitialize the system each time a fix is obtained and develop an error bias to correct the system between fixes.

E. NAVIGATION SIMULATION

The SANS proposed for use aboard the Phoenix has been simulated in order to simplify the process of determining the errors created by the sensors. This simulation is written using Common LISP Object System (CLOS), an object-oriented programming language. This allows the hardware and software of the SANS and AUV to be represented as objects and classes that are combined to create an entire system. The simulation further gives the AUV and its various components the physical properties of a rigid body. Other objects in the simulation represent the various software objects needed to interface with the hardware and a tactical level navigator as described in [Byrnes 93]. The navigator object monitors the position of the AUV and makes the required inputs to the AUV to navigate between a series of waypoints. The AUV and its components then move as required to simulate an actual mission. Using kinematics and dynamics based on Newton-Euler equations [Fu 87], the final Cartesian coordinates for the system can be continuously determined.

Within the simulation, each sensor software module has a slot containing the measurements that would be obtained by that sensor in the actual navigation system. This simulation allows changing the parameters of the AUV missions and sampling a number of missions. By running a new simulation, an estimate of the resulting error can quickly and easily be determined. A 2D graphical representation of the AUV is included as a way of visually to demonstrate the approximate error of the system. This simulation can also be

tailored to support different types of AUVs using different types of sensors. Simulation details are explained in Chapter VII.

F. SUMMARY

Many AUV missions could be accomplished using an integrated navigation system combining GPS and INS. Similar systems in other applications have been demonstrated to have superior GPS signal acquisition and reacquisition performance whenever loss of lock occurs, resulting in improved survivability in hostile environments and smaller navigation errors. This research continues an ongoing experimental study pertaining to the development of such a system and the associated problems. The current system under evaluation is of small physical size and relatively low cost. The IMU selected is representative and has limited accuracy, so additional water-speed and magnetic heading information is required. This means that accelerometers are used mainly to derive low frequency attitude information, and are not utilized for velocity or position estimation over long periods.

The availability of differential GPS in open-ocean tests in Monterey Bay will allow the experimental choice of navigation filter gains to accurately assess overall system performance in a variety of sea states and for various operational scenarios. The research goal of this thesis is to produce test results and qualitative error estimates which indicate that submerged navigation accuracy comparable to GPS surface navigation is attainable.

IV. GPS/INS HARDWARE INTEGRATION

A. INTRODUCTION

Figure 1 shows a block diagram of the hardware assembled for at-sea testing of the SANS system concept. Figure 2 presents a photograph of the major components of the corresponding physical system. The towfish was designed and built by Russ Whalen. The 10 Hz filter was built and tested by Walter Schubert [Schubert 95]. Comparison of Figure 1 to the system described in [Kwak 93] reveals a number of differences. First of all, to enable experiments using a towfish rather than an AUV, the SANS system has been broken into two subsystems in which a minimum number of components have been placed in the towfish itself and the remainder are in the towing vessel. This results in a smaller towfish, with reduced power requirements and also allows for human monitoring and interaction during the course of an experiment. When this version of SANS is integrated into Phoenix (or any other AUV), the modems and towfish data logging computer shown on Figure 1 will no longer be needed, and the computer in the towing vessel will be replaced by the AUV onboard navigation computer.

Other differences relative to [Kwak 93] include replacement of the depth cell used for pop-up navigation by a water-speed sensor. This is because depth rate cannot be used for water speed estimation during surfaced navigation. Naturally, a complete SANS would include both sensors to enable both navigation modes. Additionally, estimation of water speed from depth rate by deliberate "porpoising" (periodic excursions in the vertical plane) during submerged navigation may be useful. It is expected that this possibility will be investigated after SANS is installed in the Phoenix. Another change to the earlier SANS concept is the replacement of rotating gyros by miniaturized vibratory angular-rate sensors for improved reliability and to eliminate AUV maneuvering limits imposed by rotating gyros [Kwak 93]. These sensors are packaged together with three precision linear accelerometers in the integrated IMU shown in Figure 2 [Systron-Donner 94].

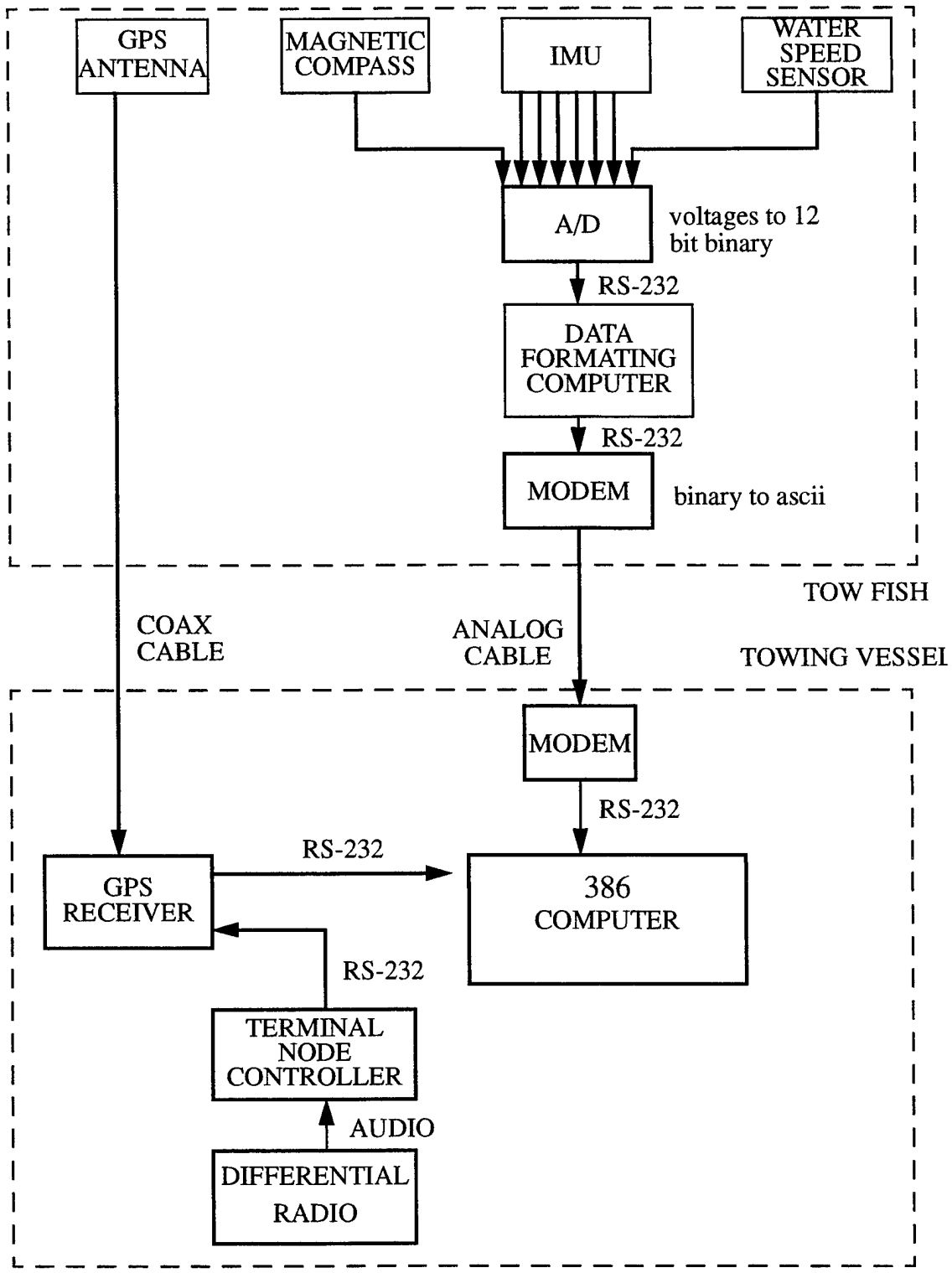

Figure 1: GPS/INS Hardware Integration

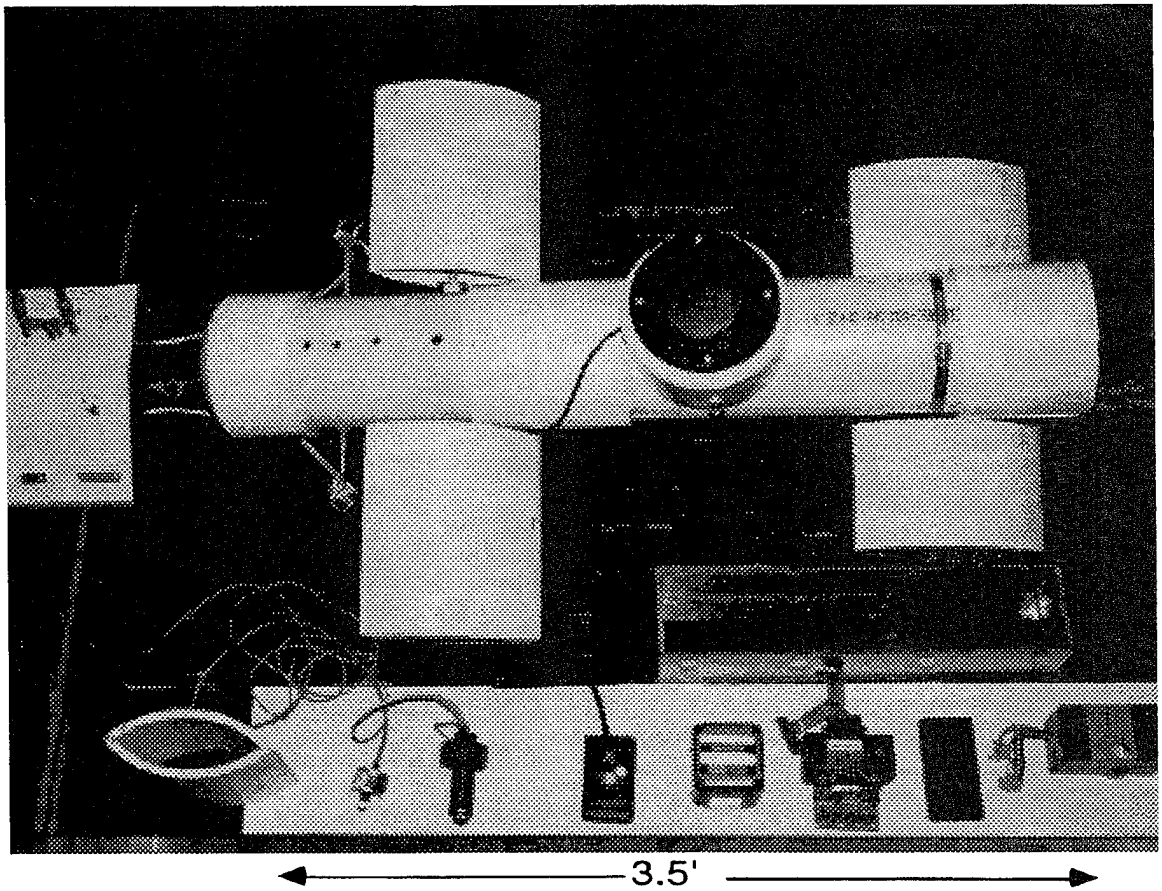

Figure 2: SANS and Towfish Components

During testing of this system, an Intel 386 based laptop PC was used for data collection. Processing of test data was accomplished off-line with a Sun 4 workstation. When SANS is installed in the Phoenix, an onboard notebook workstation (Sun Voyager or similar system) will replace both of these computers for real-time navigation. This computer will also perform other mission control functions as described in [Byrnes 93, Healey 95] while vehicle control will continue to be accomplished by the current Phoenix OS-9 system hosted on a Gespac 68030 computer running execution level software [Healey 93, 95] [Brutzman 94].

B. HARDWARE DESCRIPTION

1. Data Logging Computer

The Tattletale Model 5F-LCD computer was specifically designed for portable embedded applications requiring small size, low cost and high reliability. In order to conserve power, the Tattletale is equipped with two different modes of operation: sleep and run. The system's typical current drain is a strong function of the program that it is required to run, with the sleep mode only requiring 2.7 mA and a peak current rating of 20 mA. The Tattletale is equipped with a 12-bit analog-to-digital (A-D) converter which is capable of handling eight analog signals at a maximum sampling rate of 1600 Hz. The system also has a 480 KB data storage capacity and a 32 KB Flash EEPROM capacity for user programs [Tattletale 94a].

The Tattletale was programmed in TxBASIC to multiplex the six outputs from the Systron-Donner IMU, the water speed sensor and the compass. The software code described in [Schubert 95] was burned into the flash EEPROM and the program repeatedly fetched all eight A/D channels, using two bytes per sample, until 128 bytes of data was collected. The 128 byte packet was then transmitted via the XMODEM protocol to the towing vessel for processing. XMODEM was used due to easy availability on both units and since it includes both error correction and flow control.

2. Inertial Measuring Unit

The inertial navigation component of the AUV Phoenix was provided by a Systron-Donner Model MP-GCCCQAAB-100 "MotionPak" inertial sensing unit, pictured in Figure 3. This self-contained unit provides analog measurements in three orthogonal axes of both specific force and angular velocity. It consists of a cluster of three accelerometers and three "Gyrochip" angular rate sensors. General specifications are shown in Table 2. Accelerometer specifications and angular rate specifications are shown in Table 3 and Table 4 respectively.

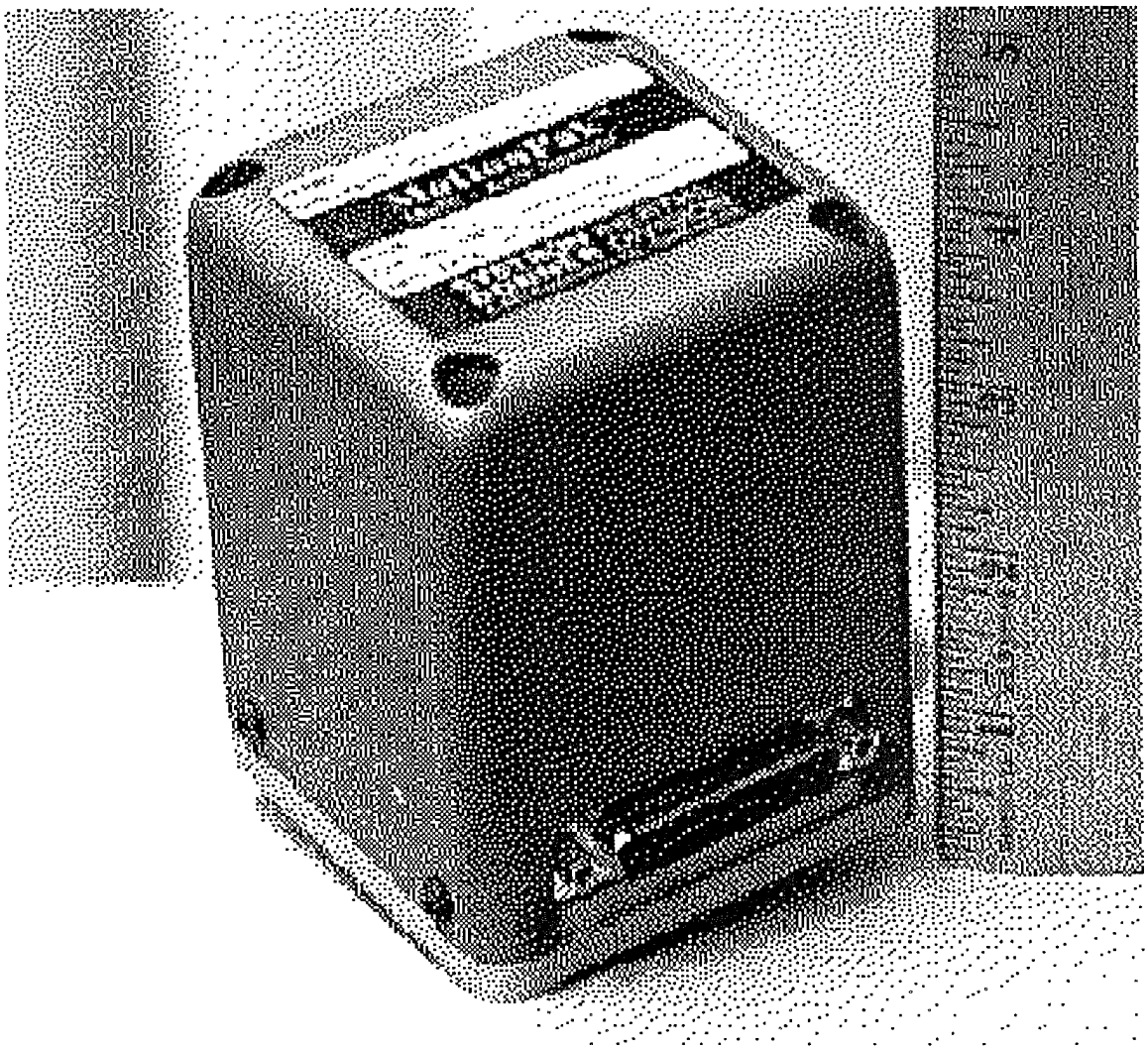

Figure 3: Systron-Donner Inertial Measuring Unit

Parameter	*Units*	*Range*
Input Voltage	DC Volts	+15, -15
Input Current	Amps	+0.246, -0.196
Temp. Range	degrees C	-40, 80
Weight	grams	912
Temp. Sensor	μ A/deg k	1.0

TABLE 2: MotionPak general specification [Systron-Donner 94]

Parameter	*Units*	*x-axis*	*y-axis*	*z-axis*
Range	g	1	1	2
Scale Factor	V/g	7.469	7.478	3.727
Scale Factor Temp. Coefficient	%/deg C	0.001	-0.002	-0.001
Bias	mg	-2.447	4.570	-0.586
Bias Temp. Coefficient	μ g/deg C	-47	-66	-48
Sensitivity	μ g	10	10	10
Bandwidth (to -90 deg phase)	Hz	797	757	901
Output Impedance	ohms	2464	2494	1177

TABLE 3: MotionPak accelerometer specifications [Systron-Donner 94]

Parameter	*Units*	*x-axis*	*y-axis*	*z-axis*
Range	deg/sec	50	50	50
Scale Factor	mV/deg/sec	50.151	49.906	50.242
Scale Factor Temp. Coefficient	%/deg C	0.03	0.03	0.03
Bias	deg/sec	-0.06	0.23	0.23
Bias Temp. Coefficient	deg/sec P-P	3	3	3
Sensitivity	deg/sec	0.002	0.002	0.002
Alignment	degrees	0.26	0.41	0.34
Noise	deg/sec/$\sqrt{Hz}$	0.008	0.009	0.008
Bandwidth (to -90 deg phase)	Hz	70	71	71

TABLE 4: MotionPak angular rate sensor specification [Systron-Donner 94]

3. Other Components

The GPS receiver used is the Motorola PVT6 receiver [Motorola 93a] which incorporates a Differential GPS (DGPS) capability and is able to track up to six satellites simultaneously. It can provide position accuracy of better than 25 meters spherical error probable (SEP) without Selective Availability (SA) and 100 meters (SEP) with SA on. Typical Time-To-First-Fix (TTFF) is 60 seconds with a reacquisition time of less than four seconds when the antenna has been obscured for up to 60 seconds [Motorola 93a]. [Norton 94] demonstrated that under normal operating conditions this receiver is capable of meeting the accuracy and time requirements of the SANS project. [Norton 94] also demonstrated this unit will perform well when using an antenna that is located on or near the sea surface as is necessary during a clandestine mission.

Each of the six output channels of the IMU are externally filtered by an active analog anti-aliasing filter with a bandwidth of 10 Hz. The filter is also used to convert the two-

sided IMU analog output to a single-sided signal within the 0 to 5 volt range of the A to D converter [Schubert 95].

The modems used were Pocket Peripherals Model PM14400FX [Practical 92] operating at a 9600 baud data rate. Communication from the towfish to the towing vessel was via a 100 ft analog cable using XMODEM protocol. This had the consequence that 128 byte packets were transmitted at approximately a five Hz rate to the laptop computer in the towing vessel. Each of these packets contained eight samples of each of the eight inputs to the towfish A to D converter as shown in Figure 7. Thus an average sampling rate of 40 Hz was achieved for each of these signals. This represents a two times oversampling of the 10 Hz bandwidth analog signals, thereby ensuring that noise aliasing was not significant in any subsequent digital processing.

As described in [Kwak 93] and [Norton 94], the magnetic compass used is a KVC C100. The water speed sensor is a paddlewheel type used for small boat applications [Tritech 95] and is essentially a four-pole rotor, single-stator alternator. AC signal frequency and voltage are proportional to sensed speed. These, and other system components shown in Figure 2, were selected based on proven technology and performed well in the SANS environment.

C. SUMMARY

The interim SANS design described in [Norton 94] is the basis for the system described in this section. The research of this thesis explores replacing the single linear accelerometer used in [Norton 94] with a three-axis IMU. The information provided by the IMU (filtered with a 10 Hz anti-aliasing filter), coupled with waterspeed and heading information, is multiplexed through a 12-bit A -D converter prior to being transmitted to the towing vessel via modem. The hardware for this version of the SANS was chosen to comply with the requirements set forth in [Kwak 93]. Even though there are many possible choices of hardware for each of the components in Figure 2, trade-offs between accuracy, size, power requirements, and cost must be considered. As further advances in

miniaturization are made, accuracy will continue to increase while price and size decrease, thus making it easier to meet the challenges of the SANS baseline requirements.

V. SOFTWARE DEVELOPMENT

A. INTRODUCTION

The purpose of the SANS software is to utilize IMU, heading, and water-speed information to implement an INS, and then integrate this with GPS information into a single system which can produce continuously accurate navigational information in real time. The INS is implemented using Kalman filtering techniques in which differential GPS fixes are treated as 'error-free data' allowing periodic reinitialization of the INS to correct accumulated errors and develop error biases. Both GPS and INS data are logged in raw form for postprocessing. In addition, each position fix is logged to a script file for postmission plotting as is attitude information.

B. SOFTWARE DESCRIPTION

This implementation continues to use object-oriented paradigms as discussed in [Stevens 93]. However, the increased complexity of this experiment, which involved navigation data from two separate sources and multiple serial ports, called for a new implementation as opposed to adapting previous SANS software. Where previous implementations of the SANS [Norton 94, Stevens 93] were done with Ada objects and assembly language routines to carry out low-level tasks, this implementation utilizes C++ objects to carry out all software operations. Use of a single language implementation simplified interfaces between software and hardware objects. The software is designed for use on a IBM-compatible personal computer with a 386SX/33Mhz processor using the Borland version 3.1 C^{++} compiler under DOS 5.0.

Figure 4 shows the SANS software objects and the types of data that are passed from one to another. The tasks preformed by the SANS software can be divided into two basic categories. The primary tasks are related to calculating the current position. These include processing incoming GPS data, IMU data, water-speed, and heading information, and integrating all information to obtain a navigational fix. These tasks are performed by the

GPS, INS and Navigator software objects respectively. Source code for these objects can be found in Appendix A. The secondary tasks involve hardware interfacing, communications, data filtering and unit conversion. The source code for these objects can be found in Appendix B. These basic but crucial tasks are handled by the Sampler, Buffer and Serial Port objects. The *main program* as illustrated in Figure 4 serves only to drive the other objects by continually querying the navigator for position updates.

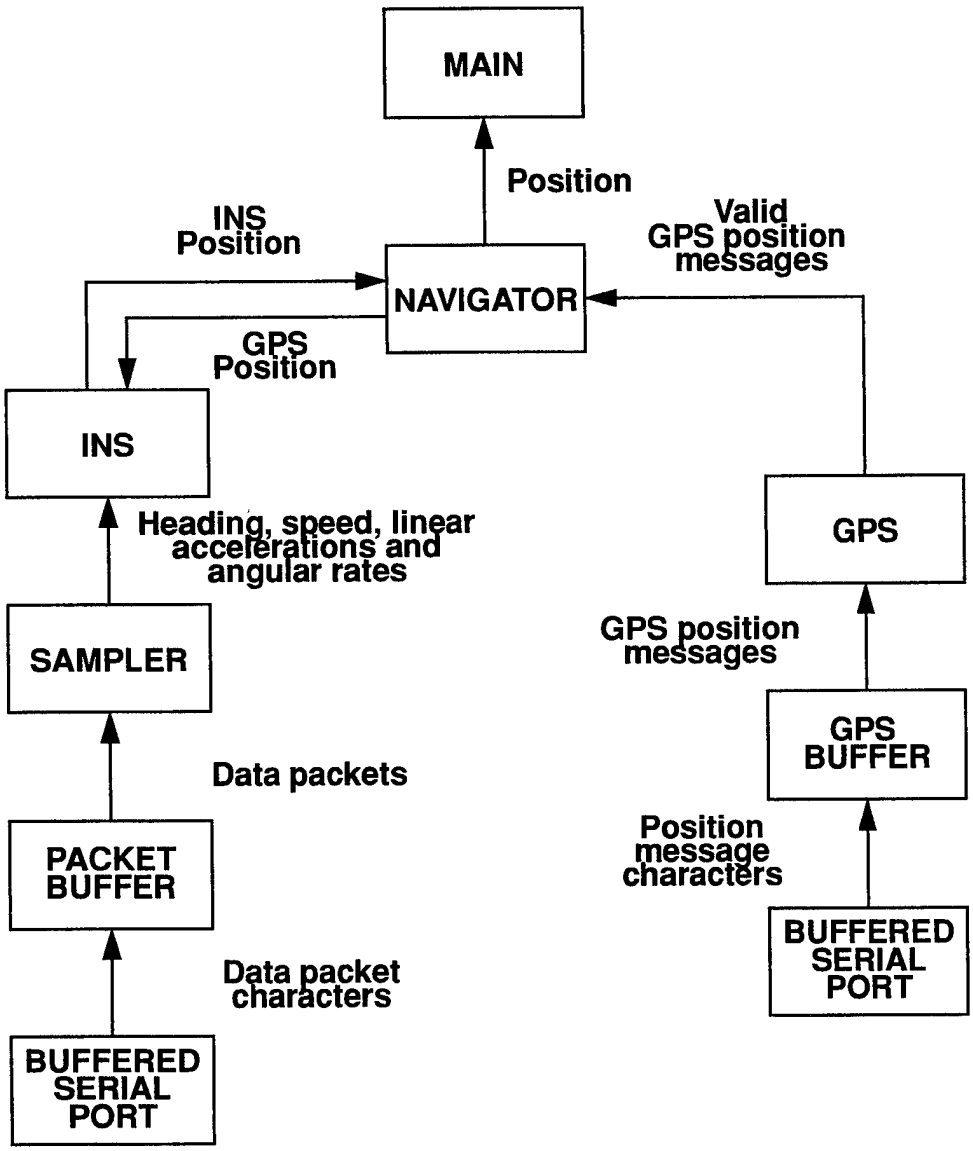

Figure 4: SANS software objects and data flow

In addition to the software outlined above a small program was implemented in TxBasic for execution by the 5F-LCD tattletale. This code is not considered part of the SANS software. The Tattletale TxBacsic program multiplexes the six outputs from the Systron-Donner IMU, the water speed sensor output and the compass output. Program code is included in Appendix E.

1. Navigator

The navigator class acts as coordinator of all navigational information. As such it determines which source is currently providing the best information, converts various position formats from one to another, and instantiates the GPS and INS objects. The interface to the object is made up of two public methods.

The first method of the navigator (initializeNavigator) initializes the navigator, preparing it to begin providing the current position upon request. This method obtains an initial GPS fix for use as the origin of the grid used by the INS object to specify positions, and calls the initialization method of the INS.

The second navigator method (navPosit) drives both the GPS and INS objects and provides the best estimate of the navigator of current position in hours, minutes, seconds and milliseconds of latitude and longitude. Each time the method is invoked, it interfaces with the GPS and INS objects to determine if none, one, or both have an updated estimate of the current position. If no update is available, the navigator returns a negative reply itself indicating that it can not provide a position update. If only INS information is available, it is converted and returned as the current estimated position. Whenever GPS information is available, it overrides the INS estimate of position and is converted and returned as the current position. GPS information is also passed to the INS object for reinitialization and error estimation purposes. All position information received by the navigator is written to a data file in raw format, as well as to a summarizing script file giving the estimated latitude and longitude in milliseconds of arc.

The navigator deals with three different position formats. GPS positions from the Motorola receiver are initially obtained entirely in latitude/longitude milliseconds. INS positions are expressed in x-y grid coordinates based upon a navigator-stored origin. The positions produced by the navigator itself are expressed in hours, minutes and seconds of latitude and longitude. In addition, GPS positions must be converted to grid coordinates prior to utilization by the INS. A total of four methods are used to convert from one format to another. Figure 5 illustrates uses and conversions of the different position formats.

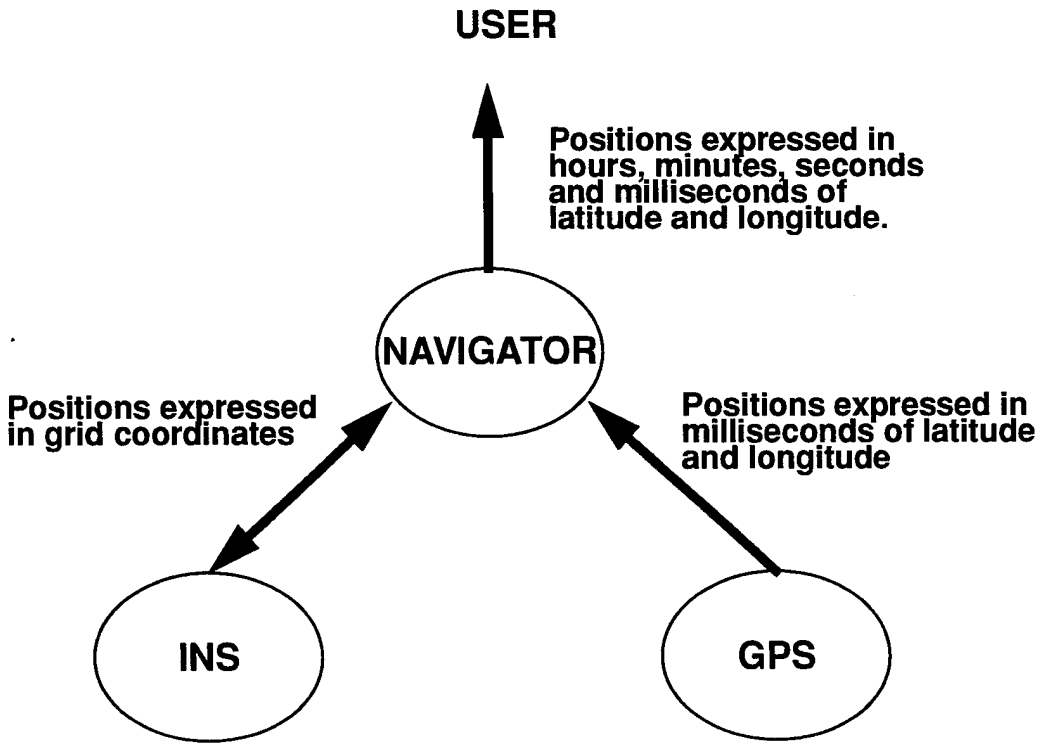

Figure 5: Navigation position format utilization.

2. GPS

GPS class objects obtain GPS position messages in the Motorola propriety format (@@Ba) and insure their validity prior to providing the information [Motorola 93a]. The object also instantiates the gps buffer and serial port objects needed to communicate with the GPS receiver via a PC serial port. Interface to the object is provided by a single method (gpsPosition). This method checks for the arrival of new messages. If one is available, a checksum is performed on it and a determination is made regarding the number of satellites in view of the GPS receiver when the received message was sent. If less than three satellites were used to produce the fix, it is considered invalid. This requirement is made to insure that GPS information will not be used while the SANS is submerged and unreliable.

3. INS

The INS class implements the inertial navigation portion of the SANS. It is the most complex class in the software. The interface consists of three public methods. Each is directly involved in the implementation of a nine-state Kalman filter. The primary method (insPosition) combines all sensor information and uses the Kalman filter described in the following paragraphs to produce a dead reckoning position estimate. The other methods support the primary method by performing special one-time or periodic operations. Initialization of the INS is performed by a method (insSetUp) which sets the INS posture at the grid coordinate origin, sets an initial heading and speed, and marks the beginning of the first integration intervals. The last public method of the class (correctPosition) inputs GPS information to reinitialize the INS position while determining a current and error correction bias. The INS class instantiates a Sampler object from which it obtains all sensor data except for GPS position fixes. It also records attitude information to a script file for post-processing and plotting.

Figure 6 is a data flow diagram for the SANS filter design. The nine state variables are the outputs of each of the three integrators and the summer of the Kalman filter and are shown in Table 5. The seven continuous-time state components of this filter consist of three

Euler angles (Φ, θ, ψ), two horizontal velocities ($\dot{x}_e$, $\dot{y}_e$) and two horizontal positions (x_e , y_e). In the actual SANS digital filter implementation, the continuous-time integration is approximated by numerical integration, so in this sense the seven "continuous-time" state components are "discrete- time" state values. This is necessary due to limitations placed on the minimum integration sampling times by the computer and A-D hardware characteristics. The two discrete-time state components ($\dot{x}_c$, $\dot{y}_c$) are composed of estimated ocean current in the East and North directions. Their discrete nature is due to diving and wave action which results in intermittent GPS signal reception. Thus the two "discrete-time" states are updated aperiodically as is characteristic of discrete event dynamic systems [Ramadage 89]. This being the case, it is difficult to apply Kalman filter theory to obtain optimal time-varying values for the gain matrices K_i shown in this figure. Instead, constant gains were computed initially from bandwidth and steady-state error considerations.

The continuous state part of Figure 6 shows that the Euler angle and linear velocity outputs are fed back to the corresponding integrator inputs. Thus if the gain matrices K_1, K_2, and K_3 are all diagonal, each of these integrators is in fact a low pass filter for each of its inputs. This is done to prevent unlimited growth of state estimates in the presence of unmodeled bias errors in state derivative inputs to integrators. Each integrator is also furnished with an independent source of low frequency information to correct for long-term decay of state estimates resulting from this feedback. This approach is usually referred to as "complementary" filtering, or sometimes as "crossover" filtering.

The sources of low frequency information include the accelerations sensed by the accelerometers ($\ddot{x}_a$, $\ddot{y}_a$, $\ddot{z}_a$), the magnetic compass readings (Ψ_c) and the water-speed (u_w). The accelerometer data in this case is utilized in a manner similar to inclinometer readings. This provides information regarding how much of the specific force felt in each axis is due to gravity.

In addition to filtering, the IMU readings require other correction or conversion. The specific force readings of the accelerometers are translated into the accelerations $\ddot{x}$, $\ddot{y}$, $\ddot{z}$

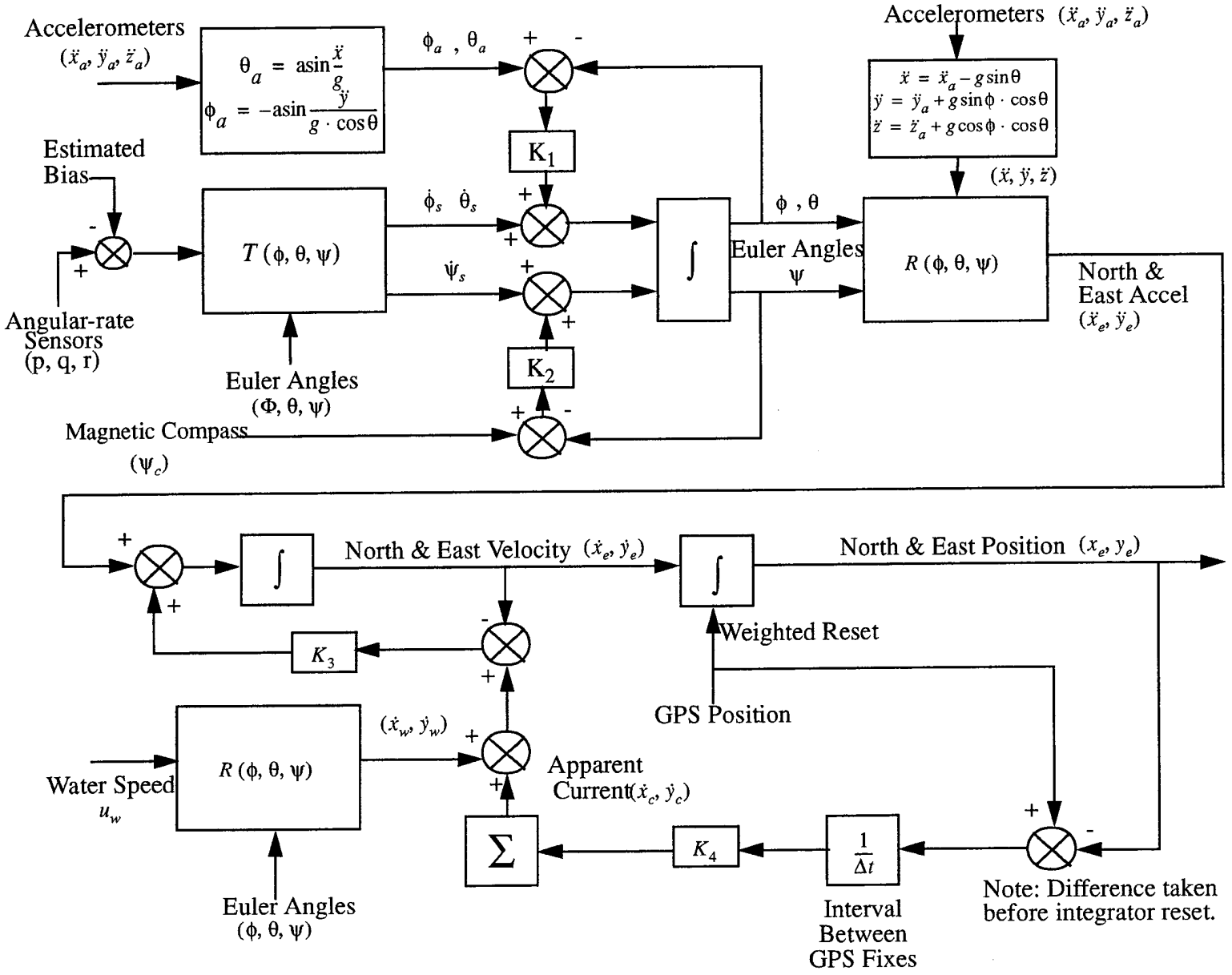

Figure 6: Nine-state velocity-aided navigation filter

prior to rotation to $\ddot{x}_e$, $\ddot{y}_e$, $\ddot{z}_e$. The angular rate sensor readings are bias corrected prior translation into Euler rates in order to correct for rate sensor drift. These biases C_p , C_q , and C_r, are continually updated by a low-pass filtering process using the equations (5-1), (5-2), and (5-3).

$$C_p = K(C_p) + (1-K)p_{uncorrected} \tag{5-1}$$

$$C_q = K(C_q) + (1-K)q_{uncorrected} \tag{5-2}$$

$$C_r = K(C_r) + (1-K)r_{uncorrected} \tag{5-3}$$

They are subtracted from the angular rate sensor readings. K is a weighting factor, with values typically ranging from 0.99 to 0.9999.

In the SANS filter design, complete confidence is placed in the precision of GPS. Matrix K_4 is therefore set to unity causing the integration of linear velocity (posture) to be reinitialized each time a GPS fix is obtained. Further discussion of the software filter design and gains can be found in [McGhee 95].

Euler Angles	Φ, θ, ψ
North & East Velocity	$\dot{x}_e$, $\dot{y}_e$
North & East Position	x_e , y_e
Apparent Ocean Current	$\dot{x}_c$, $\dot{y}_c$

Table 5: State variables of the Kalman filter

4. Sampler

The Sampler prepares raw IMU, heading and water speed data for use by the INS. This preparation includes filtering, unit conversion and time stamping. The Sampler interface consists of a single method (getSample) which controls the data formatting and returns a formatted sample if valid raw data is available and a negative response otherwise. All Sampler methods are dependent on the format of the raw data packets received by the

packetBuffer class. Therefore, the sampler will require extensive changes if this format is altered.

The packets processed by the Sampler are received via XMODEM protocol by the packetBuffer object. Figure 7 illustrates a typical packet. Each packet contains 132 bytes, 4 of which are header and 128 of which are data. The first three bytes, in order, consist of the "start of header" character (SOH 0x01), the packet sequence number and the complement of the packet sequence number. These are followed by the data bytes and a one byte checksum. The data bytes are divided into eight closely timed samples. Each of these samples consists of eight two-byte integers. The first six integers in each sample are IMU outputs. The first three are linear accelerations ($\ddot{x}, \ddot{y}, \ddot{z}$) and the next three are angular rates ($\dot{\phi}, \dot{\theta}, \dot{\psi}$). The seventh integer in each sample is output by the water speed sensor and the last integer in each of the eight samples contained in a packet is output from the compass. The integers contained in a sample are digital measurements of analog voltages output by the SANS sensors.

The first action taken by the Sampler when a packet is received is to time stamp it. Since the time difference between the eight samples contained in a single message packet is relatively small, the Sampler object then respectively averages the eight corresponding data variables contained in a packet. The averaged measurements which result represent a low-pass filtering of the eight samples. Once these eight filtered measurements are obtained they are converted from voltages to units which are usable by the INS object (i.e. feet and radians). Finally, each of the measurements is checked to ensure that it is within the limits of the sensor from which it came. If any values fall outside the capabilities of the sensor from which it came, the entire packet is considered invalid and discarded.

5. Communication Objects

The Buffer and BufferedSerialPort classes perform the routine but necessary tasks of receiving individual characters via serial port and buffering them until they are used to

estimate the current position of the SANS. The buffer classes used are actually derived from a base buffer class and are specially designed to handle specific types of messages.

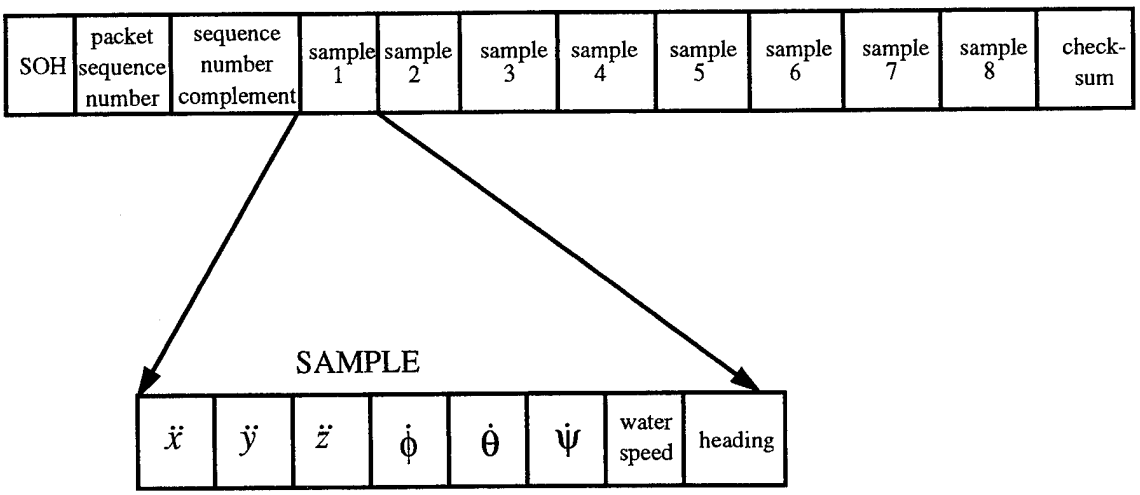

Figure 7: Modem packet format

The packetBuffer class handles packets of IMU, water speed and heading information. The gpsBuffer class handles GPS position messages in the Motorola proprietary format (@@Ba) [Motorola 93a]. The BufferedSerialPort class is derived from a base serial port class. These two classes set communications parameters, establish new interrupt service routine (ISR) vectors and initialize UARTs (Universal Asynchronous Receiver/ Transmitter) through which communications are conducted with both the GPS receiver and the tow fish. A portBank class controls the multiple instances of serialPort which are open simultaneously. This class is not illustrated in Figure 4.

The packetBuffer class implements a specialized XMODEM protocol used to communicate with the Tattletale in the towfish. Unlike a normal XMODEM implementation, each packet received is acknowledged as correct to the sender prior to inspection of its contents. This eliminates the requirement for the sender to re-transmit packets which have become time late. Instead the sender simply transmits an updated packet which contains more recent data. Only when an attempt is made to get a complete

packet from the buffer are its checksum and sequence numbers inspected. If either is found to be invalid the packet is discarded in the expectation that new data will soon be received.

The gpsBuffer class operates in a manner similar to packetBuffer class. Multiple messages are stored in circular buffers, but only the most recent arrival is considered when a request is made for updated information. As with the packetBuffer, the header and checksum are only inspected when a request is made for a message.

C. SUMMARY

The SANS software is designed to produce continuously accurate navigational information in real time. While submerged, IMU, heading and water-speed information are processed by the SANS Inertial Navigation System (INS) to produce a dead reckoning position estimation. This is integrated with DGPS information obtained during periodic surfacings using Kalman filtering techniques. The DGPS information resets the position of the INS. It is also used to generate an apparent current vector to correct future INS position estimates.

The software was implemented using object oriented paradigms. It was written in Borland version 3.1, C^{++} for use on a 386SX/33Mhz processor. The primary tasks of the software are estimation of current position and communication. The former is handled by the Navigator, INS, GPS, and Sampler objects. The later is accomplished by the GPS buffer, Packet buffer, Port bank, and Buffered Serial Port objects. The Buffer Serial Port and Buffer classes are very general and could be used in a variety of serial port communication applications.

VI. SYSTEM TESTING

A. INTRODUCTION

This chapter presents the test methods and experimental results of static laboratory testing and dynamic at-sea testing used to determine the functionality and accuracy of the towfish system. Bench testing was performed to ensure the entire system was functional prior to at-sea testing. In addition, the system was tested in its entirety by placing it on a wheeled cart and pushing it around a measured course. The final at-sea testing was performed by towing the towfish behind a boat in Monterey Bay while diving and surfacing the towfish at various intervals.

B. STATIC TESTING

1. IMU Static Tests

For cost and availability reasons, a single-sided 12 bit A-D converter [LTC 95] was selected for the breadboard SANS shown in Figure 1 and Figure 2. Figure 8 shows a sample of typical results obtained from bench testing of an accelerometer and a rate sensor. In this test, data from all six channels of the IMU were collected using a Tattletale data logging computer [Tattletale 94a]. As can be seen, the acceleration signal fluctuates an average of about ± one bit. This low level of accelerometer noise is due in part to the fact that each of the six output channels of the IMU is externally filtered by an active analog anti-aliasing filter with a bandwidth of 10 Hz. This circuit also converts the two-sided IMU analog output to a single-sided signal within the 0 to 5 volt range of the A-D converter [Schubert 95].

The x-axis (longitudinal) accelerometer signal shown in Figure 8 was obtained with the accelerometer lying on its side on a table. Thus the output should nominally be zero, which corresponds to the integer value 2048 for this A-D converter. It can be seen that the output is actually centered around 2028, which represents an apparent error of around one percent. However, this is not a correct analysis of the error effect. In fact, the output

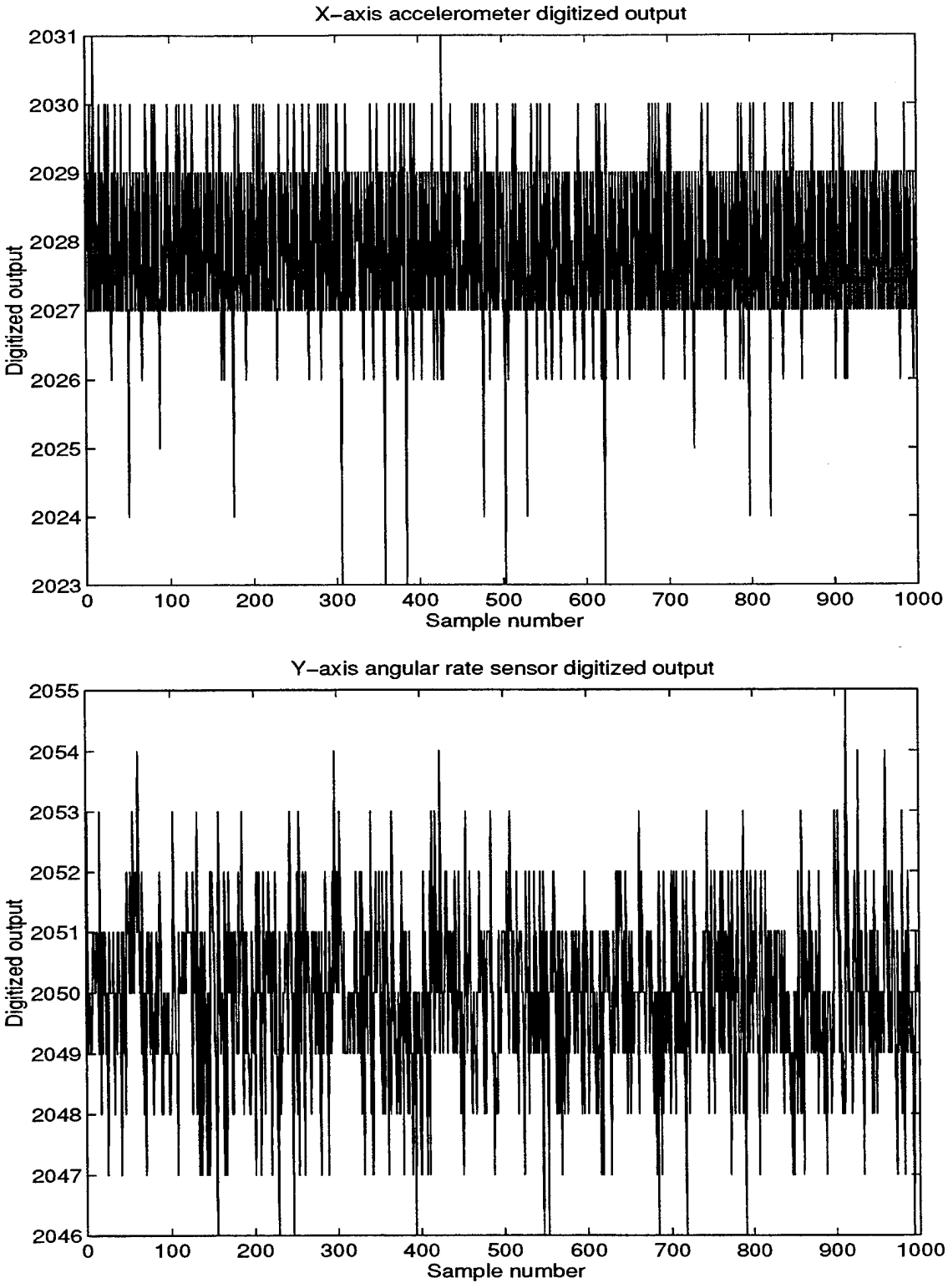

Figure 8: IMU bench test results(83 seconds at 12Hz data rate)

mean indicated could be due either to a tilt in the supporting surface or to an amplifier imbalance. Either of these effects would be eliminated in a prelaunch alignment and initialization phase before conducting a SANS mission. The real significance of Figure 8 is that limiting A to D precision to 12 bits seems to be justified, since it can be seen that typical IMU sensor noise reaches or exceeds the value of the least significant bit, at least with 10 Hz anti-aliasing filtering.

2. GPS Receiver Testing

As described in [Kwak 93] the Motorola PVT6 GPS receiver possesses generally desirable characteristics for the SANS system [Motorola 93a]. As can be seen in Figure 2, it is physically quite small. It also possesses a low power sleep mode with the time to first fix after one hour of power off typically on the order of 30 seconds [Kwak 93]. This long time is needed to acquire ephemeris (orbital) data from new satellites which may have come into view since the last GPS fix [Clynch 92]. Accuracy in latitude and longitude has been observed in static testing to be around 30 meters rms using the standard positioning service (SPS) [Kwak 93]. Figure 9 shows recent bench test results relating to raw GPS position estimates obtained with an antenna mounted on top of a five story building at the Naval Postgraduate School. Figure 9 also shows the improvement in horizontal position error (root sum square of orthogonal horizontal error components) which results from the use of differential mode using a Trimble RL base station about three km away at the Monterey Bay Aquarium Research Institute (MBARI) building in Pacific Grove. As can be seen, differential correction (DGPS) reduces rms radial position error to around 1.4 meters, corresponding to rms latitude and longitude errors of approximately 1 meter. These errors are generally consistent with the findings of an earlier, more comprehensive study of a variety of DGPS systems [Clynch 92].

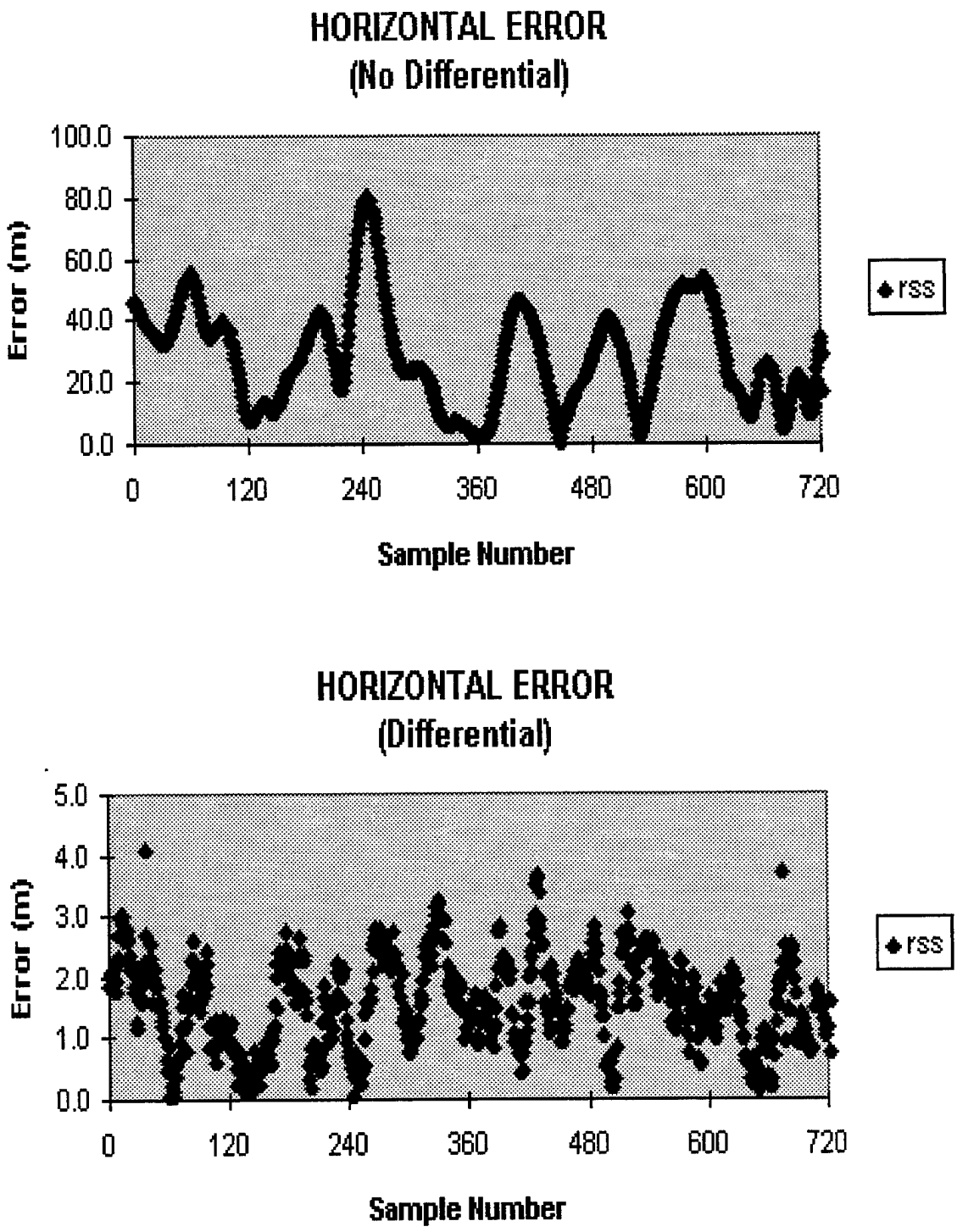

Figure 9: GPS bench test results(1 hour at 5 second intervals)

An important question relating to GPS navigation with the antenna essentially awash, as is the case for SANS, is whether or not satellite tracking can be maintained in the presence of wave action. To investigate this issue, [Norton 94] conducted bench tests in which a plastic pan of seawater submerged the GPS antenna. For satellites at small angles from the vertical, it was found that tracking could be maintained reliably for water depths of less than five mm [Norton 94]. For satellites nearer the horizon, it was found that the slant distance through the water was the limiting factor in determining reception, and that this distance also can be up to five mm without seriously affecting reception. Following these tests, the GPS antenna was mounted on an earlier simplified towfish [Norton 94], and subjected to short dives in which submergence typically lasted from two to five seconds. Results of these tests can be found in [Norton 94]. The results show that once satellite tracking is lost due to diving, the minimum of three satellites needed for surfaced navigation is typically regained in two to five seconds after surfacing. A fix is produced at that time if the satellite ephemeris data is available in the receiver memory for three or more of the satellites being tracked. These results coupled with additional test results reported earlier in this chapter, encourage us to believe that wave action will not present a serious problem to GPS reception for SANS.

3. Software Testing

Extensive static testing of the navigation software was conducted in order to establish proper functionality and to aid in determining appropriate Kalman filter gains. The Motorola receiver was set to obtain DGPS fixes at intervals of 1, 10, 20, 30, and 60 seconds in order to simulate the diving and surfacing of the towfish. For testing purposes the towfish was leveled in the roll and pitch attitudes and was placed so that the compass read 180 degrees magnetic. Figure 10 shows a representative bench test run with the Motorola receiver set to 60 seconds between fixes. Analysis of the data shows that, even with a full minute between fixes, the Kalman filter appears to be able to significantly reduce the position error over a relatively short period of time. The data also demonstrates that the

system is statically able to meet the 10 meter accuracy requirement of [Kwak 93] for a large majority of the time.

It should be noted that the above accuracy is accomplished through the development of a fictitious "apparent current", as shown in Figure 6, which compensates for other sources of static velocity error in the system. For the results shown in Figure 10, the steady state value of this current was around 0.059 knots North and 0.177 knots West (-0.177 East).

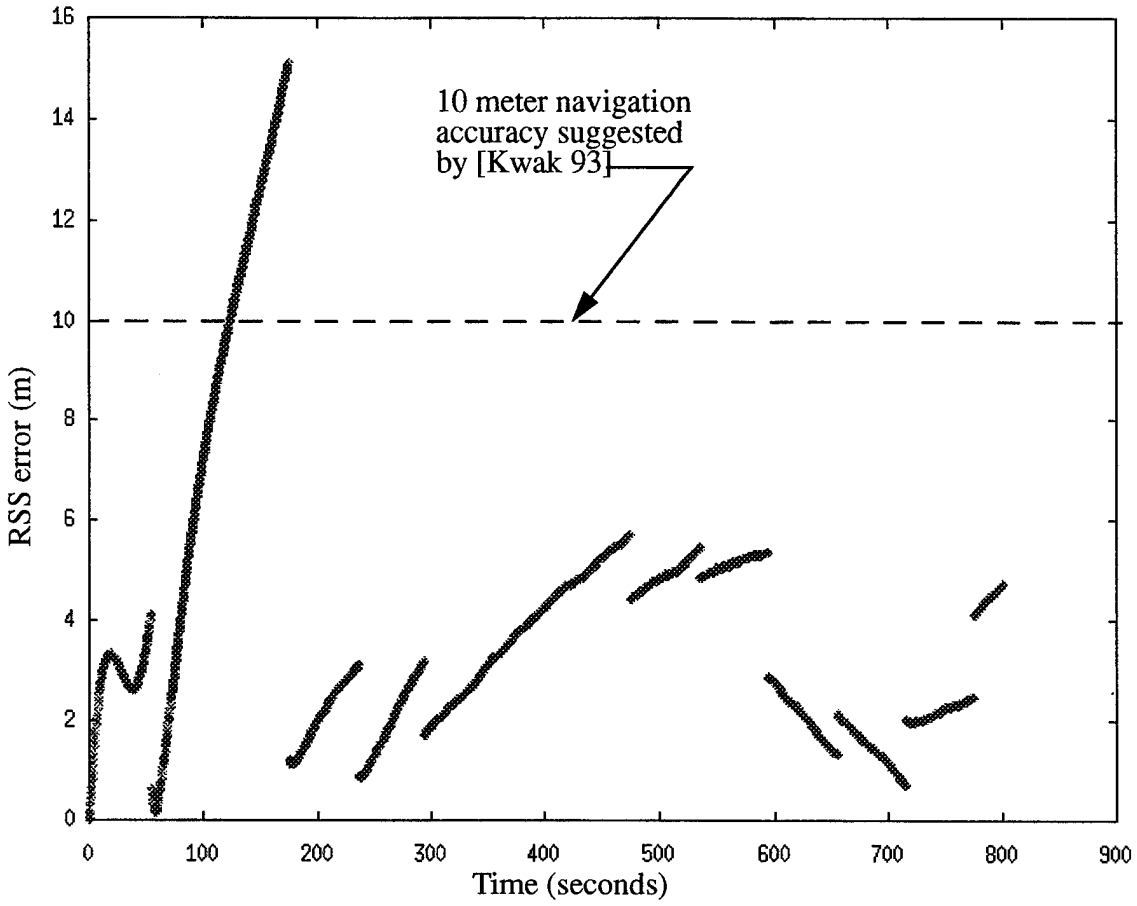

Figure 10: Bench test results with 60 seconds between DGPS fixes

Figure 11, Figure 12, and Figure 13 show the roll, pitch, and yaw attitude of the towfish during bench tests. The tests results were obtained with the towfish sitting as level as

possible with 60 seconds between DGPS fixes. The results show that the Kalman filter bias corrections to the rate sensor readings from the IMU were effective in eliminating rate sensor drift which is inherent in this type of sensor.

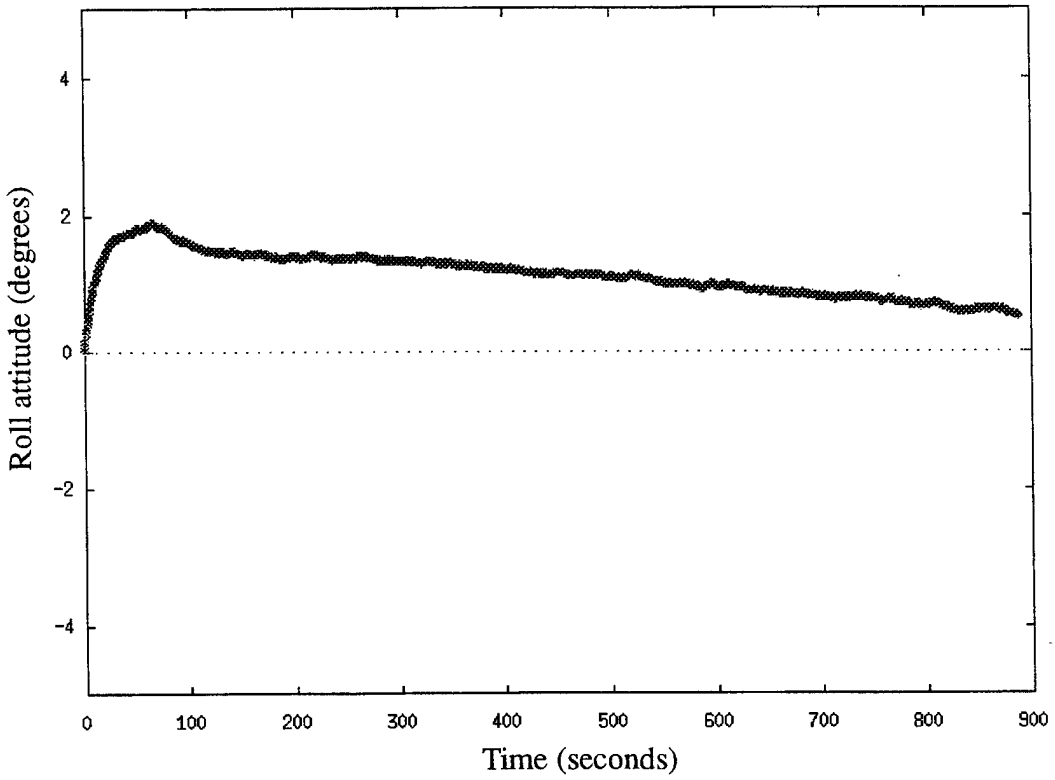

Figure 11: Bench test roll attitude

Further static testing was performed on the IMU sensor while in the towfish. The towfish was set up to be as level as possible for a period of two minutes, then the system was pitched nose-up to a 14^0 attitude for a period of two minutes and then returned to a level position for the final two minutes. The same test procedure was performed for the roll axis. Figure 14 and Figure 15 show the results of the IMU bench tests.

Figure 14 shows that there was a residual roll angle of approximately -0.5^0 (left roll). Analysis of the data shows that at approximately 120 seconds the roll attitude changed and eventually stabilized at a value which reflects a 14^0 change in attitude. The same effect can be seen at 240 seconds.

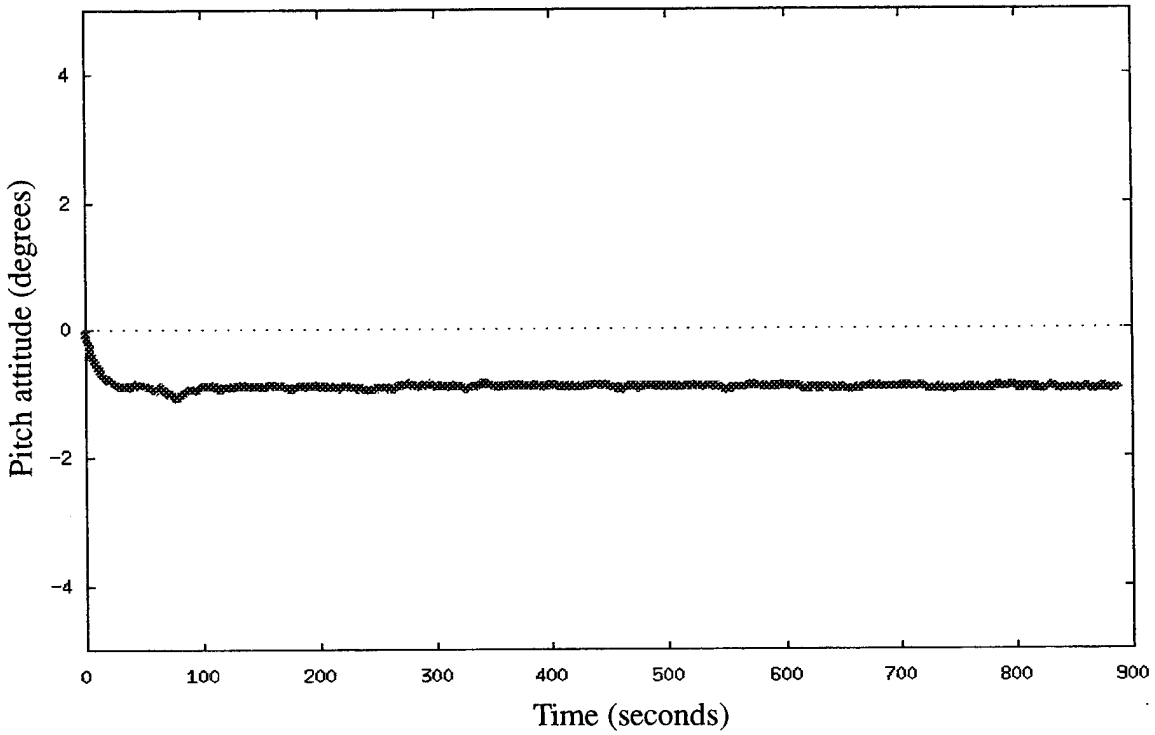

Figure 12: Bench test pitch attitude

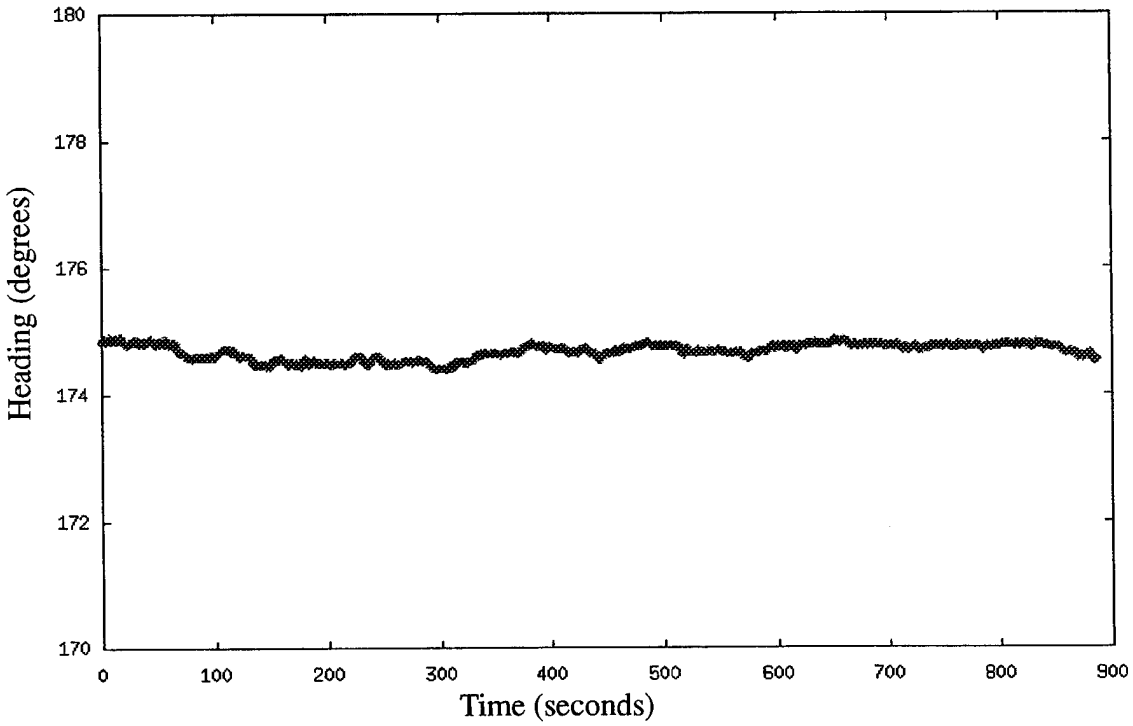

Figure 13: Bench test heading

Figure 15 shows that there was a residual pitch angle of approximately -1.5^0 (nose down). Analysis of the data shows that at approximately 130 seconds the pitch attitude changed and eventually stabilized at a value which reflects a 14^0 change in attitude. The same effect can be seen at 245 seconds.

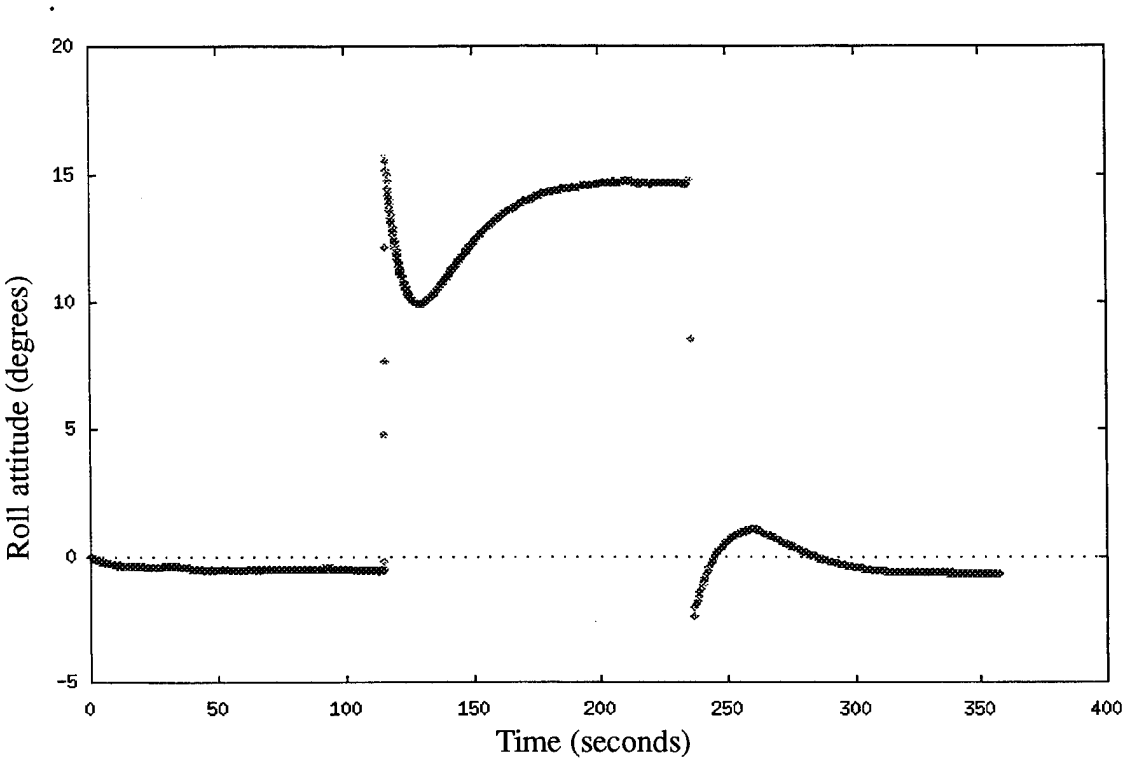

Figure 14: Roll attitude transient of 14 degrees

The large transients observable in Figure 14 and Figure 15 are undesirable and should not occur in a properly tuned filter. As of the time of completion of this thesis, however, a more appropriate set of input and feedback gains has not been found. This is an important area for further research on the SANS which should be undertaken as soon as possible. Until such tests are successfully completed, the possibility of a programming error in the SANS code must also be considered.

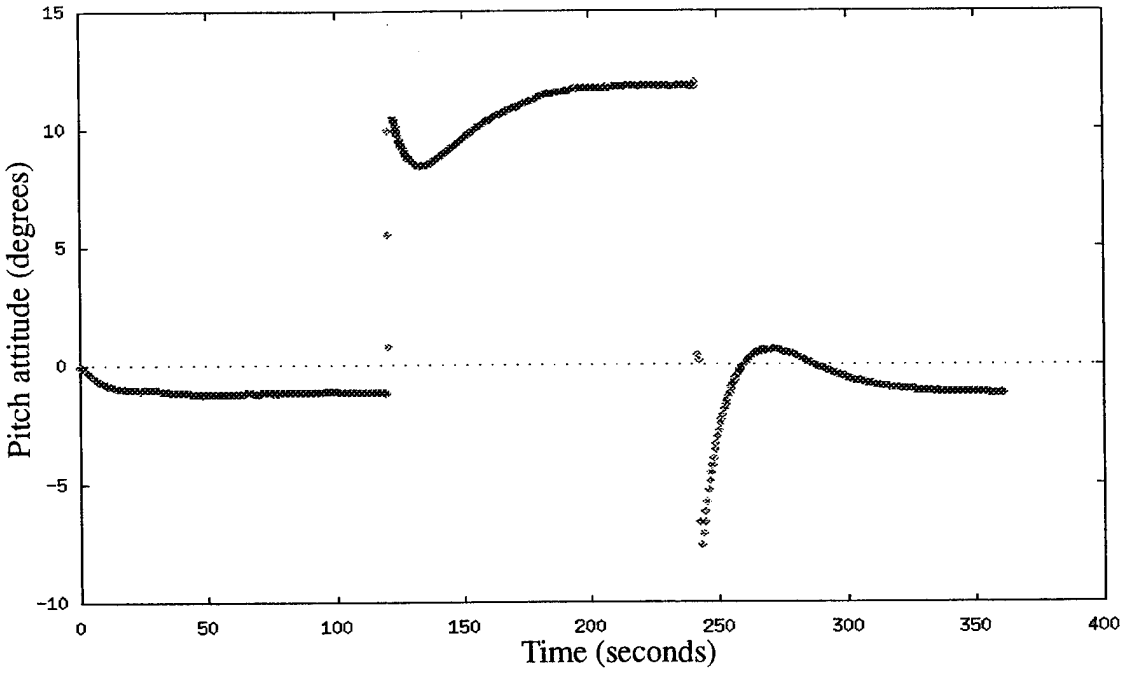

Figure 15: Pitch attitude transient of 14 degrees

C. CART TESTING

In order to ensure that the entire system was functional and that the software was sufficiently robust prior to proceeding to at-sea testing, it was necessary to assemble the entire system as it would be configured on the boat. This was accomplished by placing the system on a two-tiered wheeled cart as depicted in Figure 16. The cart and towfish system were then pushed around a surveyed course. Diving and surfacing of the towfish was simulated by setting the Motorola GPS control software to obtain a GPS fix at specific intervals. Typical intervals were 1, 10, 20, and 30 seconds between GPS fixes. Between each of these GPS fixes, the IMU was estimating and tracking position. As each subsequent GPS fix was obtained, the INS was reset to the DGPS position and the Kalman filter then used the fix to determine new biases. Typical results of this type of testing are shown in Figure 17. Another reason for performing the cart test prior to at-sea testing was to obtain initial data to aid in estimating values for the Kalman filter gains. Analysis of the data

showed that compass heading was reliable and should have a relatively large gain. The same reasoning applied to the water speed sensor. After several iterations it was determined that K_1, K_2, K_3, and K_4 should be set to 0.1, 0.6, 0.5, and 1.0 respectively. The fact that K4 is 1.0 implies that the DGPS fixes were used as truth data to reset the INS.

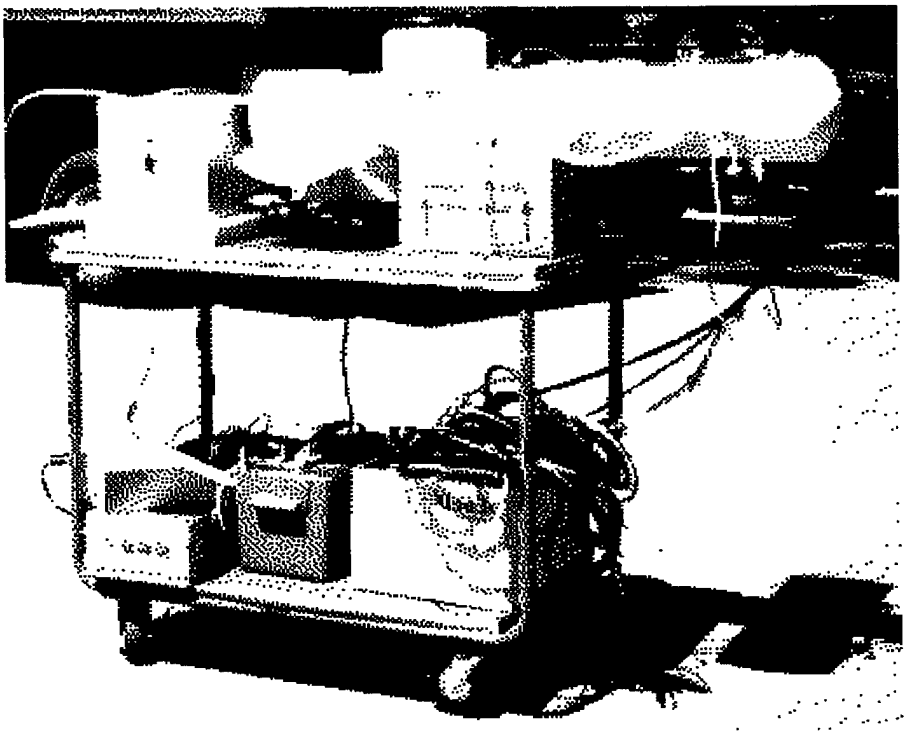

Figure 16: System configuration for cart testing

During the cart test no attempt was made to level the towfish and any results which show stabilized values other than zero can be attributed to this fact. Typical roll, pitch, and heading results are presented in Figure 18, Figure 19, and Figure 20 respectively. When the towfish program was started it was allowed 30 seconds to initialize prior to moving the cart. As the cart arrived at point 2 and point 3 the program was once again allowed 30 seconds to stabilize prior to continuing to the next point. Proceeding back to point 1 through point 3 was performed without stopping to stabilize the plot. As can be seen in Figure 18 there.

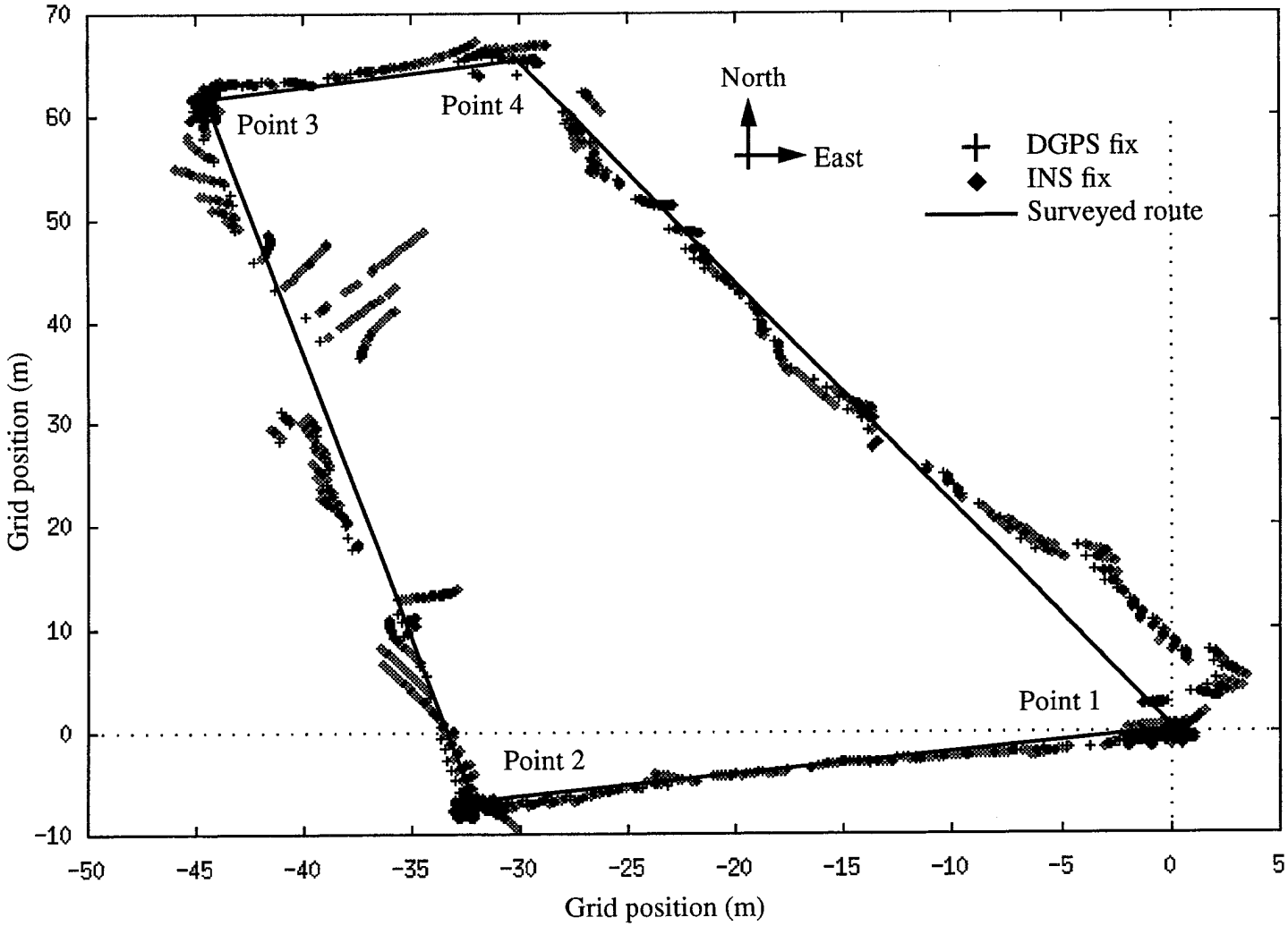

Figure 17: Typical cart test results (GPS fixes once per second)

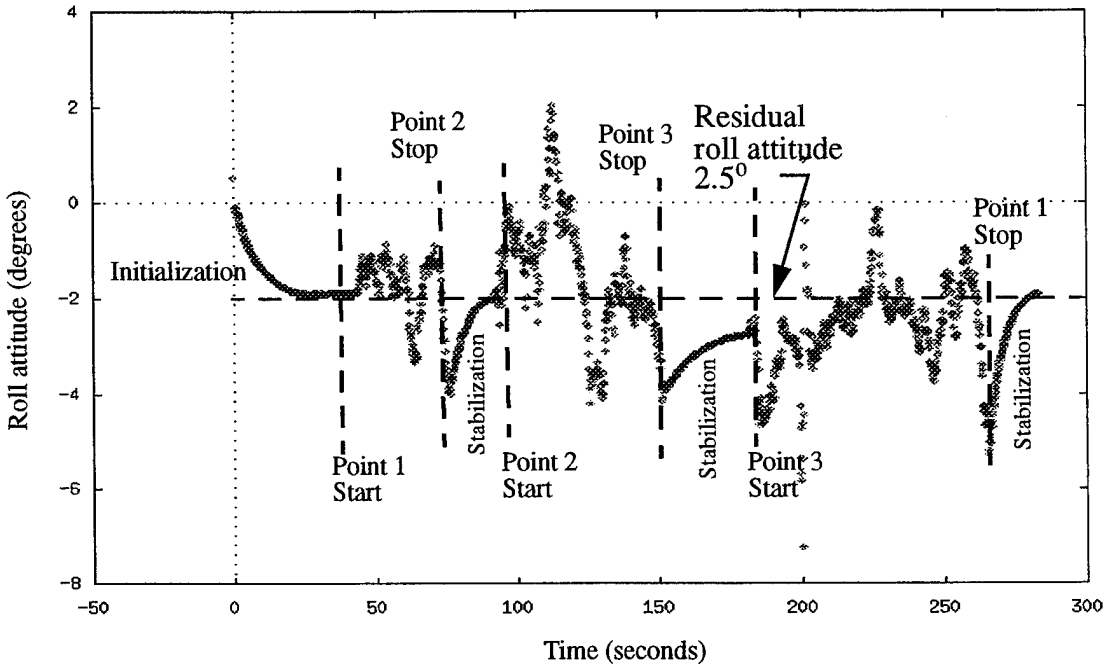

Figure 18: Roll attitude for cart test

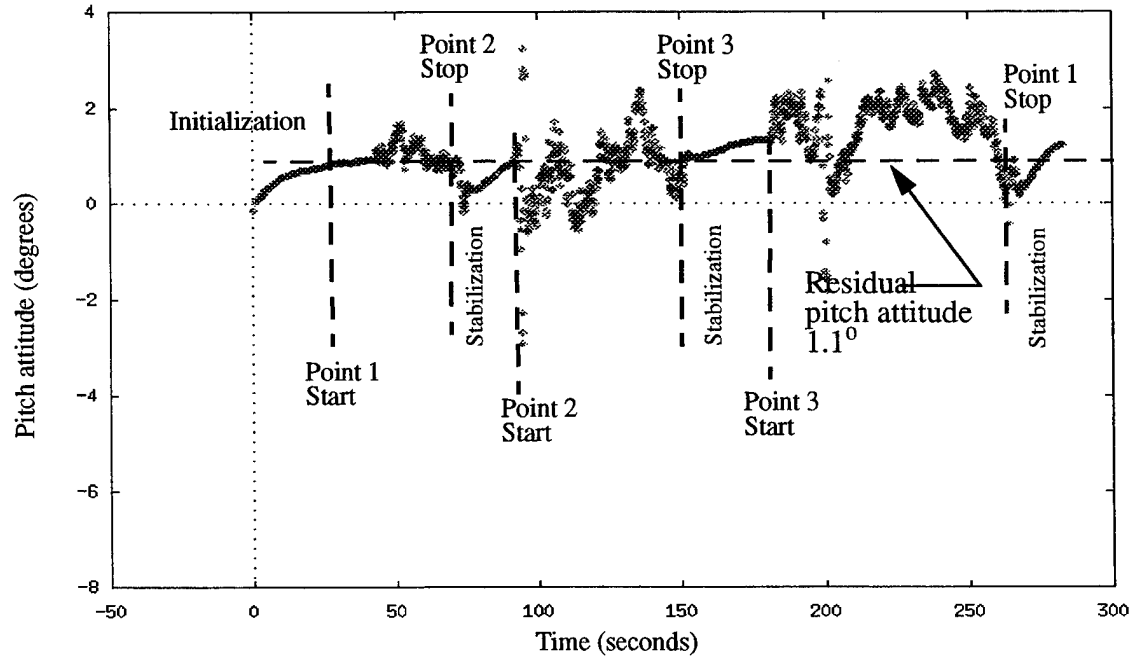

Figure 19: Pitch attitude for cart test

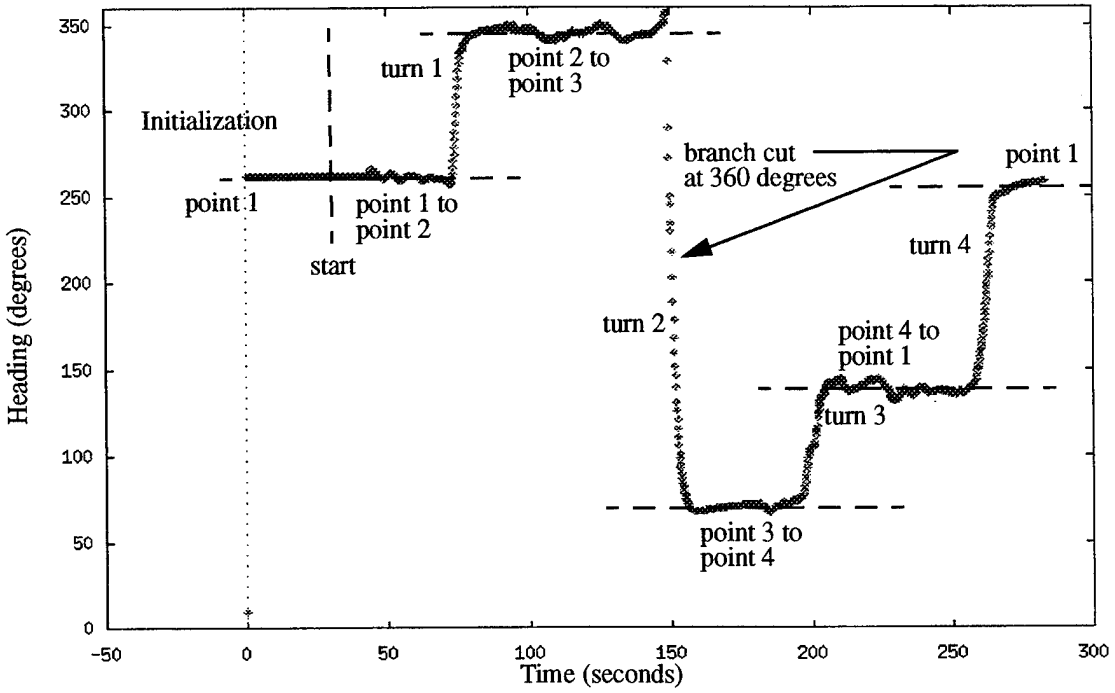

Figure 20: Heading for cart test

was a residual roll angle of approximately 2.5^0 which can be attributed to not leveling the towfish prior to the experiment. The same type of effect can be seen in Figure 19 which shows a residual pitch angle of approximately 1.1^0. Both figures show that the system appears to be reacting to the movement of the cart as it is pushed around the course. The figures also show that the system appears to working correctly since the roll and pitch values tend toward the residual values as soon as the cart is at rest.

Figure 20 shows the heading results from the cart testing. After the program was initialized, the cart was pushed in the direction of point 2. The data shows that the cart was turned approximately 90 degrees at each point which is a good correlation to the surveyed course shown in Figure 17. At point 3 the cart was turned in a direction which passed through north which produces the branch cut shown in the figure. Another factor which shows that the system appears to be working correctly is that after four turns the final heading stabilizes at the same value as the initialization heading.

D. AT-SEA TESTING

An at-sea test procedure was developed after establishing that the proposed mission requirements were feasible based on the results of the simulation, bench tests, and cart tests. The purpose of the at-sea test was to further demonstrate the ability of the GPS receiver to track satellites and obtain DGPS position information while the antenna was subjected to wave wash-over. The test was also used to verify the initial concepts of the Kalman filter techniques, as well as to obtain raw data for post-processing to be used in optimizing the Kalman filter gains.

The first step was to build a test vehicle that was able to adequately simulate the actions of an AUV in the performance of its mission. The test vehicle was required to submerge to a depth which was deep enough to obscure the antenna from satellite reception. Also, while on the surface the test vehicle had to ride high enough to reduce the susceptibility of the GPS antenna to wave effects. The vehicle also had to have the capability to be commanded to surface and submerge. The final version of the test vehicle is shown in Figure 2 and Figure 16. The vehicle was designed to be towed behind a boat with diving and surfacing controlled by pulling on control lines which moved the front set of elevators.This vehicle was designed and constructed by R. H. Whalen [Schubert 95].

The test was performed in the Monterey Bay in light seas with swells of three to four feet. Figure 21 shows the towfish during at-sea tests. The flat Motorola antenna that was supplied with the GPS receiver was used since it allows water to sit on top which is the worst case scenario. The antenna on the towfish was connected to the receiver on the boat via 60 feet of RG-213 coaxial cable where it was combined with the differential correction information being broadcast by MBARI. The eight channels of information that were being sampled by the Tattletale unit were transmitted via modem over 100 feet of RG-58 cable using the XMODEM protocol. The DGPS information and the information received via the modem were supplied to a 386 laptop computer through two serial ports. The computer was used to provide a real time navigation solution by integrating the information received from

the serial ports with Kalman filtering techniques. The 386 also logged raw data for post-processing.

The experiment started by ensuring that the towfish and the computer were able to communicate and that the DGPS information was available. The towfish was lowered to the water and allowed to gather enough GPS information to initialize its program. The GPS receiver was tracking six satellites at the commencement of the experiment. Over the next one and one half hours seven different test runs were obtained which amounted to over 30 minutes of raw data. The first four test runs amounted to primarily INS data due to minor control problems with the towfish. The last three test runs were representative of the mission profile of an AUV with dives to four to five feet in depth for approximately 30 seconds followed by 10 to 15 seconds on the surface while obtaining DGPS information. There was no way to determine the precise time required for the GPS receiver to acquire enough satellites for a fix. However, by watching the screen output on the computer, it was qualitatively estimated that the receiver was typically able to obtain a DGPS fix within five seconds of broaching the water's surface. This result is generally in agreement with that of the more quantitative evaluation of this effect reported in [Norton 94].

Typical results of at-sea testing are presented in Figure 22 through Figure 24. As discussed previously the towfish was allowed to obtain multiple DGPS fixes prior to the first dive in order to initialize the system. Once the initialization was complete the towfish was commanded to dive for approximately 30 seconds before surfacing for 10 to 15 seconds.

Figure 22 shows results of the sea test between the first two DGPS fixes. The towfish was surfaced at point 1. The towfish was then commanded to dive and allowed to navigate via the INS for approximately 30 seconds to point 2 where it was then commanded to surface. Point 3 depicts the towfish on the surface where it was allowed to update its position by obtaining a DGPS fix. Qualitatively, the data shows a good correlation to the actual track of the towing boat with the exception that sensed water speed appears to be overestimated. A quantitative analysis of the data shows that from the last DGPS fix at

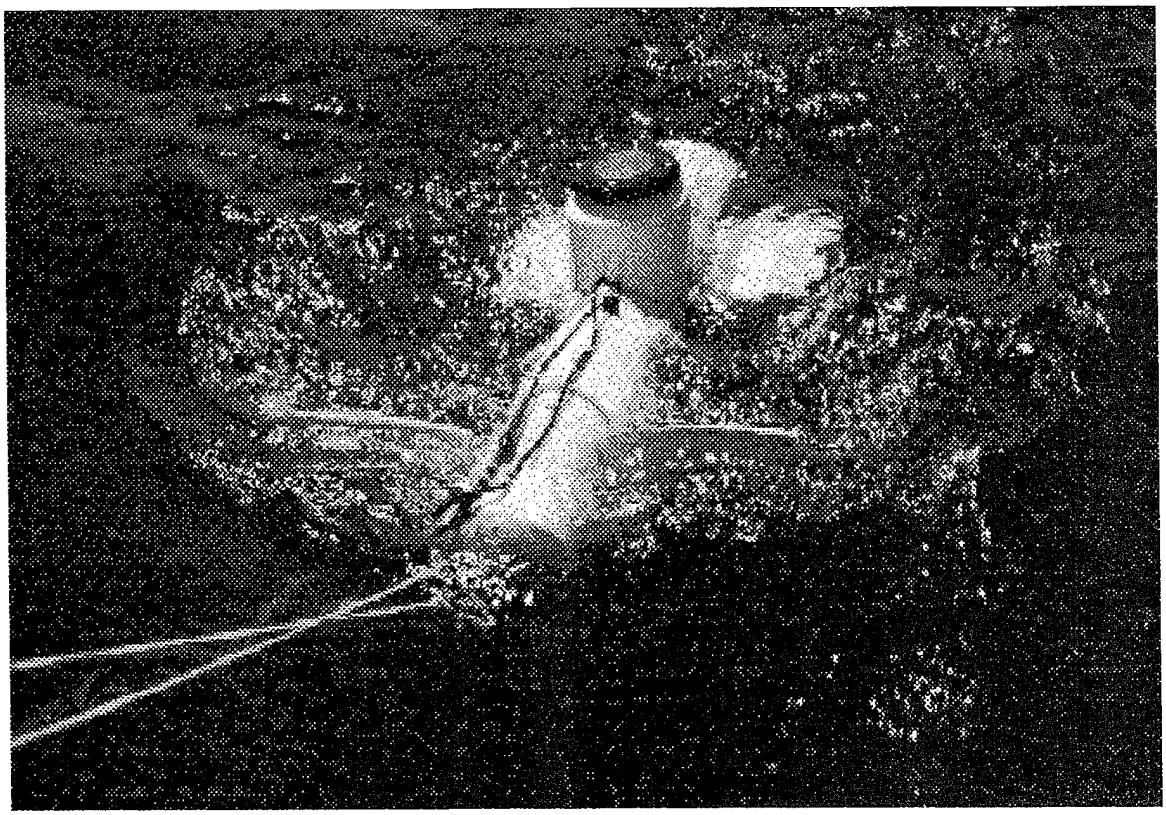

Figure 21: At-sea testing with the towfish

point 1 to the first DGPS fix at point 3 was approximately 70 m, while the distance that the INS determined was on the order of 280 m. The difference between the actual distance traveled and the estimated distance is off by a factor of 4.

Figure 23 shows results of the sea test between the second two DGPS fixes. Qualitatively the INS track continues to have a good correlation to the track of the towing vessel. A quantitative analysis shows that the difference between the two DGPS fixes (points 3 and 5) is again approximately 70 m, while the distance that the INS determined has decreased from 280 m to approximately 225 m. This decrease is an indication that the Kalman filter may be attempting to correct for the position error through estimation of ocean current. The difference between the actual distance traveled and the estimated distance has been reduced to a factor of 3.

Figure 24 presents the data for the entire test run. A similar analysis of the data can be performed between each of the DGPS fixes. Between DGPS fixes 5 and 7 the distance traveled is 44 m, while the INS determined that it traveled 130 m, which is approximately a factor of 3. Analysis of the data appears to show that the Kalman filter was able to estimate the heading fairly accurately. However, there appears to be a significant error in the waterspeed sensor. There is a potential for error in the sensor since it was not possible to calibrate the device prior to testing. Further analysis shows that the Kalman filter was attempting to correct for this error in speed as is evidenced by the decrease in each subsequent correction between GPS fixes. The filter then appears to overcompensate between points 7 and 8 and 9 and 10. This overcompensation is hypothesized to be due to the Kalman filter attempting to estimate the ocean current. Confirmation of this hypothesis is an important area for future work.

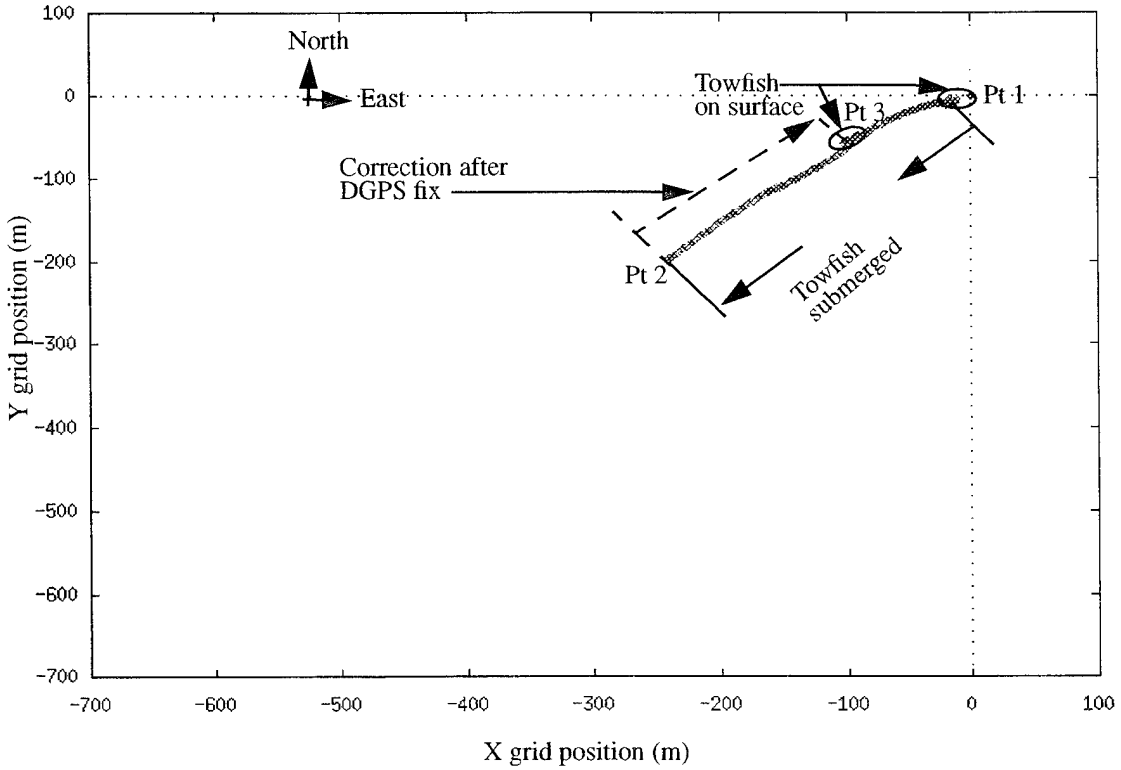

Figure 22: Sea test results of first two DGPS fixes

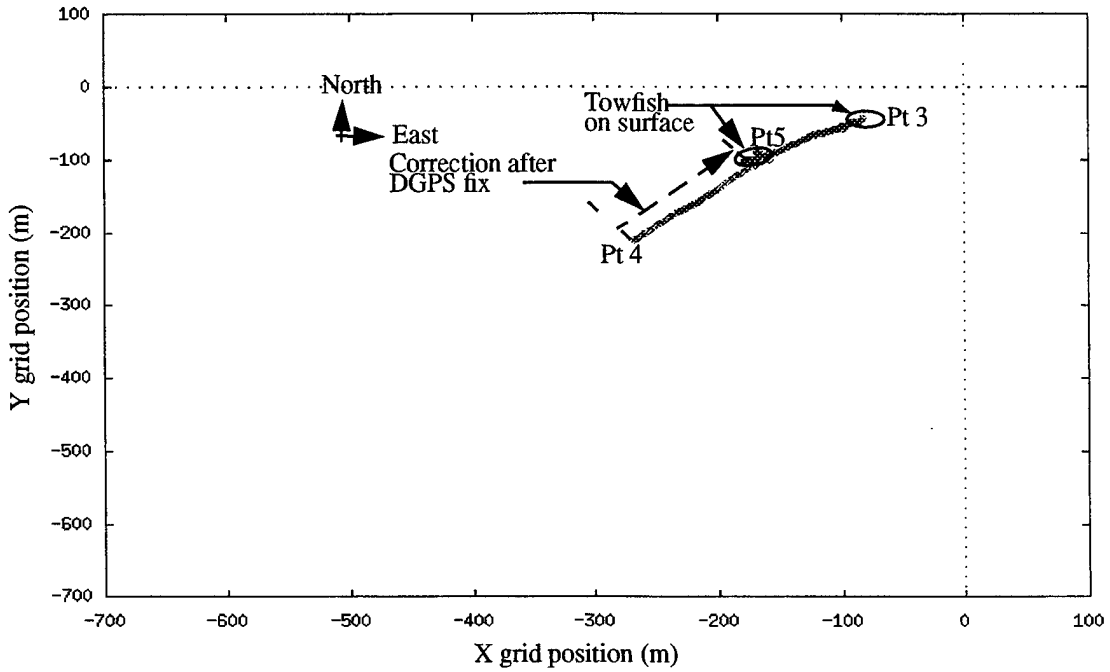

Figure 23: Sea test results of second two DGPS fixes

E. SUMMARY

This chapter provides an explanation of the experimental tests that were developed and performed to determine the feasibility of using an integrated GPS/INS system to navigate an AUV. It also explains and analyzes the results of the static tests, cart tests, and at-sea testing.

Static testing was performed to determine the suitability of the Systron-Donner IMU and differential GPS for testing purposes. Since the 10 Hz antialiasing filter discussed in [Schubert 95] was untried, it was important to test the output of the IMU after passing through this filter. The filter appears to work well and produces the desired results. Since DGPS was to be used as truth data, it was important to obtain a quantitative assessment of the accuracy of the system. The root sum square of the error show that the GPS system used is capable of producing navigation fixes well within the 100 m requirements of SPS. Also,

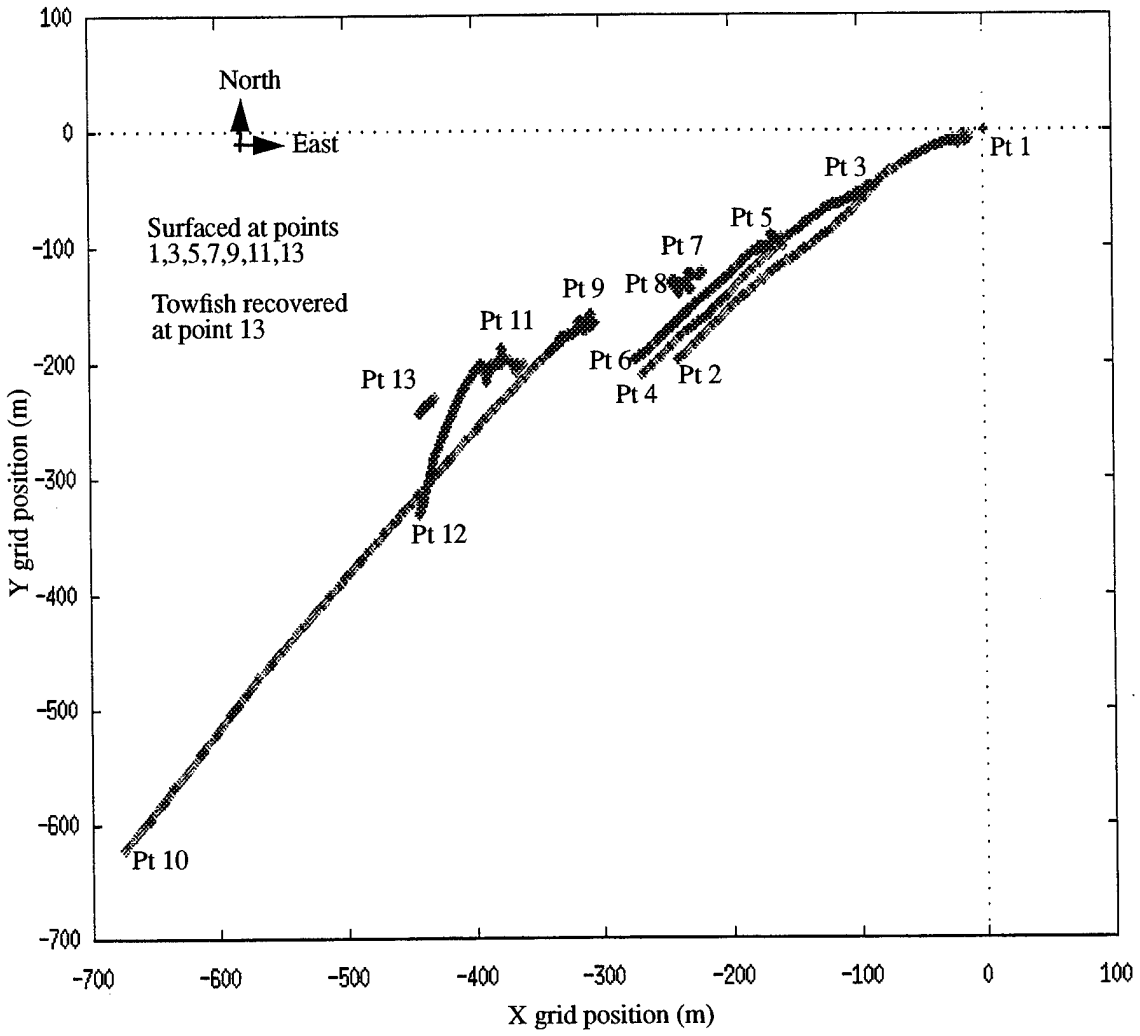

Figure 24: Typical results of at-sea tests

the DGPS system is capable of producing results in the one to two meter range which is well with the 10 m required by the SANS mission profile.

Cart testing was performed prior to at sea testing in order to establish the system was fully functional in its operational configuration. The tests were performed by placing the towfish on a wheeled cart and pushing it around a surveyed course. The results showed that the GPS/INS system was functioning properly. It also showed a good correlation to the surveyed track and provided a way to obtain initial values for the Kalman filter gains.

The mission of the Phoenix requires that the SANS and the antenna mounted on the AUV be as undetectable as possible. This requires that the antenna protrude as little as possible above the water's surface. This makes the antenna susceptible to wave action and possible loss of GPS signal. The at-sea testing showed that an antenna that is mounted on an AUV can expect to track sufficient satellites to provide GPS fixes within a relatively short amount of time after surfacing. The concept of using a relatively inexpensive IMU with limited accuracy coupled with a GPS was proven to be a viable solution to the challenge of shallow water AUV navigation. However, further at sea testing and gain tuning will perfect this solution and produce better estimates of system accuracy.

VII. SIMULATION

A. INTRODUCTION

The SANS software utilized for this research was prototyped in simulation prior to actual implementation. The goals of this simulation were to simulate navigation using an integrated INS and GPS, interface with high level software responsible for controlling an entire AUV mission, model the hardware devices of the AUV in software, and finally, to test and allow experimentation with Kalman filter designs. Attainment of these goals allowed a simulated AUV to find the shortest path around numerous obstacles while navigating through a series of way-points. Many of the algorithms and procedures utilized were directly translated and implemented into the actual SANS software design. This simulation helped to reduce the development time required for the SANS software as a whole. One of the primary reasons for this improvement was the 2D graphical representations of the motion of the AUV which the simulation produced.

Due to the rapid prototyping utility of CLOS (Common LISP Object System) [Koschmann 90], the CLOS language was chosen for implementation of the simulation. The object-oriented nature of CLOS allowed the AUV and SANS to be represented as objects and classes. Each of these was tested individually using the CLOS interpreter. The interpretive nature of the language also allowed for quick testing of each individual method in a class and inspection of the slot values of individual objects to determine their status.

This chapter will provide an overview of the simulation architecture and where it fits into the Rational Behavior Model (RBM) as described in [Byrnes 93]. It also highlights some of the important aspects of the implementation and some of the differences between the simulation and the actual SANS code. Finally a discussion of the results of the simulation and possibilities for further use is presented.

B. CODE ARCHITECTURE

The Rational Behavior Model software architecture (RBM) [Byrnes 93] provides a method of dealing with the diversity of the processes involved in controlling an autonomous vehicle. This diversity is due to the inherent differences between high-level processes based on artificial intelligence (AI) techniques, low-level processes involving engineering and robotics concerns, and coordination processes which must link the two. To deal with these varying requirements, the RBM divides the processes into three levels. These levels are termed the *Strategic level,* which handles mission control, the *Tactical* level, which determines what actions are necessary to satisfy the demands of the *Strategic* level, and the *Execution* level which carries out those actions in real time. Figure 25 illustrates where the code implemented for the simulation of this thesis fits into RBM. The objects contained in the dashed circle represent the objects which were implemented in prototyping the SANS. Those objects outside the dashed circle were used to drive the navigation software. The SANS objects lie entirely within the *Tactical* and *Execution* levels.

The Navigator, INS and replanner classes are part of the tactical level. These objects represent software in the actual SANS and form the basis for the design of the real-time software used in this research. Their basic role in the simulation is to respond to navigation related commands given by the OOD object by processing information acquired from hardware objects at the execution level.

Those objects which appear as part of the execution level in Figure 25 are considered hardware. These include the water-speed, IMU, GPS, UHF, Auto-Pilot and the Sub itself. The hardware object classes, with the exception of the AUV class, are derived from the black box class. The black box class itself is derived from a base class called rigid body. The sub class is descended directly from the rigid body class and it instantiates an object from each of the other hardware classes when instantiated itself. Figure 26 illustrates these inheritance relationships.

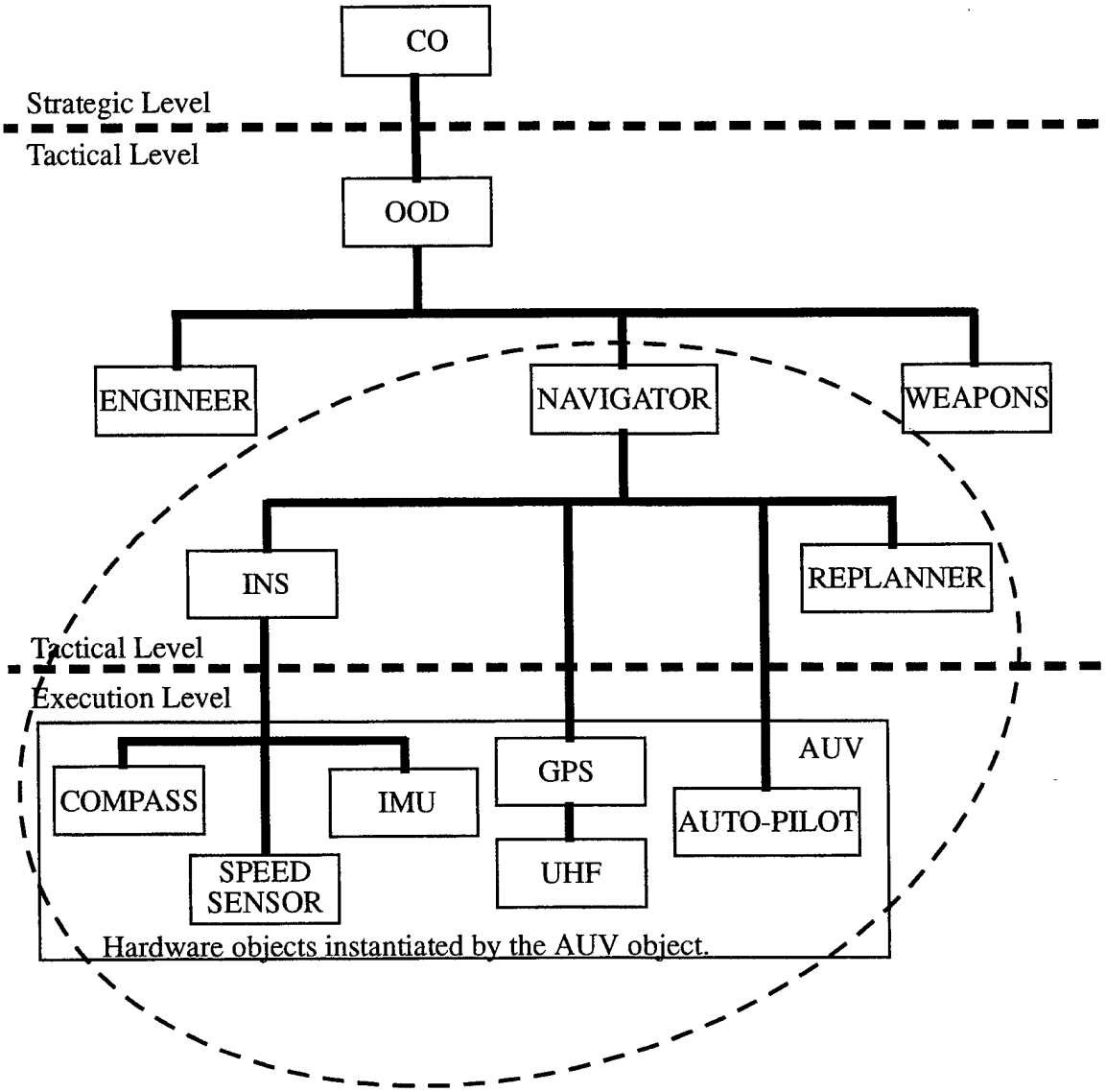

Figure 25: SANS objects in the RBM hierarchy

There are several objects which support the simulation that are related only in an indirect manner to the navigation problems. In order to approximate the mechanics of real-world motion, the SANS simulation includes an large amount of kinematics code. The methods of the euler-angle-rigid-body class control the simulated motions of those objects derived from it using numerous matrix manipulation functions. The strobe-camera class

supports the graphical depiction of the motions of the simulated AUV. Figure 27 shows the image of the AUV created by this class.

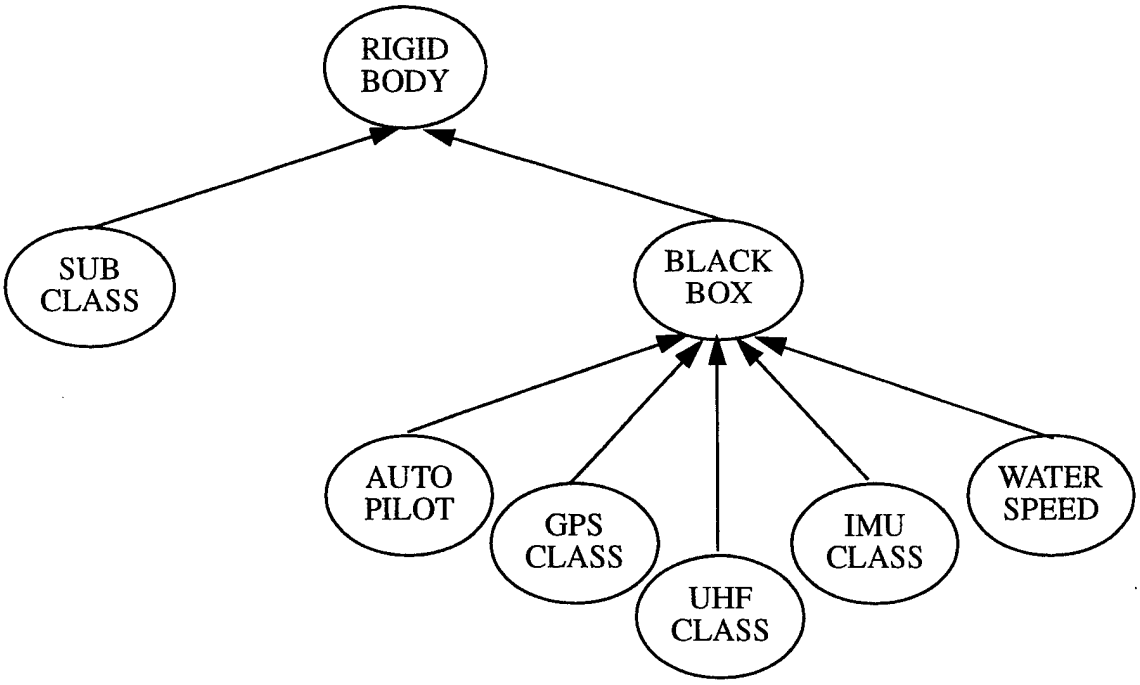

Figure 26: SANS simulation class hierarchy

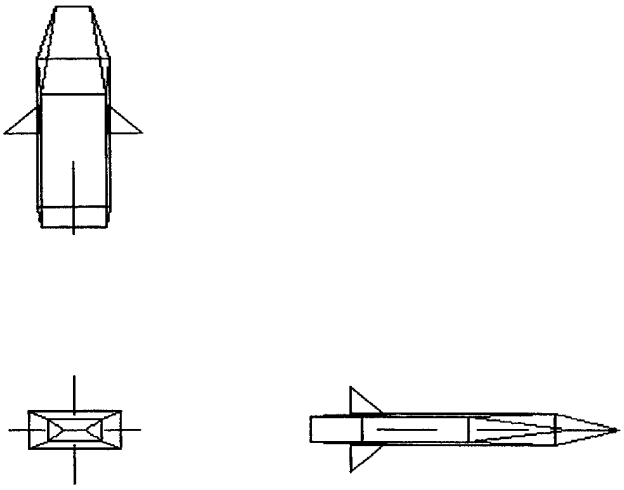

Figure 27: Wire frame depictions of AUV

C. IMPLEMENTATION

The simulated geometric world used in implementing the SANS simulation was kept as simple as possible. All positions are stated in coordinates relative to the origin. Courses and headings are expressed in radians measured clockwise from the "north" or positive x-axis. Obstacles are represented as ordered sets of vertices which specify *normal* or counter clockwise polygons as described in [Kanayama 95]. No attempt was made to model the complex hydrodynamic forces present in the real-world AUV environment. Figure 28 depicts the "world" in which the simulated AUV navigates. The following sections describe the functionality corresponding to each software class.

1. Navigator Class

The navigator acts as the primary controller for all navigation tasks. Once told to proceed to a specific fix, it will continue to direct the auto-pilot towards the fix position until the way-point representing it is captured or an abort command is received from the OOD object. Throughout any simulated mission, the navigator will provide information as requested by higher level objects for use in tactical and strategic decision making. The basic tasks of the Navigator object are to determine current position based on inputs from the INS and GPS object inputs, manage and update navigational way-points and intermediate sub-way-points, and determine the course to the next way point.

The state of the navigator at any given time is based on the contents of its five slots. These contain the target way-point to which it is currently proceeding, the sub-way points it must traverse while transiting to the target way-point, and an ordered list of way-points which must be captured in order to complete the mission. These slots are continually updated through out a mission.

Update of the status of the navigator is driven by orders and queries sent to the navigator by the OOD object. Without regard to content, whenever an order or query is received, the navigator commands the AUV to update its position by a set time slice and

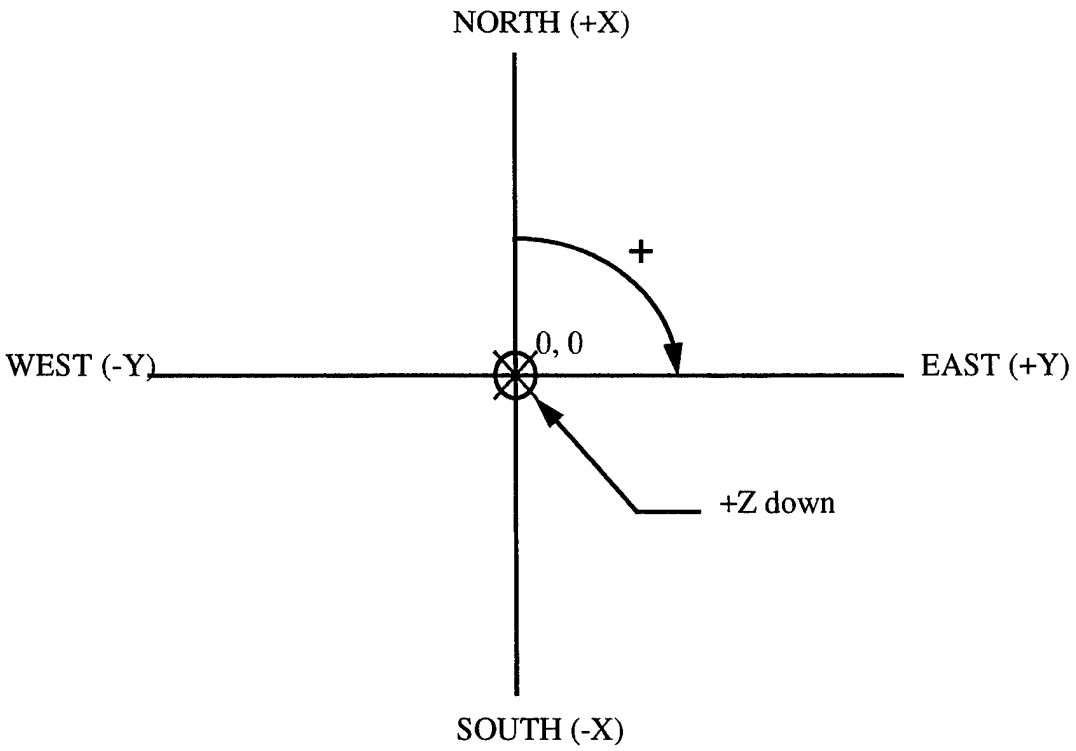

Figure 28: Geometric world

requests from the INS object its latest estimate of the current position. Only after these tasks have been completed does the navigator inspect the order or query and attempt to satisfy it.

Orders and queries are received as text strings via similar methods termed *order* and *query* respectively. Case statements are used in these two methods to determine what action is necessary. Orders, unlike queries, are designed to initiate action and require no response other than acknowledgment that the message was received. Queries on the other hand constitute a request for information. On receipt of either an order or a query, the navigator calls a sequence of methods as it performs the actions necessary to satisfy it. In this simulation, responses to some communications from the OOD object were left as stubs for future work.

The Navigator accomplishes way-point control through four methods. The simplest method determines if the current target way-point has been captured by examining the

distance between the estimated position of the AUV and the target-way-point. The other methods accomplish their purpose by destructive updates to the way-point list, sub-way-point list, and target-way-point slots. Update of the sub-way-point list is accomplished via the replanner object each time a new target-way-point is designated. This allows the AUV to follow the shortest path around simulated obstacles to the new target-way-point. Updates of the target-way-point and way-point list slots take place each time the Navigator is told to proceed to a new way-point. Overall updates are based upon the orders of the OOD which determines to what location (and hence what way-point) the mission will next proceed.

2. Replanner Class

The replanner solves the shortest path planning problem between any two way-points in a polygon environment. The algorithms used to arrive at a solution are based upon the techniques described in [Kanayama 95]. The replanner takes as input arguments a starting point, an end point and several sets of vertices. The sets of vertices describe convex polygons which represent simulated obstacles. Upon completion of execution the method returns a subset of the vertices which describe the shortest path from the start point to the end point. For purposes of this shortest path planning problem, the AUV is considered to be a dimensionless vehicle or a point robot.

The first step in solving the shortest path problem involves finding all "tangents" or possible path segments in the polygon world. These "tangents" can be divided into three types of line segments; tangents from the starting point to polygons, common tangents between polygons and tangents from polygons to the end points. Figure 29 shows the tangents in an example polygon world.

The second step in solving the shortest path problem requires that a determination be made as to the visibility of each tangent. Any tangent which enters one of the obstacle polygons is said to be invisible. There are several such invisible tangents in Figure 29. Visibility determination must be made for each individual tangent. The exact method used

to determine the visibility of a tangent will be based upon the type of tangent. It will, however, involve checking for intersection between the "tangent" and the line segments which make up the polygon obstacles. The visibility test in the polygonal world is an essential task in shortest path planning, but is one of the most time-consuming ones. Figure 30 shows the example polygon world with the invisible tangents removed.

Following the visibility test, a weighted graph ia available to which a modified Dykstra search is applied. This graph is composed of the visible "tangents," the line segments making up the boundaries of obstacle polygons, the vertices describing the obstacle polygons, and the original start and end points. Figure 30 is the graph on which search would be run. The shortest path generated by the search is specified using as an ordered list

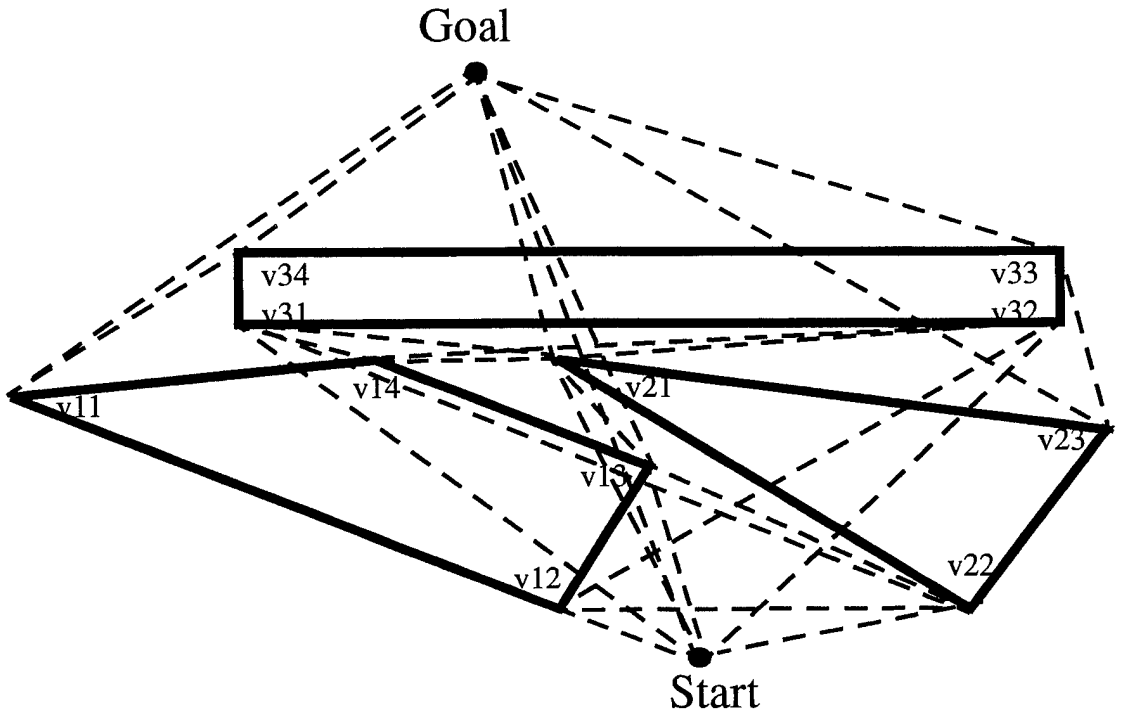

Figure 29: All tangents in an example polygon world

of polygon vertices. This list could be returned by the replanner to the navigator which could designate the vertices it contains as the sub-way-points. The sub-way-points would then be followed when travelling from one location (way-point) to another (Figure 31).

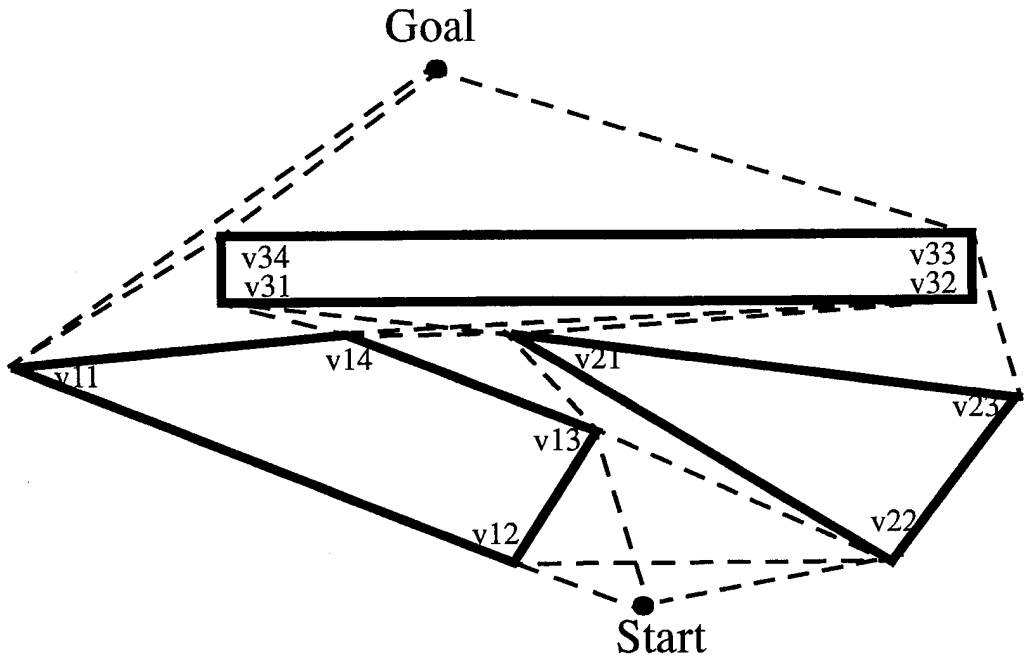

Figure 30: Visible tangents in an example polygon world

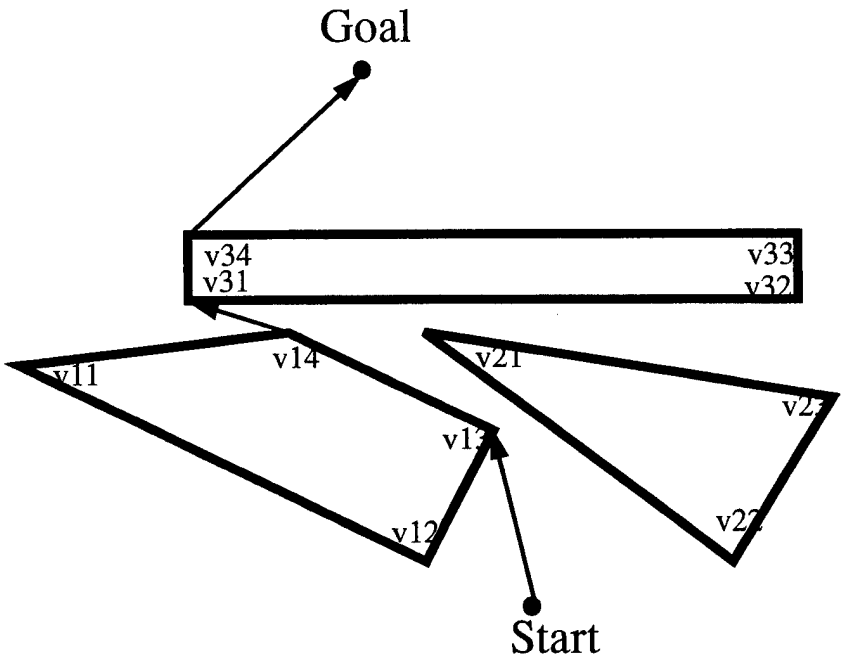

Figure 31: Shortest path determined from visibility graph

3. INS Process Class

The INS acts as a test bed for the Kalman filter design described in chapter V. It performs simulated dead reckoning in-between GPS fixes to maintain an updated estimate of the current position. The state of the INS at any given time is based upon the contents of the estimated posture and velocity slots it contains. These values are continually updated based on inputs provided by the IMU, water speed and compass objects. These inputs include linear accelerations and angular rates in body coordinates which are translated, filtered and finally numerically integrated to obtain an estimate of the position and attitude of the AUV.

4. Auto-pilot and Sub Classes

In order to simplify the motion simulation model, the auto-pilot is assumed to be "perfect". The "perfect" auto-pilot assumption indicates the there is a tight coupling between the Sub and the auto-pilot. This assumption eliminates the necessity of modeling hydrodynamic forces. The auto-pilot receives speed, heading and depth commands from the navigator. It is then responsible for insuring that these parameters are meet by the rigid body Sub. With the exception of longitudinal velocity, which lags the requested input by a first order time lag, the velocity vector of the submarine is exactly what the auto-pilot has commanded. This implies that the angle of attack and sideslip at any given time are zero. The following computations are involved in determining the pitch rate, roll rate, yaw rate and longitudinal velocity of the sub.

The rigid-body sub is assumed to be capable of instantaneously changing its pitch velocity vector. In other words there is no "mushing" as the vehicle establishes a new pitch attitude. This assumption allows use of the following first order equations for pitch rate(q):

$$\theta_{commanded} = K_d(z_{commanded} - z_{actual}) \tag{7.1}$$

$$q = K_q(\theta_{commanded} - \theta_{actual}) \tag{7.2}$$

All turns are assumed to be flat with no angle of bank. Any roll rate (p)that might occur is damped according to the following equation.

$$p = K_p \cdot \phi_{actual} \tag{7.3}$$

The lack of side slip allows the use of a smooth motion steering function designed for line tracking [Kanayama 95] to be used to determine yaw rate, r. This steering function limits the AUV to only tangential motions. Thus the translational speed v and the rotational speed w or equivalently, the translational speed v and the path curvature c are the only degrees of freedom fully simulated. The speed of the AUV is assumed to be relatively low.

For a vehicle which makes only tangential motions, the curvature changes continuously as a function of arc length, s. The curvature dc/ds at any given position along the vehicles track can therefore be expressed using the following steering function,

$$\frac{dc}{ds} = -\left(3K_\sigma(c_{actual} - c_{desired}) + 3K_\sigma^{\,2}(\Psi_{actual} - \Psi_{desired}) + K_\sigma^{\,3}\Delta d\right) \tag{7.4}$$

Δd is the signed distance between the vehicles actual position and the desired track. K_σ is related to the turning radius of the vehicle and determines the smoothness of the its motion. Using $\frac{dc}{ds}$ and the longitudinal velocity or translational speed, v, the yaw rate is determined as follows,

$$r = \frac{dc}{ds}\, v^2\, \Delta t = \frac{dc/dt}{ds/dt}\, v^2\, \Delta t = \frac{dc}{dt}\, v\, \Delta t \tag{7.5}$$

Parasitic and induced drags as well as other hydrodynamic forces are ignored. The first order time lag of longitudinal acceleration is computed as follows:

$$\dot{u} = K_u(u_{commanded} - u_{actual}) \tag{7.6}$$

5. IMU, Waterspeed and Compass Classes

The IMU, Waterspeed and Compass objects provide the INS with the information necessary to dead reckon between fixes. The sensor reading of the waterspeed object is obtained from the $\dot{x}$ position of the sub velocity slot. Similarly the heading which the compass object provides is obtained by accessing the ψ position in the posture slot of the sub.

The IMU generates simulated accelerometer and angular rate sensor readings based on the movement of the sub. These values are derived from the velocity rates $\dot{u}$, $\dot{v}$, and $\dot{w}$,and the angular rates $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$. The angular rates values are read in body coordinates directly from the velocity-growth-rate slot of the sub. The specific forces which would be sensed by a real world accelerometer are not directly available from the velocity growth rate slot. They are generated using the following (Newton-Euler) equations:

$$\ddot{x}_a = \dot{u} - (wq + g\sin\theta) \tag{7.7}$$

$$\ddot{y}_a = \dot{u} - (wp + ur) - g\cos\theta\sin\phi \tag{7.8}$$

$$\ddot{z}_a = \dot{w} - (uq + vp) - g\cos\theta\cos\phi \tag{7.9}$$

6. GPS and UHF Classes

The GPS and UHF classes simulate the process of obtaining differential-corrected GPS information. Each time the GPS object is queried by the navigator for a new fix, a random function is invoked which simulates the variability in satellite reacquisition time by the receiver. Once three satellites have been acquired the method accesses the posture

slot of the sub to obtain the current x, y, z position. This position is then vector added to simulated pseudorange corrections provided by the UHF object to create a differential corrected position. This corrected position is returned to navigator.

7. Clock Class

The clock controls the timing and speed of execution of the simulation. Prior to each iteration or update of the simulation the clock is incremented by a set time interval. Through out the next iteration all objects will determine the current simulation time by calling the *current-time* method of the clock. Since all objects will obtain an identical time during any given iteration regardless of execution order, the processes appear to operate simultaneously.

D. SIMULATION RESULTS

Figure 32 shows the wire frame AUV executing a simulated mission while using way-point navigation. The 'strobe' effect is due to periodic snap shots taken by the strobe camera class. In this mission the AUV transited to and captured four way-points. The vehicle was initialized at the origin on a heading of North with a velocity of zero. Once the initialization was completed, the OOD commanded the navigator to proceed to the first way-point. The navigator then calculated the course to the way-point and commanded the auto-pilot to accelerate and turn to the calculated course. During transit, the OOD continually requested the navigator to determine if the way-point had been captured. Upon being informed of capture, the OOD commanded the navigator to proceed to the next way-point. As above, the navigator made the necessary course calculations and commanded the auto-pilot accordingly. This process was repeated for the remaining two way-points. Following capture of the fourth way-point, the navigator proceeded to a recovery point where the mission was completed.

Throughout the simulated mission the INS calculated estimates of the AUV's position. These estimates were found to contain unacceptably large position errors. At the time of the writing of this thesis, the source of these errors was undetermined. However, the

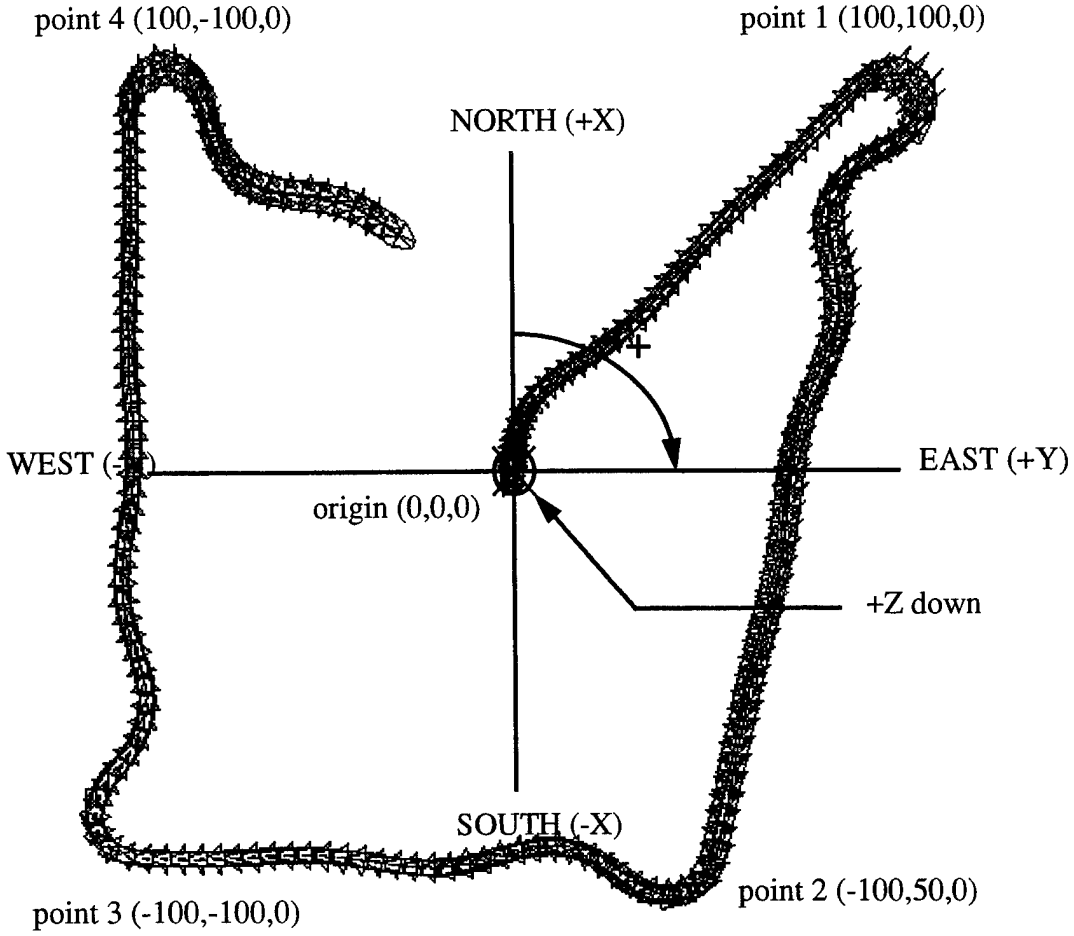

Figure 32: AUV way-point navigation

simulation enhanced the software engineering of the real-time SANS software and provides a basis for future analysis of both real-time and simulated navigation problems.

E. SUMMARY

The primary goal of this simulation was to simulate AUV navigation using an integrated INS and GPS system. The greatest benefit to this research was the ability of the simulation to act as a test bed for various filter and navigator designs. The simulation goes beyond the basic navigation problems of an AUV, however, and attempts to examine where the navigation problem fits into the RBM hierarchy and controlling software as a whole

This secondary purpose accounts for the extensive interface capabilities of the navigator class.

In order to take advantage of the object oriented paradigm, CLOS was chosen as the prototyping language. This made it possible to represent the many parts of the SANS as individual objects. The interpretive nature of CLOS sped testing and development.

The primary differences between actual SANS and the prototype SANS used in this simulation are due to simplifying assumptions regarding the environment in which the simulated AUV operates and the motions of the simulated AUV. Unlike the actual SANS, the simulation SANS operates in a nearly perfect environment. All errors in the current configuration are due to the numerical inaccuracies of the machine on which the simulation is run. Additional differences lie in the simplified communications between the simulated software and hardware objects. In the actual SANS implementation, these problems required a great deal of labor to overcome.

Future uses of the simulation might take on many forms. With the addition of simulated noise, improved Kalman filters might be tested and optimized. Simulated missions might be run to further test the RBM hierarchy. Auto-pilot designs could be tested by removing the "perfect" auto-pilot assumption and adding realistic hydrodynamic forces, either directly to the rigid-body sub class or through networked interaction with the NPS AUV Virtual World [Brutzman 94].

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The purpose of this thesis was to develop and test an integrated navigation unit using INS and GPS. The standards against which this unit can be evaluated are the requirements for a Small Autonomous Under Water Vehicle (AUV) Navigation System (SANS) described by[Kwak 93]. Evaluation of the concept involved two major phases. The first utilized computer simulation to determine where the unit might fit into a larger software hierarchy and test various concepts and methodologies. The second phase progressed into the actual implementation and testing of the GPS/INS integrated unit. This testing began with land-based cart tests and eventually advanced to open-water experiments where the unit was placed in a towed body behind a boat in both submerged and surfaced states. Examination of the GPS/INS unit in the light of the research questions which guided this experiment demonstrates the ability of the unit to meet SANS requirements.

The navigation system to be installed in the NPS Phoenix would be simpler than the one developed and tested in this research. This is due to the nature of the tow experiments performed in this research. Actual installation would eliminate the need for the hardware and software required to allow communication between those components of the system placed in the boat during the tow tests and those located in the towed body. In the final combined configuration, when all components are located within the Phoenix itself, there will be no need for the modems, tattle-tale or software implementing the XMODEM protocol. The computer simulation of the navigation system indicates that this simplified version will fit well with the hardware and software architectures of the Phoenix itself.

The simulation developed in conjunction with this research demonstrates the feasibility of an AUV navigating from point to point while conducting open ocean-transit, given the ability to accurately estimate the vehicle position. The software design establishes a system of way-points and then calculates the courses and headings necessary to transit from one to another. Water and land based tests showed that a integrated

navigation system combining GPS and INS is capable of providing a sufficiently accurate position to support this way-point navigation system.

The simulation prototypes a software interface capable of communicating with AUV controlling programs based on the Rational Behavior Model (RBM) [Byrnes 93]. The prototyped interface is capable of responding to a large number of commands which can be used to control a variety of mission types. Future plans for Phoenix indicate that the controlling software architecture will be based upon the RBM [Healey 95].

The open-water experiments conducted in conjunction with this research tested the capability of an integrated GPS/INS navigation system to obtain accurate differential GPS fixes. Throughout these tow tests the system was able to quickly reacquire at least three satellites and provide accurate fixes using differential corrections. Incorporation of the system into the Phoenix will provide it with this same capability.

Review of the simulation and examination of the experimental data indicate that the GPS/INS integrated unit developed in this research is capable of meeting the SANS requirements. All the requirements for a SANS were either meet or demonstrated to be obtainable with further research. Optimization of the INS filter gains and calibration of the various sensors making up the unit should allow it to meet the accuracy requirements in all regimes.

B. RECOMMENDATIONS FOR FUTURE WORK

There are many ways to build on the foundation this research has established. Much work remains to be done to perfect the Kalman filter design incorporated into the INS. This work can proceed along the same pattern established here by further modeling using and computer simulation followed by implementation into the real-time software and more rigorous tow tests. The following paragraphs will identify some specific areas and directions in which to move ahead.

Several simplifying assumptions were made during the construction of the computer simulation. These simplifications can be divided into two major areas. The first of these is

the absence of a realistic error model. The addition of computer generated noise to the simulated sensor readings will create an environment in which the Inertial Navigation System (INS) can be more fully tested and optimized. Removal of the "perfect" auto-pilot assumption from the simulation will be another improvement. The application of a realistic model of the various hydrodynamic forces which are present during real-world AUV operations will produce more true-to-life accelerometer and rate readings for processing by the navigation system. The virtual world described in [Brutzman 94] presents an opportunity for further experimentation in this area. Further study can also be applied to the replanner class to make it capable of not only providing the shortest path but also a safe path in the correct coordinate system.

Much calibration and bench testing of the SANS remains to be done. The water-speed sensor in particular is still largely untested. The 10-hertz filter is effective, but nevertheless runs at excessive operating temperatures. This indicates greater-than-necessary power consumption and the possibility of premature failure. The drift rate of the Motion Pack unit itself is unknown. Bench testing of this IMU will provide more insight into the bounds of its accuracy.

The final step in this research will be incorporation of the SANS into the NPS AUV. The first step in this evolution will include simplification of the hardware/software interface. The Navigator object will require extensive modifications to make it more closely resemble the navigator object implemented in the simulation. Additional capabilities will include the ability to manage and calculate headings towards geographical fixes and response capabilities for to a variety of orders and queries.

APPENDIX A: Real Time Navigation Source Code (C^{++})

A. TOWTYPES.H

```
#ifndef __TOWTYPES_H
#define __TOWTYPES_H

#include <stdio.h>
#include <dos.h>
#include <time.h>

#include "globals.h"        // Types used by serial communications software

#define GPSBLOCKSIZE   68 // Size of Motorola @@Ba position message
#define PACKETSIZE 133// Size of packet received via X-modem protocol

#define ONE_G 32.2185// One g in feet per second
#define GRAVITY -32.2185     // In feet per second

#define TicksToSecs(x) ((double) ((10 * x) / 182))

typedef char  ONEBYTE;
typedef short TWOBYTE;
typedef long  FOURBYTE;

typedef unsigned char  UNSIGNED_ONEBYTE;
typedef unsigned short UNSIGNED_TWOBYTE;
typedef unsigned long  UNSIGNED_FOURBYTE;

// Holds lat/long expressed in milseconds
struct latLongMilSec {
  long latitude;
  long longitude;
};
```

```
// Holds a latitude or longitude expressed in hours minutes and degrees
struct T_GEODETIC {
 TWOBYTE         degrees;
 UNSIGNED_TWOBYTE  minutes;
 double          seconds;
};

// Holds a latitude and longitude expressed as T_GEODETICs
struct latLongPosition {
 T_GEODETIC latitude;
 T_GEODETIC longitude;
};

// Holds a grid position
struct grid {
  double x,y,z;
};

// 3 X 3 matrix
struct matrix {
  float element[3][3];
};

// 3 X 1 matrix or vector
struct vector {
  float element[3];
};

// Oversize area to hold a GPS message
typedef BYTE GPSdata[2 * GPSBLOCKSIZE];

// Defines a type for storing INS packets
typedef BYTE PACKET[PACKETSIZE];
```

```
// Structure for passing around various types of INS information.
// The positions in the sample field of a stampedSample structure
        // sample[0]: x acceleration
        // sample[1]: y acceleration
        // sample[2]: z acceleration
        // sample[3]: phi
        // sample[4]: theta
        // sample[5]: psi
        // sample[6]: water speed
        // sample[7]: heading

struct stampedSample {
  grid est;              //position as estimated by the INS.
  double sample[8];      //sampler converted sample.
  PACKET samplePacket;   //raw packet of samples.
  struct time timeStamp; //time the sample arrived.
  float deltaT;
};

#endif
```

B. LOCATION.H

```
//Conversion constants for location of 36:35:42.2N and 121:52:28.7W
#define LatToFt 0.10134 //converts degrees Latitude to ft
#define LongToFt 0.08156 //converts degrees Longitude to ft
#define HemisphereConversion -1  //-1 if west of of Greenwich

//If variation is west then magvar should be negative
#define RADIANMAGVAR 0.261799  // Local area Magnetic variation in radians
```

C. TOWFISH.CPP

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>

#include "towtypes.h"
#include "nav.h"

int  breakHandler(void);
```

```
void printPosition (const latLongPosition&);

/*****************************************************************
  PROGRAM:Main
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:   Drives the navigator and its associated software. Counts
  the positions displays each to the screen. Exited only when control
  break is entered at the keyboard.
  RETURNS:   0
  CALLED BY:   none
  CALLS:      initializeNavigator (nav.h)
                              navPosit (nav.h)
                              printPosition
                              breakHandler
*****************************************************************/

int
main (int argc, char *argv[])
{
  ctrlbrk(breakHandler);  // trap all breaks to release com ports
  setcbrk(1);   // turn break checking on at all times

  char *dataFile, *scriptFile;

  switch (argc) {
        case 2:
              scriptFile = new char[strlen(argv[1])];
              strcpy(scriptFile, argv[1]);//explicit script file only
              dataFile = "data";//default raw data file
              break;
        case 3:
              scriptFile = new char[strlen(argv[1])];
              strcpy(scriptFile, argv[1]);//explicit script file only
              dataFile = new char[strlen(argv[2])];
              strcpy(dataFile, argv[2]);//explicit script file only
              break;
        default:
              scriptFile = "script";//default script file
              dataFile = "data";//default raw data file
  }

  cout << "\nRecording script in " << scriptFile;
```

```
 cout << "\nRecording data in " << dataFile << endl;

 //Instantiate the navigator
 navigator nav1(scriptFile, dataFile);

 latLongPosition currentLocation; // Lat/Long of most recent fix
 Boolean  fixReceived = FALSE;//True if a new fix was recieved
 int     fixCount=0;  // Count of navigation fixes recieved

 //Initialize the navigator
 currentLocation = nav1.initializeNavigator();

 cout << "\nInitialization Complete!\n";
 cout << "Initial Position:\n";

 //Print the initial position
 printPosition(currentLocation);

 while (TRUE)  {

        // Attempt to get a fix from the navigator
        fixReceived = nav1.navPosit(currentLocation);

        if (fixReceived) {
                // New fix recieved
                cout << "\nFix " << ++fixCount << endl;
                printPosition(currentLocation);
        }
  }
}

/*********************************************************************
  PROGRAM: printPositon
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Displays position to the screen
  RETURNS:  void
  CALLED BY:mail
  CALLS:      none
*********************************************************************/

void
printPosition (const latLongPosition& posit)
```

```
{
 cout << "Latitude: " << posit.latitude.degrees << ':'<<
        posit.latitude.minutes << ':' << posit.latitude.seconds << endl;
 cout << "Longitude: " << posit.longitude.degrees << ':'<<
        posit.longitude.minutes << ':' << posit.longitude.seconds << endl;
}

/*************************************************************************
  PROGRAM:breakHandler
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Cleans up com ports upon program exit.
  RETURNS: 0
  CALLED BY:  main
  CALLS:      cleanup (portBank.h)
*************************************************************************/

int
breakHandler(void)
{
  COMports.cleanup();
  exit(0);
  return 0;     // keep the compiler happy
}
```

D. NAV.H

```
#ifndef __NAVIGATOR_H
#define __NAVIGATOR_H

#include <fstream.h>
#include <iostream.h>
#include <math.h>

#include "towtypes.h"
#include "gps.h"
#include "ins.h"
#include "location.h"

// Converts milseconds to degrees
#define MSECS_TO_DEGREES (1.0/(1000.0 * 3600.0))
```

```
/*************************************************************************
  CLASS:navigator
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Combines GPS and INS information to return the current
  estimated position.
*************************************************************************/

class navigator {

  public:

        //Constructor, opens script and data files
        navigator(char* scriptFile = "navScript", char* dataFile = "navData")
        : positionData (scriptFile), rawData(dataFile),elapsedTime(0.0){}

        //Destructor, closes script and data files
        ~navigator() {positionData.close(); rawData.close();}

        //provides the navigator's best estimate of current position
        Boolean navPosit (latLongPosition&);

        //Initialize the navigator
        latLongPosition initializeNavigator();

  private:

        INS ins1;//INS object instance.
        GPS gps1;//GPS object instance.

        ofstream positionData;   // Position script file
        ofstream rawData;// Post processing output file.

        latLongMilSec origin; //lat-long of navigational origin

        //Write position information to script file
        void writeScriptPosit(int, latLongMilSec&, char);

        //Write an INS packet and its timeStamp to the outPut file
        void writeInsData(const stampedSample& drPosition);

        //Write a GPS message to the outPut file.
        void writeGpsData(const GPSdata& satPosition);
```

```
        //Returns the position in Miliseconds
        latLongMilSec getMilSec(const GPSdata&);

        //Convert position in milSec to degress, minutes, seconds and milsec
        latLongPosition milSecToLatLong(const latLongMilSec&);

        //Convert xy (grid) position to lat long
        latLongMilSec gridToMilSec(const grid&);

        //Converts lat/long to xy position
        grid milSecToGrid(const latLongMilSec&);

        //Parses and returns the time of a GPS message.
        double getGpsTime(const GPSdata& rawMessage);

        float elapsedTime;

};

#endif
```

E. NAV.CPP

```
#include "nav.h"
#include "signal.h"
#define SIGFPE 8// Floating point exception

/*************************************************************************
  PROGRAM:navPosit
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:      11 July 1995
  FUNCTION:Provides the navigator's best estimate of current position.
  Attempts to obtain GPS and INS position fixes from the gps
  and ins objects and copies the most accurate fix available
  into the input argument 'navPosition'. Writes the raw position
  fix data to the output file for post processing. Sets a return
  flag to indicate whether a valid fix was obtained.
  RETURNS:TRUE, a valid position fix is in the variable 'navPosition'.
             FALSE, otherwise.
  CALLED BY:towfish.cpp (main)
  CALLS:     gpsPosition (gps.h)
                              correctPosition (ins.h)
                              insPosition (ins.h)
                              getMilSec (nav.h)
                              gridToMilSec (nav.h)
                              milSecToGrid (nav.h)
                              milSecToLatLong (nav.h)
                              writeScriptPosit (nav.h)
*************************************************************************/
void fpeNavPosit(int sig)
{if (sig == SIGFPE) cerr << "floating point error in navPosit\n";}

Boolean
navigator::navPosit (latLongPosition& navPosition)
{
  signal (SIGFPE, fpeNavPosit);
  GPSdata satPosition;     // the latest GPS position
  stampedSample drPosition; // the latest INS position
  latLongMilSec gpsMilSec;  // the latest GPS position in milseconds
  latLongMilSec insMilSec;  // the latest INS position in milseconds
  static int fixCount(0);

  //Get the INS and GPS positions
  Boolean insFlag = ins1.insPosition(drPosition);
  Boolean gpsFlag = gps1.gpsPosition(satPosition);
```

```
//INS and GPS positions obtained?
if (insFlag && gpsFlag) {
        cout << "\nINS & GPS: ";
        // Write INS packet to out put file.
        elapsedTime += drPosition.deltaT;
        writeInsData(drPosition);
        //Write GPS message to output file.
        writeGpsData(satPosition);
        //Parse position from GPS messsage
        gpsMilSec = getMilSec(satPosition);
        //Write milsec position to script file
        writeScriptPosit(++fixCount, gpsMilSec, 'G');
        //Pass GPS position to INS object for navigation corrections.
        ins1.correctPosition(milSecToGrid(gpsMilSec), getGpsTime(satPosition));
        //Covert position in mil sec to latitude and longitude.
        navPosition = milSecToLatLong(gpsMilSec);
        return TRUE;
}
else {
        //Only INS position obtained?
        if (insFlag) {

                cout << "\nINS: ";
                // Write INS Packet to output file.
                elapsedTime += drPosition.deltaT;
                writeInsData(drPosition);
                insMilSec = gridToMilSec(drPosition.est);
                //Write milsec position to script file
                writeScriptPosit(++fixCount, insMilSec, 'I');
                navPosition = milSecToLatLong(insMilSec);
                return TRUE;
        }
        else {
                // Only GPS position obtained?
                if (gpsFlag) {
                        cout << "\nGPS: ";
                        // Write GPS message to output file.
                        writeGpsData(satPosition);
                        //Parse position from GPS messsage
                        gpsMilSec = getMilSec(satPosition);
                        //Write milsec position to script file
                        writeScriptPosit(++fixCount, gpsMilSec, 'G');
                        //Pass GPS position to INS object for navigation corrections.
```

```
                    ins1.correctPosition(milSecToGrid(gpsMilSec)
                                        ,getGpsTime(satPosition));
                    //Convert position in mil sec to lat/long.
                    navPosition = milSecToLatLong(getMilSec(satPosition));
                    return TRUE;
                }
                else {
                    return FALSE; // No new position available
                }
            }
        }
}
/*************************************************************************
  PROGRAM:   writeScriptPosit
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Writes the fix number, the position in milSec and the type
                        of fix to the script file.
  RETURNS:   void
  CALLED BY:navPosit (nav.cpp)
                        initialPosit  (nav.cpp)
  CALLS:     None
*************************************************************************/
void
navigator::writeScriptPosit(int fixNumber, latLongMilSec& posit, char fixType)
{
  positionData << fixNumber << ' '
                        << posit.latitude << ' '
                        << posit.longitude << ' '
                        << fixType << ' '
                        << elapsedTime << endl;
}

/*************************************************************************
  PROGRAM:   writeInsData
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Writes the packet and the time stamp contained in a stamped
                        sample to the out put file for post processing.
  RETURNS:   void
  CALLED BY:navPosit (nav.cpp)
  CALLS:     None
*************************************************************************/
```

```
void
navigator::writeInsData(const stampedSample& drPosition)
{
  for( int j= 0; j < PACKETSIZE; j++) {
        rawData << drPosition.samplePacket[j];
  }
  rawData << drPosition.timeStamp.ti_min
                << drPosition.timeStamp.ti_hour
                << drPosition.timeStamp.ti_hund
                << drPosition.timeStamp.ti_sec << endl;
}

/*************************************************************************
  PROGRAM:   writeGpsData
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:   Writes a raw GPS message to the out put file for post
                              processing.
  RETURNS:    void
  CALLED BY:navPosit (nav.cpp)
  CALLS:      None
*************************************************************************/

void
navigator::writeGpsData(const GPSdata& satPosition)
{
  for( int j = 0; j < GPSBLOCKSIZE; j++) {
        rawData << satPosition[j];
  }
}
```

```
/*************************************************************************
  PROGRAM:initializeNavigator
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:      11 July 1995
  FUNCTION:   Obtains an initial GPS fix for use as a navigational origin for
  grid positions used by the INS object. Saves the origin and passes
  it to the INS object in latLong form.
  RETURNS:TRUE
  CALLED BY:  towfish (main)
  CALLS:      gpsPosition (gps.cpp)
                              correctPosition (ins.cpp)
                              getMilSec (nav.cpp)
                              milSecToGrid (nav.cpp)
*************************************************************************/

latLongPosition
navigator::initializeNavigator()
{

  GPSdata satPosition;  //gps position message

  // Loop until an initial GPS fix is obtained.
  while(!gps1.gpsPosition(satPosition))
        { /* */ }
  //Save navigational origin for later grid position conversions.
  origin = getMilSec(satPosition);
  //Write the initial position to the script file
  writeScriptPosit(0, origin, 'G');
  //Pass GPS position to INS object for navigation corrections.
  ins1.insSetUp(getGpsTime(satPosition));
  //Return the initial position to the caller.
  return milSecToLatLong(origin);
}
```

```
/*************************************************************************
  PROGRAM:getMilSec
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Extracts a position in mili seconds from a Motorola (@@Ba)
  position contained in the input argument 'rawMessage' and returns it.
  RETURNS:The latitude and longitude in milseconds.
  CALLED BY:   navPosit (nav.cpp)
                              initializeNavigator (nav.cpp)
  CALLS:none.
*************************************************************************/

latLongMilSec
navigator::getMilSec(const GPSdata& rawMessage) {

  FOURBYTE temps4byte;
  latLongMilSec position;

  temps4byte      = rawMessage[15];
  temps4byte      = (temps4byte<<8) + rawMessage[16];
  temps4byte      = (temps4byte<<8) + rawMessage[17];
  temps4byte      = (temps4byte<<8) + rawMessage[18];

  position.latitude = temps4byte;

  temps4byte      = rawMessage[19];
  temps4byte      = (temps4byte<<8) + rawMessage[20];
  temps4byte      = (temps4byte<<8) + rawMessage[21];
  temps4byte      = (temps4byte<<8) + rawMessage[22];

  position.longitude = temps4byte;

  return position;
 }
```

```
/*************************************************************************
  PROGRAM:milSecToLatLong
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Converts a position expressed in totally in mili seconds to
  degress, minutes, seconds and mili seconds and returns the result.
  RETURNS:The position in degress, minutes, seconds and mili seconds.
  CALLED BY:   navPosit (nav.cpp)
  CALLS:none
*************************************************************************/

latLongPosition
navigator::milSecToLatLong(const latLongMilSec& milSec) {

  latLongPosition  position;

  double degrees, minutes;

  degrees = (double)milSec.latitude * MSECS_TO_DEGREES;
  position.latitude.degrees = (TWOBYTE)degrees;

  if(degrees < 0)
        degrees = fabs(degrees);

  minutes = (degrees - (TWOBYTE)degrees) * 60.0;
  position.latitude.minutes = (TWOBYTE)minutes;
  position.latitude.seconds = (minutes - (TWOBYTE)minutes) * 60.0;

  degrees = (double)milSec.longitude * MSECS_TO_DEGREES;
  position.longitude.degrees = (TWOBYTE)degrees;

  if(degrees < 0)
        degrees = fabs(degrees);

  minutes = (degrees - (TWOBYTE)degrees) * 60.0;
  position.longitude.minutes = (TWOBYTE)minutes;
  position.longitude.seconds = (minutes - (TWOBYTE)minutes) * 60.0;

  return position;

 }
```

```
/*************************************************************************
  PROGRAM:gridToMilSec
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Convert a grid position to a latitude and longitude in mil-
  seconds and returns the result.
  RETURNS:The latitude and longitude in milseconds.
  CALLED BY:navPosit (nav.cpp)
  CALLS:none
*************************************************************************/
void fpeGridToMilSec(int sig)
{if (sig == SIGFPE) cerr << "floating point error in gridToMilSec\n";}

latLongMilSec
navigator::gridToMilSec(const grid& posit)
{
  signal(SIGFPE, fpeGridToMilSec);
  latLongMilSec  latLong;
  cout << "\nposit.x = " << posit.x << "\nposit.y = " << posit.y << endl;

  //converts grid in ft to latitude
  latLong.latitude = origin.latitude + (posit.x / LatToFt);

  //converts grid in ft to longitude
  latLong.longitude = origin.longitude +
                HemisphereConversion * (posit.y / LongToFt);

  return latLong;
}

/*************************************************************************
  PROGRAM:milSecToGrid
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Convert a latitude and longitude expressed in milseconds to
  a grid position based on the lat/long of the grid origin.
  RETURNS:The grid position
  CALLED BY:navPosit (nav.cpp)
                              initializeNavigator (nav.cpp)
  CALLS:     none
  COMMENTS:   altitude is always assumed to be zero.
*************************************************************************/
```

```
//Converts latitude/longitude to xy coords in ft from origin
grid
navigator::milSecToGrid(const latLongMilSec& posit)
{
  grid  position;

  position.x = (posit.latitude - origin.latitude) * LatToFt;
  position.y = HemisphereConversion *
                                (posit.longitude - origin.longitude) * LongToFt;
  position.z = 0;

  return position;
}

/*************************************************************************
  PROGRAM:getGpsTime
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Parse the time of a gps message.
  RETURNS:The time of the gps message in seconds
  CALLED BY:navPosit (nav.cpp)
                                initializeNavigator (nav.cpp)
  CALLS:      none
*************************************************************************/
double
navigator::getGpsTime(const GPSdata& rawMessage)
{
  UNSIGNED_ONEBYTE    tempchar, hours, minutes;
  UNSIGNED_FOURBYTE   tempu4byte;
  double seconds;

  hours   = rawMessage[8];
  minutes = rawMessage[9];

  tempchar       = rawMessage[10];
  tempu4byte     = rawMessage[11];
  tempu4byte     = (tempu4byte<<8) + rawMessage[12];
  tempu4byte     = (tempu4byte<<8) + rawMessage[13];
  tempu4byte     = (tempu4byte<<8) + rawMessage[14];
  seconds = (double)tempchar + (((double)tempu4byte)/1.0E+9);

  return hours * 3600.0 + minutes * 60.0 + seconds;
}
```

F. GPS.H

```
#ifndef _GPS_H
#define _GPS_H

#include "portbank.h"
#include "towtypes.h"
#include "gpsbuff.h"

/*************************************************************************
  CLASS:gps
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Reads GPS messages from the GPS buffer. Checks for valid
                          checksum and minimun number of satelites in view.
*************************************************************************/

class GPS {

 public:

        //Class Constructor
        GPS() : port2(COMports.Init(COM2, BYTE(3), b9600,
                          NOPARITY, BYTE(8), BYTE(1), NONE, messages)) { }

        //returns the latest gps position and a flag
        Boolean gpsPosition (GPSdata&);

 private:

        //buffer for gps data
        GPSbuffer messages;

        //instantiates serial port communications on comm2
        bufferedSerialPort& port2;

        //calculates the check sum of the message
        Boolean checkSumCheck(const GPSdata);

};

#endif
```

G. GPS.CPP

```
#include <math.h>
#include "gps.h"

/*************************************************************************
  NAME:gpsPosition
  AUTHOR:  Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
  Determines if an updated gps position message is available and
  copies it into the input argument 'rawMessage'. If the message
  has a valid checksum and was obtained with atleast three
  satelites in view,  a 'TRUE' is returned to the caller,
  indicating that the message is valid.
  RETURNS:    TRUE, if a valid position message is contained in the
                                input argument.
  CALLED BY:  navPosit (navigator.h)
  CALLS:Get (buffer.h)
                          checkSumCheck (gps.h)
*************************************************************************/

Boolean
GPS::gpsPosition (GPSdata& rawMessage)
{
  if (messages.Get(rawMessage)) {

        // Check for a valid check sum and more the 3 satelites
        return Boolean((checkSumCheck(rawMessage)) && (rawMessage[39] >= 3));
  }
  else {
        return FALSE;  // No updated position is available.
  }

}
```

```
/*************************************************************************
  PROGRAM:checkSumCheck
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
  Takes an exclusive or of bytes 2 through 64 in a Motorola format (@@BA)
  position message and compares it to the checksum of the message of the
  message.
  RETURNS: TRUE, if the message contains a valid checksum
  CALLED BY:   gpsPosition (gps)
  CALLS:       none
*************************************************************************/

Boolean
GPS::checkSumCheck(const GPSdata newMessage)
{
  ONEBYTE chkSum(0);

  for (int i = 2; i < 65; i++) {
        chkSum ^= newMessage[i];
  }

  return Boolean(chkSum == newMessage[65]);
}
```

H. INS.H

```
#ifndef _INS_H
#define _INS_H

#include <time.h>
#include <math.h>
#include <dos.h>
#include <stdio.h>

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

#include "towtypes.h"
#include "sampler.h"
#include "location.h"
```

```
/*************************************************************************
  CLASS:ins
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Takes in linear accelerations, angular rates, speed and
  heading information and uses kalman filtering techniques to return
  a dead reconing position.
*************************************************************************/
class INS {

 public:

        //Constructor initializes gains
        INS();

        ~INS() {attitudeData.close();}

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

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

        //records the initial position of and time
        void insSetUp(double);

 private:

        ofstream attitudeData;// Post processing output file.

        double posture[6];    // ins estimated posture (x y z phi theta psi)
        double velocities[6]; // ins estimated linear and angular velocities
                                    // x-dot y-dot z-dot phi-dot theta-dot psi-dot
        double current[3];    // ins estimated error current (x-dot y-dot z-dot)

        struct time lastTime;     //time of last ins position fix
        double lastGPStime;    //time of last gps position fix

        sampler sam1;           //sampler instance

        matrix rotationMatrix;    //body to euler transformation matrix

        double biasCorrection[8]; //Software bias corrections for IMU rate sensors
```

```
        // Kalman filter gains.
        float Kone1, Kone2, Ktwo, Kthree1, Kthree2, Kfour1, Kfour2;

        // Finds the difference between two times of struct time type
        double findDeltaT (struct time& next, struct time& last);

        // Transforms from body coordinates to earth coordinates
        // and removes the gravity component
        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 update the posture
        void updatePosture (stampedSample&);

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

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

        // Convert magnetic direction based magnetic variation.
        double trueHeading(const double);

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

        //Applies bias corrections to a sample
        void applyBiasCorrections(double sample[]);

};

// Post multiply a matrix times a vector and return result.
vector operator* (matrix&, double[]);

#endif
```

I. INS.CPP

```
#include <iostream.h>
#include "ins.h"

/*************************************************************************
  PROGRAM: ins (constructor)
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Constructor initializes kalman filter gains and linear and
  angular velocities.
  RETURNS: nothing
  CALLED BY:  navigator class
  CALLS:      none
*************************************************************************/

INS::INS() : Kone1(0.1), Kone2(0.1),
                        Ktwo(0.6),
                        Kthree1(0.5), Kthree2(0.5),
                        Kfour1(1.0), Kfour2(1.0),
                        attitudeData("attitude")
{
  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

  // Initialize error bias to zero
  current[0] = 0.0;
  current[1] = 0.0;
  current[2] = 0.0;
}
```

```
/*************************************************************************
  PROGRAM: findDeltaT
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Converts two times stored in the time structure type of
  dos.h into the time in seconds and returns the difference.
  RETURNS: difference in seconds between the two input times.
  CALLED BY:   insPosit (ins.cpp)
  CALLS:       none
*************************************************************************/

double
INS::findDeltaT (struct time& next, struct time& last)
{
  double present, past;

  present = next.ti_hour * 3600.0 + next.ti_min * 60.0
                          + next.ti_sec + next.ti_hund / 100.0;
  past = last.ti_hour * 3600.0 + last.ti_min * 60.0
                  + last.ti_sec + last.ti_hund / 100.0;

  // Did 2400 occur imbetween present and past?
  if (present < past) {
        present += 86400.0;
  }
    .
  return present - past;
}
```

```
/*************************************************************************
  PROGRAM: insPosit
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  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 complimentary
  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)
*************************************************************************/

Boolean
INS::insPosition(stampedSample& newSample)
{
  double thetaA, phiA, xIncline, yIncline;
  double deltaT;  // Integration interval
  double waterSpeedCorrection[3]; // Filter correction for drift
                                  // and water speed

  static float elapsedTime = 0.0; // Maintains elapsed time

  if (sam1.getSample(newSample)) {

        newSample.sample[7] = trueHeading(newSample.sample[7]);

        applyBiasCorrections(newSample.sample);
/*
        cout << "\nx accel: " << newSample.sample[0]
             << " y accel: " << newSample.sample[1]
             << " z accel: " << newSample.sample[2] << endl;
```

```
        cout << "\nphi dot: " << newSample.sample[3]
             << " theta dot: " << newSample.sample[4]
             << " psi dot: " << newSample.sample[5] << endl;

        cout << "water speed: " << newSample.sample[6]
             << " heading: " << newSample.sample[7] << endl;
*/
        deltaT = findDeltaT(newSample.timeStamp, lastTime);

        if (deltaT == 0) {
            cout << "\nZero divide error in insPosition\n";
            printf("newSample.timeStamp: %2d:%02d:%02d.%02d\n",
             newSample.timeStamp.ti_hour, newSample.timeStamp.ti_min,
             newSample.timeStamp.ti_sec, newSample.timeStamp.ti_hund);
            printf("lastTime: %2d:%02d:%02d.%02d\n",
             lastTime.ti_hour, lastTime.ti_min,
             lastTime.ti_sec, lastTime.ti_hund);
            deltaT = 0.2;
        }

        newSample.deltaT = deltaT;

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

        if (fabs(yIncline) > 1.0) {
            cerr << "\n Inclination error! \n";
            return FALSE;
        }
        //Calculate low freq pitch and roll
        thetaA = asin(xIncline);
        phiA = -asin(yIncline);

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

        //Calculate estimated pitch rate (phi-dot).
        velocities[3] = newSample.sample[3] + Kone1 * (phiA - posture[3]);
        //Calculate estimated roll 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 pitch rate to obtain pitch angle
posture[3] += deltaT * velocities[3];
//integrate estimated roll rate to obtain roll angle
posture[4] += deltaT * velocities[4];
//integrate estimated yaw rate to obtain heading
posture[5] += deltaT * velocities[5];

elapsedTime += deltaT;

attitudeData << elapsedTime << ' ' << posture[3] << ' ' << posture[4]
                        << ' ' << posture[5] << endl;

buildRotationMatrix();

//Transform accels to earth coordinates
transformAccels(newSample.sample);

//Transform water speed to earth coordinates
transformWaterSpeed(newSample.sample[6], waterSpeedCorrection);

//Calculate speed over the ground
waterSpeedCorrection[0] += current[0];
waterSpeedCorrection[1] += current[1];

// Subtract out previous velocity and apply statistical gain
waterSpeedCorrection[0] =
      Kthree1 * (waterSpeedCorrection[0] - velocities[0]);
waterSpeedCorrection[1] =
      Kthree2 * (waterSpeedCorrection[1] - velocities[1]);

// Determine filtered accelerations
newSample.sample[0] += waterSpeedCorrection[0];
newSample.sample[1] += waterSpeedCorrection[1];

//Integrate accelerations to obtain velocities
velocities[0] += newSample.sample[0] * deltaT;
velocities[1] += newSample.sample[1] * deltaT;
velocities[2] += newSample.sample[2] * deltaT;

//Integrate velocities to obtain posture
posture[0] += velocities[0] * deltaT;
posture[1] += velocities[1] * deltaT;
```

```
        posture[2] += velocities[2] * deltaT;

        lastTime = newSample.timeStamp;

        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];

        newSample.est.x = posture[0];
        newSample.est.y = posture[1];
        newSample.est.z = posture[2];

        return TRUE;
  }
  else {
        return FALSE;// New IMU information was unavailable.
  }
}

/*************************************************************************
  PROGRAM:correctPosition
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION: Reinitializes the INS based on a known position and compute
  apparent current based on past accumulated errors of the INS. It is
  called by the navigator each time a new GPS (true) fix is obtained.
  RETURNS:void
  CALLED BY: navPosit (nav)
  CALLS:     none
*************************************************************************/

void
INS::correctPosition(const grid& truePosit, double positTime)
{
  double deltaT;

  // Correct for new day if necessary
  if (positTime < lastGPStime) {
        positTime += 86400;
  }
```

```
  // Find time since last gps fix.
  deltaT = positTime - lastGPStime;

  // Detemine INS error since last gps fix
  double deltaX = truePosit.x - posture[0];
  double deltaY = truePosit.y - posture[1];

  // Reinitialize posture to known position (gps fix)
  posture[0] = truePosit.x;
  posture[1] = truePosit.y;
  posture[2] = 0.0;

  // Add gain filtered error to previous errors
  current[0] += Kfour1 * (deltaX / deltaT);
  current[1] += Kfour2 * (deltaY / deltaT);

  // Save the time of the gps fix for next calculation
  lastGPStime = positTime;
}

/*************************************************************************
  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. Initilizes
  the times of the last GPS fix and last IMU information.
  RETURNS:void
  CALLED BY:initializeNavigator (nav)
  CALLS:     calulateBiasCorrections (ins)
                              getSample (sampler)
                              buildRotationMatrix (ins)
                              transformWaterSpeed (ins)

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

void
INS::insSetUp(double originTime)
{
  stampedSample newSample;
```

```
  //Set posture to straight and level at the origin.
  posture[0] = 0.0;
  posture[1] = 0.0;
  posture[2] = 0.0;
  posture[3] = 0.0;
  posture[4] = 0.0;

  //set imu biases
  calculateBiasCorrections();

  while(!sam1.getSample(newSample)) {/* */};

  //set initial true heading
  posture[5] = trueHeading(newSample.sample[7]);

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

  // initialize times
  lastTime = newSample.timeStamp;
  lastGPStime = originTime;
}

/*************************************************************************
  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
INS::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
INS::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
INS::transformBodyRates (double newSample[])
{
```

```
  vector earthRates = buildBodyRateMatrix() * &(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
INS::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
INS::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:post multiplication 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:      None
*************************************************************************/
```

```
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: trueHeading
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Convert magnetic direction to true based on local magnetic
                        variation.
  RETURNS:   true heading
  CALLED BY:  insPosit
                        insSetUp
  CALLS:     none
*****************************************************************************/

double
INS::trueHeading(const double magHeading)
{
  static double twoPi(2.0 * M_PI);
  double trueHeading = magHeading + RADIANMAGVAR;

  if (trueHeading > twoPi) {
        trueHeading -= twoPi;
  }

  return trueHeading;

}
```

```
/*************************************************************************
  PROGRAM:calculateBiasCorrections
  AUTHOR:Eric Bachmann, Dave Gay
  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
INS::calculateBiasCorrections()
{

  int biasNumber(10);
  stampedSample biasSample;

  biasCorrection[3] = 0.0;
  biasCorrection[4] = 0.0;
  biasCorrection[5] = 0.0;

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

        while(!sam1.getSample(biasSample)) {/* */};

              biasCorrection[3] += biasSample.sample[3];
              biasCorrection[4] += biasSample.sample[4];
              biasCorrection[5] += biasSample.sample[5];
  }

  biasCorrection[3] = -(biasCorrection[3]/biasNumber);
  biasCorrection[4] = -(biasCorrection[4]/biasNumber);
  biasCorrection[5] = -(biasCorrection[5]/biasNumber);
}
```

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

void
INS::applyBiasCorrections(double sample[])
{
  static const float biasWght(0.99), sampleWght(0.01);

  biasCorrection[3] = (biasWght * biasCorrection[3])
                                        - (sampleWght * sample[3]);
  biasCorrection[4] = (biasWght * biasCorrection[4])
                                        - (sampleWght * sample[4]);
  biasCorrection[5] = (biasWght * biasCorrection[5])
                                        - (sampleWght * sample[5]);

  sample[3] += biasCorrection[3];
  sample[4] += biasCorrection[4];
  sample[5] += biasCorrection[5];
}
```

J. SAMPLER.H

```
#ifndef __SAMPLER_H
#define __SAMPLER_H

#include <time.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>

#include "portbank.h"
#include "towtypes.h"
#include "packbuff.h"
```

```
#define xyAccelLimit ONE_G// Max accell in x and y diretion
#define zAccelLimit 2 * ONE_G  // Max accel in z direction
#define rateLimit  0.872665// Max rotational rate in radians
#define speedLimit 25.3 //Max water speed
#define headingLimit 2 * M_PI

#define rateUnits(angular) ((((angular-2048.0) / 2048.0 ) * 50.0) * (M_PI/180.0))
#define accelUnits(linear) (((linear-2048.0) / 2048.0 ) * GRAVITY)
#define zAccelUnits(linear) (((linear-2048.0) / 2048.0) * (2.0 * GRAVITY))
#define depthUnits(depth) (((depth - 819.0) / (4096.0-819.0)) * 180.0)
#define waterSpeedUnits(speed)  ((speed / 4096.0) * 25.3) //feet per second
#define headingUnits(heading)  (((heading - 81.92) / 4.096) * (M_PI/180.0))

/*************************************************************************
  CLASS:sampler
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Formats, timestamps, low pass filters and limit checks IMU,
  water-speed and heading information.
  COMMENTS: This class is extremely dependent upon the specific
  hardware configuration. It is designed to isolate to the INS from
  these particulars.
*************************************************************************/

class sampler {

  public:

        //Class Constructor
        sampler() : port1(COMports.Init(COM1, BYTE(4), b9600,
                                NOPARITY, BYTE(8), BYTE(1), NONE, packets)),
                                packets(port1) { }

         //checks for the arrival of a new sample and formats it.
         Boolean getSample(stampedSample&);

  private:

        //instantiates serial port communications on comm1
        bufferedSerialPort& port1;

        //buffer for gps data
```

```
        packetBuffer packets;

        //Averages the eight samples of a samplePacket
        void createSample(stampedSample& newSample);

        //Converts the voltages of a sample to real world units
        void formatSample (stampedSample&);

        // Returns TRUE if all sample values are within limits
        Boolean inLimitSample(stampedSample& newSample);

};


#endif
```

K. SAMPLER.CPP

```
#include <iostream.h>
#include "sampler.h"

/*************************************************************************
  PROGRAM: getSample
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Prepares raw sample data for use by the INS  object
  RETURNS:  TRUE, if a valid sample was obtained
  CALLED BY:insPosit (ins)
                              insSetup (ins)
  CALLS:      Get (packetBuffer)
                              createSample (sampler)
                              formatSample (sampler)
                              inLimitSample (sampler)
*************************************************************************/

//checks for the arrival of a new sample
Boolean
sampler::getSample(stampedSample& newSample)
{

  // Attempt to obtain sample
  if (packets.Get(newSample.samplePacket)) {
```

```
        //Time stamp the sample
        gettime(&newSample.timeStamp);

        //Low pass filter the sample
        createSample(newSample);

        //Convert Voltages to units
        formatSample(newSample);

        //Check for out of limit readings
        if (inLimitSample(newSample)) {
                return TRUE;
        }
        else {
                cerr << "\n Motion Pack Sample Out of Limits \n";
                return FALSE;
        }
  }
  else {

        return FALSE;  //Valid sample not available
  }
}

/*************************************************************************
  PROGRAM: createSample
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Low pass filters eight closely spaced sets of sensor
  readings by summing the readings of each sensor and computing the
  average.
  RETURNS:  void
  CALLED BY: getSample
  CALLS:    none
*************************************************************************/

void
sampler::createSample(stampedSample& newSample)
{
  UNSIGNED_TWOBYTE tempNum;

  for (int i = 0; i < 8; i++) {
        newSample.sample[i] = 0.0;
```

```
}

for (i = 3; i < 131;) {
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[0] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[1] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[2] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[3] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[4] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[5] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[6] += double(tempNum/16);
        tempNum = newSample.samplePacket[i++];
        tempNum = (tempNum << 8) + newSample.samplePacket[i++];
        newSample.sample[7] += double(tempNum/16);
 }

         newSample.sample[0] /= 8.0;
         newSample.sample[1] /= 8.0;
         newSample.sample[2] /= 8.0;
         newSample.sample[3] /= 8.0;
         newSample.sample[4] /= 8.0;
         newSample.sample[5] /= 8.0;
         newSample.sample[6] /= 8.0;
         newSample.sample[7] /= 8.0;

}
```

```
/*************************************************************************
  PROGRAM:formatSample
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Converts integers representing voltage readings into
  units which are useable by the INS.
  RETURNS: void
  CALLED BY:getSample
  CALLS:     none
*************************************************************************/

//Calls the methods to convert the voltages to real world units
void
sampler::formatSample (stampedSample& newSample)
{
  newSample.sample[0] = accelUnits(newSample.sample[0]);
  newSample.sample[1] = accelUnits(newSample.sample[1]);
  newSample.sample[2] = zAccelUnits(newSample.sample[2]);

  newSample.sample[3] = rateUnits(newSample.sample[3]);
  newSample.sample[4] = rateUnits(newSample.sample[4]);
  newSample.sample[5] = rateUnits(newSample.sample[5]);

  newSample.sample[6] = waterSpeedUnits(newSample.sample[6]);
  newSample.sample[7] = headingUnits(newSample.sample[7]);
}

/*************************************************************************
  PROGRAM:inLimitSample
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Checks sensor readings to insure they are with in the
  limits of the sensor from which they originated.
  RETURNS: True, If all readings are with in limits.
  CALLED BY:getSample
  CALLS:     none
*************************************************************************/

Boolean
sampler::inLimitSample(stampedSample& newSample)
{
  Boolean limitFlag(TRUE);
```

```
if (fabs(newSample.sample[0]) > xyAccelLimit) {
        limitFlag = FALSE;
        cerr << "\n imu x accel out of limits\n";
}
if (fabs(newSample.sample[1]) > xyAccelLimit) {
        limitFlag = FALSE;
        cerr << "\n imu y accel out of limits\n";
}
if (fabs(newSample.sample[2]) > zAccelLimit)  {
        limitFlag = FALSE;
        cerr << "\n imu z accel out of limits\n";
}
if (fabs(newSample.sample[3]) > rateLimit)    {
        limitFlag = FALSE;
        cerr << "\n imu p rate out of limits\n";
}
if (fabs(newSample.sample[4]) > rateLimit)    {
        limitFlag = FALSE;
        cerr << "\n imu q rate out of limits\n";
}
if (fabs(newSample.sample[5]) > rateLimit)    {
        limitFlag = FALSE;
        cerr << "\n imu r rate out of limits\n";
}
if (fabs(newSample.sample[6]) > speedLimit)   {
        limitFlag = FALSE;
        cerr << "\n water speed out of limits\n";
}

if (fabs(newSample.sample[7]) > headingLimit) {
        limitFlag = FALSE;
        cerr << "\n heading out of limits\n";
}

return limitFlag;
}
```

APPENDIX B: Serial Port Communication Source Code (C^{++})

A. GLOBALS.H

```
#ifndef __GLOBALS_H
#define __GLOBALS_H

// types
typedef  unsigned charBYTE;
typedef  unsigned short WORD;
typedef  unsigned long  DWORD;

#define  MEM(seg,ofs)    (*((BYTE far*)MK_FP(seg,ofs)))
#define  MEMW(seg,ofs)   (*((WORD far*)MK_FP(seg,ofs)))

enum Boolean   {FALSE, TRUE};

// basic bit twiddles
#define  set(bit)          (1<<bit)
#define  setb(data,bit)      data | set(bit)
#define  clrb(data,bit)      data & !set(bit)
#define  setbit(data,bit)    data = setb(data,bit)
#define  clrbit(data,bit)    data = clrb(data,bit)

// specific to ports
#define  setportbit(reg,bit)  outportb(reg,setb(inportb(reg),bit))
#define  clrportbit(reg,bit)  outportb(reg,clrb(inportb(reg),bit))

#endif
```

B. BUFFER.H

```
#ifndef __BYTEBUFFER_H
#define __BYTEBUFFER_H

#include "towtypes.h"

#define  BYTEBUFSIZE   32
```

```
/*************************************************************************
  CLASS:Buffer
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
  Base class for use as a polymorphic reference in the serial port code
  which defines a buffer to be used in serial port communications.
*************************************************************************/

class Buffer  {

  public:

        //Constructor
        Buffer(WORD sz) : getPtr(0), putPtr(0), size(sz)  { }

        //Checks for the arrival of new characters in the buffer
        virtual Boolean  hasData()      { return Boolean(putPtr != getPtr); }

        //How much of the Buffer is used (rounded percentage 0 - 100)
        virtual int     capacityUsed();
        //Read from the buffer
        virtual Boolean   Get(BYTE&) = 0;
        //Read to the buffer
        virtual void      Add(BYTE) = 0;

  protected:

        //Increment the pointer to next position
        void  inc(WORD& index)
                { if (++index == size) index = 0; }

        //Decrement the pointer
        WORD before(WORD index)
                { return ((index == 0) ? size - 1 : index - 1);}

        WORD    getPtr, //Location of unread data
                        putPtr, //Location to read data to
                        size;   //Size of the buffer in bytes
};
```

```
/*************************************************************************
 Defines a single buffer of a specified size for buffering charaters
 received via serial port.
*************************************************************************/

class byteBuffer : protected Buffer {

  public:

        byteBuffer(BYTE sz=BYTEBUFSIZE);
        ~byteBuffer()    { delete [] buf; }

        Buffer::hasData;

        //
        Buffer::capacityUsed;

        // buffer extraction
        Boolean    Get(BYTE&);

        // buffer insertion
        void    Add(BYTE ch);
        void    Add(const char*);
        byteBuffer&  operator += (BYTE ch)     { Add(ch); return *this; }

  protected:

        BYTE*    buf;
};

#endif
```

C. BUFFER.CPP

```
#include <iostream.h>
#include <stdio.h>

#include "globals.h"
#include "packbuff.h"

//*****************************************
// Buffer
//*****************************************
```

```
// Returns the percentage of the buffer in use

int
Buffer::capacityUsed()
{
  int cap = (putPtr + size) % size - getPtr;
  return 100 * cap / size;
}

//*****************************************
// byteBuffer
//*****************************************

//Constructor, instantiates a buffer
byteBuffer::byteBuffer(BYTE sz) : Buffer(sz)
{
  buf = new BYTE[size];
}

//Reads a charcter from the buffer

Boolean
byteBuffer::Get(BYTE& data)
{
  if (hasData()) {
          data = buf[getPtr];
          inc(getPtr);
          return TRUE;
  }
  return FALSE;
}

//Writes a character to the buffer and checks for buffer overflow

void
byteBuffer::Add(BYTE ch)
{
  buf[putPtr] = ch;
  inc(putPtr);
  if (!hasData()) {   // if there's no data after adding data, it overflowed
        cerr << "\nError: byteBuffer overflow\n";
  }
}
```

```
//Writes a character to the buffer

void
byteBuffer::Add(const char* s)
{
  while (*s)
        Add(*s++);
}
```

D. PACKBUFF.H

```
#ifndef __PACKBUFF_H
#define __PACKBUFF_H

#include "buffer.h"
#include "towtypes.h"

#define PACKETBLOCKS   10
#define SOH   0x01
#define NAK   0x15
#define ACK   0x06
#define EndofFile   0x1a
#define EOT   0X04
#define SecsToTicks(x)  ((long)(x * 182L) / 10L)

class bufferedSerialPort;

/*************************************************************************
  CLASS:packetBuffer
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Class buffers packets of data received via a modified XMODEM protocol.
  Uses a multiple buffer system in which each buffer is capable of
  holding a single packet of information. Buffers are filled and processed
  sequentially in a round robin fashion. Packets are checked for validity
  only upon attempted reads from the buffer.
*************************************************************************/

class packetBuffer : public Buffer  {

public:

  packetBuffer(bufferedSerialPort&, BYTE packetBlocks=PACKETBLOCKS);
```

```
  ~packetBuffer(){delete [] block;}

  Boolean  hasData();     // check a block for valid packet
  Boolean  Get(BYTE&) {return FALSE;} //Satifies inheritence requirements
  Boolean  Get(PACKET);   // get a complete structure filled in
  void     Add(BYTE ch);  // build a packet as each byte is added

protected:

  Boolean validPacket(PACKET); //check the checksum of a packet
  Boolean timeOut();       //check for an event time out

  bufferedSerialPort& port; //com port to ACK and Nak acknowledgements

  PACKET  *block;          // hold the buffered sample packet
  WORD    current,         // the current block in use
                  last;    // the last block in use

  BYTE    byteIndex;  // place to put the next charater received
  long    waitTime;   // time waiting since last time out
  Boolean  arriveFlag; //indicates if charater arrived since last check
 };

#endif
```

E. PACKBUFF.CPP

```
#include <iostream.h>
#include <stdio.h>
#include "serial.h"
#include "packbuff.h"

//*****************************************
// packetBuffer
//*****************************************
```

```
/*************************************************************************
  PROGRAM: Packet Buffer Constructor
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:    Constructor, Sets up number of buffers. Sets pointers to write into
  the buffers. Sends first NAK to start XMODEM protocol.
  CALLS:send (serialPort)
*************************************************************************/

packetBuffer::packetBuffer(bufferedSerialPort& p, BYTE packetBlocks) :
  port(p), current(0), last(0), byteIndex(0), waitTime(SecsToTicks(10)),
  arriveFlag(FALSE), Buffer(packetBlocks)
{
  block = new PACKET[packetBlocks];
  port.Send(NAK);
}
/*************************************************************************
  PROGRAM: Add
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION: Reads a single character into the buffer array. Check for
  end of a packet before moving to next buffer. Sends ACK and NAK following
  each Packet.
  RETURNS:   void
  CALLED BY: interupt driven
  CALLS: none
*************************************************************************/
void
packetBuffer::Add(BYTE data){

  static Boolean EOTflag = FALSE;

 if (data == EOT && EOTflag) {
        port.Send(ACK);    //Acknowledge End of transmission character
        port.Send(NAK); //Request next packet
  }


  if (EOTflag && (data == SOH)) {

        //setup for the start of a packet
        last = current;
        inc(current);
```

```
        byteIndex = 0;
        EOTflag = FALSE;
  }

  arriveFlag = TRUE;
  block[current][byteIndex++] = data; //write character to buffer


 if (byteIndex == 131) {
        port.Send(ACK);      //Acknowledge packet
        waitTime = SecsToTicks(1L); //Reset wait time to one second
        EOTflag = TRUE;          //Start waiting for the EOT character
  }
}

/*************************************************************************
  PROGRAM: Get
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Reads a packet from the buffer into the input argument
  RETURNS:  TRUE, if a packet was read
  CALLED BY: getSample (sampler)
  CALLS:      timeOut (packBuf)
                              hasData (packBuf)
                              inc (packBuf)
*************************************************************************/

Boolean
packetBuffer::Get(GPSdata data)
{
  if (arriveFlag) {

        arriveFlag = FALSE;//Reset flag to indicate new character arrivals

        if (hasData())  {
              memcpy (data, block[last], PACKETSIZE);
              last = current;
              return TRUE;    //valid packet received
        }
        else {
              return FALSE;
        }
  }
```

```
  else {
        if (timeOut()) {

              // Possible race condition if characters arrive during these stmt
              // Assume lost character go to beginning of next buffer
              byteIndex = 0;
              inc(current);

              port.Send(NAK);              //indicate lack timeout to sender
              waitTime = SecsToTicks(1L);     //set wait time to one second
        }
        return FALSE;
  }
}
/*************************************************************************
  PROGRAM: hasData
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Determine if a valid new packet has arrived.
  RETURNS:  TRUE, if a new unread packet is in a buffer
  CALLED BY:Get (packbuff)
  CALLS:      validPacket
*************************************************************************/
Boolean
packetBuffer::hasData()
{
  if (last != current) {
        if (validPacket(block[last])) {
              return TRUE; //New valid packet available
        }
        else {
              last = current;
              return FALSE;
        }
  }
  else {
        return FALSE;
  }
}
```

```
/*************************************************************************
  PROGRAM: validPacket
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION: Checks a packet for sequence number and complement and
  checks for valid checksum.
  RETURNS:  TRUE if the packet is valid
  CALLED BY:hasData (packBuff)
  CALLS:      none
*************************************************************************/
Boolean
packetBuffer::validPacket(PACKET packet)
{
  BYTE sum(0);

  if (char(packet[1]) == ~char(packet[2])) {

        for (int i = 3; i < 131; i++) {
              sum += packet[i];
        }

        if (sum == packet[131]) {
//            cerr << " valid packet" << endl;
              return TRUE;
        }
        else {
              cerr << "invalid packet checksum " << endl;
              return FALSE;
        }
  }

  cerr << "invalid packet sequence number" << endl;
  return FALSE;
}
```

```
/*************************************************************************
  PROGRAM:timeOut
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Determine if a time out has occured since the last character
  was received by checking the wait time slot.
  RETURNS:  TRUE, if a time out has occured
  CALLED BY:Get
  CALLS:      none
*************************************************************************/
Boolean
packetBuffer::timeOut()
{
  static unsigned long far *timer = (unsigned long far *) MK_FP(0x40, 0x6c),
                                        currentTime,
                                        lastTime;

  currentTime = *timer;// Get the current time.

  if (waitTime > 0) {      // New wait period?
        waitTime *= -1; // Go negative, count up
        lastTime = currentTime; //Save time to later determine time delta
  }

  if (currentTime != lastTime) {  // Has time passed?
        if (currentTime <= lastTime) {
              waitTime++;
        }
        else {
              waitTime += (unsigned int) (currentTime - lastTime);
        }
        lastTime = currentTime;  //Save time to later determine time delta
  }

  return Boolean(waitTime >= 0L);  // Return TRUE is zero has been reached
}
```

F. GPSBUFF.H

```
#ifndef __GPSBUFF_H
#define __GPSBUFF_H
```

```
#include "buffer.h"
#include "towtypes.h"


#define  GPSBLOCKS     4
#define  LINE_FEED     10

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

  Class buffers GPS position messages via serial port communications.
  Uses a multiple buffer system in which each buffer is capable of
  holding a single position message. Buffers are filled and processed
  sequentially in a round robin fashion. Messages are checked for validity
  only upon attempted reads from the buffer.

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

class GPSbuffer : public Buffer  {

public:

   GPSbuffer(BYTE GPSblocks=GPSBLOCKS);
   ~GPSbuffer(){delete [] block;}

   Boolean  hasData();     // a complete structure is ready
   Boolean  Get(BYTE&)   {return FALSE;}
   Boolean  Get(GPSdata);  // get a complete structure filled in
   void     Add(BYTE ch);  // build the structure as each byte is added

protected:

   Boolean validHeader(GPSdata); // check a block for valid header
   GPSdata  *block;     // hold the buffered GPS data
   WORD     current,    // the current GPS block in use
                        last;       // the last GPS block in use

   BYTE     *putPlace;  // place to put the next charater received
};

#endif
```

G. GPSBUFF.CPP

```
#include <iostream.h>
#include <stdio.h>

#include "gpsbuff.h"

//*****************************************
// GPSbuffer
//*****************************************
/*************************************************************************
  PROGRAM:GPSbuffer (Constructor)
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Allocates message buffers, indicate that no data has been
        received by equalizing current and last and set position
                into which initial character will be read.
  RETURNS:nothing.
  CALLED BY:navigator class (nav.h)
  CALLS:none.
*************************************************************************/

GPSbuffer::GPSbuffer(BYTE GPSblocks) :
  current(0), last(0), Buffer(GPSblocks)
{
  block = new GPSdata[GPSblocks];
  putPlace = &(block[current][0]);
}

/*************************************************************************
  PROGRAM:GPSbuffer::Add
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Interrupt driven routine which writes incoming characters into
        into the gps buffers
  RETURNS:nothing.
  CALLED BY:  interupt driven by bufferedSerialPort
  CALLS:none.
*************************************************************************/

void
GPSbuffer::Add(BYTE data){
```

```
  static Boolean lfFlag = FALSE;

  //Is a new message starting?
  if (lfFlag && (data == '@')) {

        last = current;  // Set last to buffer with newest message.
        inc(current);    // Set current to the next buffer
        // Set putPlace to the beginning of the next buffer.
        putPlace = &(block[current][0]);
        lfFlag = FALSE;  // reset for end of next message.
  }

  *putPlace++ = data;// Write character into the buffer.

  //Has the end of a message been received?
  if (data == LINE_FEED) {
        lfFlag = TRUE;
  }
}

/*************************************************************************
  PROGRAM:GPSbuffer::Get
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Checks to see if a new message has arrived, copies it into the
        input argument data and returns a flag to indicate whether a new
                message was received.
  RETURNS:TRUE, if a new valid position has been received.
                FALSE, otherwise
  CALLED BY:navPosit (nav.cpp)
                initializeNavigator (nav.cpp)
  CALLS:GPSbuffer::hasData
*************************************************************************/

Boolean
GPSbuffer::Get(GPSdata data)
{
  // Has a new valid message been received.
  if (hasData()) {
                // Copy the message out of the buffer.
          memcpy (data, block + last, GPSBLOCKSIZE);
```

```
                // Indicate that this message has been read.
        last = current;
        return TRUE;
  }
  else {
        return FALSE;
  }
}

/*************************************************************************
  PROGRAM:GPSbuffer::hasData
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Determines whether a new message has been received and checks
                to see if it has a valid header.
  RETURNS:TRUE, if a new valid message has been received.
  CALLED BY:  GPSbuffer::Get (buffer.cpp)
  CALLS:      validHeader (buffer.cpp)
*************************************************************************/

Boolean
GPSbuffer::hasData()
{
  if ((last != current) && (validHeader(block[last]))) {
        return TRUE;
  }
  else {
        return FALSE;
  }
}
/*************************************************************************
  PROGRAM:validHeader
  AUTHOR:Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Checks to see if a message has the proper header for a Motorola
                position message. (@@Ba)
  RETURNS:TRUE, if the header is valid. FALSE, otherwise.
  CALLED BY:GPSbuffer::hasData (buffer.cpp)
  CALLS:none.
  COMMENTS:
*************************************************************************/
```

```
Boolean
GPSbuffer::validHeader(GPSdata dataPtr)
{
 if ((dataPtr[0] == '@') &&
         (dataPtr[1] == '@') &&
         (dataPtr[2] == 'B') &&
         (dataPtr[3] == 'a')) {
        return TRUE;
 }
 else {
        return FALSE;
 }
}
```

H. PORTBANK.H

```
#ifndef __PORTBANK_H
#define __PORTBANK_H

#include "serial.h"
#include "buffer.h"

/*************************************************************************
  CLASS:portBank
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Manages up to four bufferdSerialPort instances.
*************************************************************************/

class portBank {
public:

  portBank();

  ~portBank()   { cleanup(); }

  bufferedSerialPort&  Init(COMport portnum, BYTE irq, BaudRate, ParityType,
                        BYTE wordlen, BYTE stopbits, handShake, Buffer&);
  void  cleanup();

  friend IntHandlerType COM1handler, COM2handler, COM3handler, COM4handler;

protected:
```

```
  bufferedSerialPort*  ports[4];

};

extern portBank   COMports;

#endif
```

I. PORTBANK.CPP

```
#include <iostream.h>

#include "serial.h"
#include "buffer.h"
#include "portbank.h"

portBank    COMports;

// Constructor, sets up array of ports

portBank::portBank()
{
  for (int i = 0; i < 4; i++)
        ports[i] = 0;
}

// Resets all ports to the original parameters

void
portBank::cleanup()
{
  for (int i=0; i<4; i++)
        delete ports[i];
}

// Initializes a serial port based up the input arguments

bufferedSerialPort&
portBank::Init(COMport portnum, BYTE irq, BaudRate baud, ParityType parity,
                        BYTE wordlen, BYTE stopbits, handShake shake, Buffer& buf)
{
  int   index = BYTE(portnum) - 1;
```

```
  if (ports[index])
        delete ports[index];
  ports[index] = new bufferedSerialPort(portnum, irq, baud, parity,
                                        wordlen, stopbits, shake, buf);
  return *ports[index];
}

// Three specific interrupt handlers which map each interupt to the
// proper ISR.

void
interrupt
COM1handler(...)
{
  COMports.ports[0]->processInterrupt();
  EOI;
}

void
interrupt
COM2handler(...)
{
  COMports.ports[1]->processInterrupt();
  EOI;
}

void
interrupt
COM3handler(...)
{
  COMports.ports[2]->processInterrupt();
  EOI;
}

void
interrupt
COM4handler(...)
{
  COMports.ports[3]->processInterrupt();
  EOI;
}
```

J. SERIAL.H

```
#ifndef __SERIAL_H
#define __SERIAL_H

#include <dos.h>
#include <stdio.h>
#include "globals.h"
#include "buffer.h"

// user defines
#define   ALMOST_FULL   80   // % full to turn off DTR

// leave the following alone - hardware specific

enum COMport      {COM1=1, COM2, COM3, COM4};
enum BaudRate     {b300, b1200, b2400, b4800, b9600};
enum ParityType   {ERROR=-1, NOPARITY, ODD, EVEN};
enum handShake    {NONE, RTS_CTS, XON_XOFF};
enum Shake        {off, on};
enum interruptType {rx_rdy, tx_rdy, line_stat, modem_stat};

#define BIOSMEMSEG 0x40
#define DLAB       0x80
#define IRQPORT    0x21
#define EOI        outportb(0x20,0x20)

#define COM1base   MEMW(BIOSMEMSEG,0)
#define COM2base   MEMW(BIOSMEMSEG,2)
#define COM3base   0x03e8
#define COM4base   0x02e8

#define TX         (portBase)
#define RX         (portBase)
#define IER        (portBase+1)
#define IIR        (portBase+2)
#define LCR        (portBase+3)
#define MCR        (portBase+4)
#define LSR        (portBase+5)
#define MSR        (portBase+6)
#define LO_LATCH   (portBase)
#define HI_LATCH   (portBase+1)
```

```
/*************************************************************************
  CLASS:serialPort
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Defines a simple serial port.
*************************************************************************/

class serialPort  {

  public:

        serialPort(COMport port, BaudRate, ParityType, BYTE wordlen,
                        BYTE stopbits, handShake);

        Boolean    Send(BYTE data);
        Boolean    Get(BYTE& data);

        inline Boolean dataReady();
        Boolean statusChanged()
                        { return Boolean((ifportbit(MSR,0) || ifportbit(MSR,1))); }

        // the rest are only if handshake is specified as RTS_CTS
        Boolean    isCTSon()        { return ifportbit(MSR,4); }
        Boolean    isDSRon()        { return ifportbit(MSR,5); }

        void       setDTRon()       { setportbit(MCR,0); }
        void       setDTRoff()      { clrportbit(MCR,0); }
        void       toggleDTR();

        void       setRTSon()       { setportbit(MCR,1); }
        void       setRTSoff()      { clrportbit(MCR,1); }
        void       toggleRTS();

  protected:

        WORD         portBase;
        handShake    ShakeType;
        Shake        DTRstate,
                                    RTSstate;

        inline Boolean    ifportbit(WORD, BYTE);
        inline void       toggle(Shake&);
```

```
};

// this is the type for a standard interrupt handler
typedef void interrupt (IntHandlerType)(...);


/*************************************************************************
  CLASS:bufferedSerialPort
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Defines a buffered serial port which is interrupt driven
  on receive, and buffers all incoming characters in the specified buffer
*************************************************************************/

class bufferedSerialPort : public serialPort  {

  public:

        bufferedSerialPort(COMport portnum, BYTE irq, BaudRate, ParityType,
                                    BYTE  wordlen,  BYTE  stopbits,  handShake,
Buffer& );

        ~bufferedSerialPort();

        Boolean    Get(BYTE& data);    // buffered version

  protected:

        Buffer&    buf;

        BYTE       irqbit,    //Value to allow enable PIC interrupts for COM port
                              origirq,   //keep the original value of the 8259 mask register
                              comint;

        void  processInterrupt();    // buffers the incoming character

        IntHandlerType *origcomint;  // keep the original vector for restoring later

        // this allows the actual handlers to access processInterrupt()
            friend   IntHandlerType   COM1handler,   COM2handler,   COM3handler,
 COM4handler;
```

```
};

#endif
```

K. SERIAL.CPP

```
#include <iostream.h>
#include <stdio.h>
#include "serial.h"

/*************************************************************************
        Usage Note: Because of the interrupt handlers used, you MUST call
  your bufferedSerialPort objects port1, port2, or port3, so the
  right handler gets called and can properly service the interrupt.
*************************************************************************/

/*************************************************************************
  PROGRAM: serialPort (Constructor)
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Initializes the one of the Serial Ports.
                1) Determines the base I/O port address for the given COM port
                2) Sets the 8259 IRQ mask value
                3) Initializes the port parameters - baud, parity, etc.
                4) Calls the routine to initialize interrupt handling
                5) Enables DTR and RTS, indicating ready to go
*************************************************************************
**/

serialPort::serialPort(COMport port, BaudRate speed, ParityType parity,
                                        BYTE wordlen, BYTE stopbits, handShake hs) :
  DTRstate(off), RTSstate(off), ShakeType(hs)
{
  switch (port) {
        case COM1:  portBase = COM1base;
                                        break;
        case COM2:  portBase = COM2base;
                                        break;
        case COM3:  portBase = COM3base;
                                        break;
        case COM4:  portBase = COM4base;
                                        break;
```

```
 } //switch

 const WORD    bauddiv[] = {0x180, 0x60, 0x30, 0x18, 0xC};
        // Change 1
 outportb(IER,0);      // disable UART interrupts
 (void)inportb(LSR);
 (void)inportb(MSR);
 (void)inportb(IIR);
 (void)inportb(RX);
 outportb(LCR,DLAB);   //set DLAB so can set baud rate (read only port)
 outportb(LO_LATCH,bauddiv[speed] & 0xFF);
 outportb(HI_LATCH,(bauddiv[speed] & 0xFF00) >> 8);
 setportbit(MCR,3);              //turn OUT2 on

 BYTE  opt = 0;
 if (parity != NOPARITY)  {
        setbit(opt,3);      // enable parity
        if (parity == EVEN)  // set even parity bit. if odd, leave bit 0
              setbit(opt,4);
 }
         // now set the word length. len of 5 sets both bits 0 and 1 to
         // 0, 6 sets to 01, 7 to 10 and 8 to 11
 opt |= wordlen-5;
 opt |= --stopbits << 2;
 outportb(LCR,opt);

 if (ShakeType == RTS_CTS)  {
        setDTRon();
        setRTSon();
 }
}

/*************************************************************************
  PROGRAM: Get
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Gets a byte from the port. Returns true if there's one there, and fills
        in the byte parameter. If theres no character, the parameter is left
        alone, and false is returned
*************************************************************************/
```

```
Boolean
serialPort::Get(BYTE& data)
{
  if (dataReady()) {       //make sure there's a char there
        data = inportb(RX);    //read character from 8250
        return TRUE;
  }
  else
        return FALSE;
}

/*************************************************************************
  PROGRAM: Send
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
       Outputs a single character to the port. Returns Boolean
       status indicating whether successful
*************************************************************************/

Boolean
serialPort::Send(BYTE data)
{
  while (!(ifportbit(LSR,5)))     // wait until THR ready
        ; // NULL statement

  switch (ShakeType) {
        case NONE:    outportb(TX,data);
                                    return TRUE;
        case RTS_CTS:  if (isCTSon() && isDSRon()) {
                                    outportb(TX,data);
                                    return TRUE;
                                     }
                                     else  return FALSE;
        case XON_XOFF:
        default:
              // add this later if needed
                    break;
  }
  return FALSE;
}
```

```
/*************************************************************************
  PROGRAM: dataReady
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:Checks port to see if a character has arrived.
*************************************************************************/

inline
Boolean
serialPort::dataReady()
{
/*
  if (ifportbit(LSR,1)) {
        cerr <<"\nOverrun Error\n";
  }
  if (ifportbit(LSR,2)) {
        cerr <<"\nParity Error\n";
  }
  if (ifportbit(LSR,3)) {
        cerr <<"\nFraming Error\n";
  }
*/
  return ifportbit(LSR,0);
}

/*************************************************************************
  PROGRAM: ifportbit
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
*************************************************************************/

inline
Boolean
serialPort::ifportbit(WORD reg, BYTE bit)
{
  BYTE on = inportb(reg);
        on &= set(bit);
  return Boolean(on == set(bit));
}
```

```
/*************************************************************************
  PROGRAM: toggleDTR
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION: toggles Data Transmit Ready if RTS_CTS is off
*************************************************************************/

void
serialPort::toggleDTR()
{
  if (ShakeType != RTS_CTS)
        return;
  if (DTRstate == off)
        setDTRon();
  else
        setDTRoff();
  toggle(DTRstate);
}

/*************************************************************************
  PROGRAM: toggleRTS
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION: toggle Ready to Send (RTS) if RTS_CTS is on.
*************************************************************************/

void
serialPort::toggleRTS()
{
  if (ShakeType != RTS_CTS)
        return;
  if (RTSstate == off)
        setRTSon();
  else
        setRTSoff();
  toggle(RTSstate);
}
```

```
/*************************************************************************
  PROGRAM: toggle
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION: toggles value of the input variable
*************************************************************************/

inline
void
serialPort::toggle(Shake& h)
{
  if (h == off)
        h = on;
  else
        h = off;
}

// ********************************************************************
//   bufferedSerialPort
// ********************************************************************

/*************************************************************************
  PROGRAM:bufferedSerialPort (Constructor)
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Initializes the interrupts for the Serial Port.
                        1) takes over the original COM interrupt
                        2) sets the port bits, parity, and stop bit
                        3) enables interrupts on the 8250 (async chip)
                        4) enables the async interrupt on the 8259 PIC
*************************************************************************/

bufferedSerialPort::bufferedSerialPort(COMport portnum, BYTE irq,
                                       BaudRate baud, ParityType parity, BYTE wordlen,
                                       BYTE stopbits, handShake hs, Buffer& b)
  : serialPort(portnum, baud, parity, wordlen, stopbits, hs),
        buf(b), irqbit(irq), comint(irqbit+8)
{
  if (ShakeType == RTS_CTS)  {   // turn it off first, because it was enabled
        setDTRoff();            // in the base class
        setRTSoff();
  }
```

```
  origcomint = getvect(comint);      //remember the original vector

  switch (portnum) {
        case COM1:
          setvect(comint,COM1handler);      //point to the new handler
          break;
        case COM2:
          setvect(comint,COM2handler);      //point to the new handler
          break;
        case COM3:
          setvect(comint,COM3handler);      //point to the new handler
          break;
        case COM4:
          setvect(comint,COM4handler);      //point to the new handler
          break;
  }

//  setportbit(MCR,3);              //turn OUT2 on
  disable();    // disable all interrupts - critical section
  setportbit(IER,rx_rdy);          //enable ints on receive only
  origirq = inportb(IRQPORT);       //remember how it was
  clrportbit(IRQPORT,irqbit);       //enable COM ints

  if (ShakeType == RTS_CTS)  {
        setDTRon();
        setRTSon();
  }
  enable();
  EOI;
}

/*************************************************************************
  PROGRAM: ~bufferedSerialPort
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:
        Resets the interrupts.  1) disables the 8250 (async chip)
                                2) disables the interrupt chip for async int
                                3) resets the 8259 PIC

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

```
bufferedSerialPort::~bufferedSerialPort()
{
  setvect(comint,origcomint);  //set the interrupt vector back
  outportb(IER,0);             //disable further UART interrupts
  outportb(MCR,0);             //turn everything off
  outportb(IRQPORT,origirq);
  EOI;
}

/*************************************************************************
  PROGRAM: Get
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Calls Get base on buffer type
*************************************************************************/

Boolean
bufferedSerialPort::Get(BYTE& data)
{
  return buf.Get(data);
}

/*************************************************************************
  PROGRAM: processInterupt
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:  Calls the ISR based upon buffer type
*************************************************************************/

void
bufferedSerialPort::processInterrupt()
{
  if (dataReady()) {       //make sure there's a char there
        BYTE data = inportb(RX);   //read character from 8250
        buf.Add(data);
        if (ShakeType == RTS_CTS && buf.capacityUsed() > ALMOST_FULL)
              setDTRoff();
  }
}
```

```
/*************************************************************************
  PROGRAM: showPorts
  AUTHOR:Frank Kelbe, Eric Bachmann, Dave Gay
  DATE:11 July 1995
  FUNCTION:   Prints interupt vector addresses
*************************************************************************/

int
showPorts()
{
  BYTE* p = (BYTE*)COM2base;
        p += 5;
        fprintf(stderr,"%X   ",*p++);
        fprintf(stderr,"%X\n",*p++);
  fprintf(stderr,"IRQPORT = %X", inportb(IRQPORT));
  return 0;
}
```

APPENDIX C: Navigation Simulation Source Code (LISP)

; OOD, Engineer, and Weapons classes written by Brad Leonhardt (Fall 94)

; Written for CS4314 final project

A. OOD.CL

```
;
;----------------------ENGINEER CLASS AND METHODS-------------------------
(defclass engineerclass()() )

(defmethod order((engineer engineerclass)string)
   (write-string "        ENG --> EOOW : ")
   (write string)
   (write-line "!"))

(defmethod query((engineer engineerclass)string)
   (write-string "        ENG --> EOOW :  ")
   (write string)
   (write-string "?")
   (write-string " ")
   (member (read) *affirmative*))

;----------------------WEAPONS OFFICER CLASS AND METHODS--------------------
(defclass weapons-officerclass()() )

(defmethod order((weapons-officer weapons-officerclass)string)
   (write-string "        WEPS --> SONAR: ")
   (write string)
   (write-line "!"))

(defmethod query((weapons-officer weapons-officerclass)string)
   (write-string "        WEPS --> SONAR: ")
   (write string)
   (write-string "?")
   (write-string " ")
   (member (read) *affirmative*))
```

```
;------------------------OOD CLASS AND METHODS-----------------------------
(defclass oodclass ()

  ( (engineer
     :initform (make-instance 'engineerclass)
     :accessor engineer)
    (navigator
     :initform (make-instance 'navigatorclass)
     :accessor navigator)
    (weapons-officer
     :initform (make-instance 'weapons-officerclass)
     :accessor weapons-officer) ))

(defmethod command((ood oodclass) string person)
   (write-string "    OOD --> ")
   (write person)
   (write-string " :")
   (write string)
   (write-line "!")
   (order person string))

(defmethod ask ((ood oodclass) string person)
   (write-string "    OOD --> ")
   (write person)
   (write-string " :")
   (write string)
   (write-line "?")
   (query person string))

(setf *affirmative* '(1 t y))

(defmethod goto-wp ((ood oodclass))
  (do* ()
    ((target-point-reached (navigator ood)) (order ood 'Get_next-waypoint))
    (query ood 'Waypoint_reached)))

;----------------------------OOD ORDERS-----------------------------------

(defmethod order((ood oodclass) order)
  (increment-time auv-clock)
```

```
(case order
  (Initialize_vehicle
    (and (command ood order (engineer ood))
         (command ood order (navigator ood) )
         (command ood order (weapons-officer ood)) ))

  (Transit_to_task_location
   (command ood order (navigator ood)))

  (Search_area_for_target
        (command ood order (weapons-officer ood)))

  (Commence_task_on_target
        (command ood order (weapons-officer ood)))

  (Transit_to_recovery_point
   (command ood order (navigator ood)))

  (Abort_mission
     (cond
         ((ask ood 'Is_recovery_point_obtainable (navigator ood))
          (setf *current_phase* 8))
         (t (command ood order (engineer ood)))))

  (Mission_complete
        (command ood order (engineer ood)))

  (Get_next-waypoint
   (command ood order (navigator ood)))

  (Get_GPS_fix
         (command ood order (navigator ood)))

  (Loiter_and_Start_Global_Replanner
         (command ood order (navigator ood)))

  (Log_new_obstacle
         (command ood order (navigator ood)))
  (Loiter_and_Start_Local_Replanner
         (command ood order (navigator ood)))

  (Drop_package
         (command ood order (weapons-officer ood)))
```

```
    (t nil) ))

;-----------------------------OOD QUERIES-----------------------------------
(defmethod query ((ood oodclass) question)

  (increment-time auv-clock)

  (case question

    (Critical_Systems_OK
     (cond ((ask ood question (engineer ood))
            (write-line "OOD -->CO: yes" ) t)
           (t (write-line "OOD -->CO: no")) ))

    (Initialization_complete
     (cond ((and (ask ood question (weapons-officer ood))
                 (ask ood question (navigator ood))
                 (ask ood question (engineer ood))
                 (write-line "OOD -->CO: yes" )) t)
           (t (write-line "OOD -->CO: no")) ))

    (Initialization_aborted
     (cond ((and (not (ask ood question (weapons-officer ood)))
                 (not (ask ood question (navigator ood)))
                 (not (ask ood question (engineer ood)))
                 (write-line "OOD -->CO: no" )))
           (t (write-line "OOD -->CO: yes") t)))

    (Task_location_reached
     (cond ((ask ood question (navigator ood))
            (write-line "OOD -->CO: yes" ) t)
           (t (write-line "OOD -->CO: no")) ))

    (Target_found
     (cond ((ask ood question (weapons-officer ood))
            (write-line "OOD -->CO: yes" ) t)
           (t (write-line "OOD -->CO: no")) ))

    (At_recovery_point
     (cond ((ask ood question (navigator ood))
            (write-line "OOD -->CO: yes" ) t)
           (t (write-line "OOD -->CO: no")) ))
```

```
(Waypoint_reached
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(Waypoint_process_OK
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(Got_next_waypoint
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(GPS_fix_needed
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(GPS_fix_obtained
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(Abort_GPS_fix
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(NonCritical_Systems_OK
 (cond ((and (ask ood question (weapons-officer ood))
             (ask ood question (engineer ood)) )
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(Area_clear_of_uncharted_obstacles
 (cond ((ask ood question (navigator ood))
       (write-line "OOD -->CO: yes" ) t)
      (t (write-line "OOD -->CO: no")) ))

(New_obstacle_logged
 (cond ((ask ood question (navigator ood))
```

```
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Log_system_failure
 (cond ((ask ood question (navigator ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Setpoints_and_modes_sent
 (cond ((ask ood question (navigator ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Setpoints_and_modes_system_OK
 (cond ((ask ood question (navigator ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Search_pattern_completed
 (cond ((ask ood question (weapons-officer ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Sonar_failure
 (cond ((ask ood question (weapons-officer ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Standoff_distance_reached
 (cond ((ask ood question (weapons-officer ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Abort_homing
 (cond ((ask ood question (weapons-officer ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))

(Is_package_dropped
 (cond ((ask ood question (weapons-officer ood))
      (write-line "OOD -->CO: yes" ) t)
     (t (write-line "OOD -->CO: no")) ))
```

```
(Is_package_drop_aborted
 (cond ((ask ood question (weapons-officer ood))
       (write-line "OOD -->CO: yes" ) t)
       (t (write-line "OOD -->CO: no")) ))

(t nil) ))
```

B. NAVIGATOR.CL

```
(setf auv-clock (make-instance 'clock-class))

(setf auv (make-instance 'sub-class))

;----------------------NAVIGATOR CLASS AND METHODS-----------------------
(defclass navigatorclass()
 ((replanner
   :initform (make-instance 'replannerclass)
   :accessor replanner)
  (gps-obtained-flag
   :initform 'no
   :accessor gps-obtained-flag)
  (last-position
   :initform '(0 0 0)
   :accessor last-position)
  (target-way-point
   :initform '(0 0 0 0)
   :accessor target-way-point)
  (way-point-list
   :initform '((0 -50 160 0)(0 -100 100 0)(0 -100 -100 0)
               (0 50 -50 0)(0 100 100 0)(0 150 150 0))
   :accessor way-point-list)
  (sub-way-point-list
   :initform ()
   :accessor sub-way-point-list)))

(defmethod update-position ((navigator navigatorclass))
  (update-auv-position auv) ; causes real world position of auv to change
  (setf (last-position navigator) (list (first (posture auv))
                                        (second (posture auv))
                                        (third (posture auv))))
  (if (target-point-reached navigator) (update-sub-way-point navigator)))
```

```
(defmethod update-sub-way-point ((navigator navigatorclass))
 (with-slots(target-way-point sub-way-point-list) navigator
  (cond ((not (null sub-way-point-list))
       (setf sub-way-point-list (rest sub-way-point-list))
       (setf target-way-point (first sub-way-point-list))
       (update-helm-commands (auto-pilot auv)
        (list (calculate-target-track navigator)
            (commanded-speed (auto-pilot auv))
            (commanded-depth (auto-pilot auv))
            (commanded-dive-angle (auto-pilot auv)))))
      (t (update-way-point navigator)))))

(defmethod calculate-target-track ((navigator navigatorclass))
 (let* ((course (radian-true-course (last-position navigator)
                        (rest (target-way-point navigator)))))
   (list (second (target-way-point navigator))
       (third (target-way-point navigator))  course)))

(defmethod update-way-point ((navigator navigatorclass))
 (with-slots (last-position target-way-point sub-way-point-list
         way-point-list) navigator
  (cond ((not(null way-point-list))
       (setf sub-way-point-list (rest (plan-route
                            (replanner navigator) world
                            (append (list 0) last-position)
                            (first way-point-list))))
       (setf target-way-point (first sub-way-point-list))
       (setf way-point-list (rest way-point-list)))
      (t (setf target-way-point '(0 0 0 0))))
  (update-helm-commands (auto-pilot auv)
   (list (calculate-target-track navigator)
       (commanded-speed (auto-pilot auv))
       (commanded-depth (auto-pilot auv))
       (commanded-dive-angle (auto-pilot auv))))))

(defmethod get-recovery-point ((navigator navigatorclass))
 (with-slots (target-way-point way-point-list) navigator
  (cond ((not(null way-point-list))
     (setf target-way-point (last way-point-list))
      (setf way-point-list nil))
     (t (setf target-way-point '(0 0 0 0)))))
  (update-helm-commands (auto-pilot auv)
   (list (calculate-target-track navigator)
```

```
        (commanded-speed (auto-pilot auv))
        (commanded-depth (auto-pilot auv))
        (commanded-dive-angle (auto-pilot auv)))))

(defmethod target-point-reached ((navigator navigatorclass))
 (with-slots (target-way-point last-position) navigator
  (if (> 10 (distance last-position (rest target-way-point))) t)))

(defmethod get-gps-fix ((navigator navigatorclass))
 (let* ((gps-fix (get-gps-fix (gps auv)))
        (gps-fix-time (current-time auv-clock)))
       (correct-position (ins auv) gps-fix gps-fix-time)
       (setf (last-position navigator) gps-fix))
 (update-helm-commands (auto-pilot auv)
  (list (calculate-target-track navigator)
        (commanded-speed (auto-pilot auv))
        (commanded-depth (auto-pilot auv))
        (commanded-dive-angle (auto-pilot auv)))))
(defmethod check-setpoints-modes ((navigator navigatorclass)) t)

(setf *affirmative* '(1 t y))

;----------------------------NAV ORDERS----------------------------------
(defmethod order((navigator navigatorclass) order)
 (update-position navigator)
  (case order
    (Initialize_vehicle

      (initialize (imu auv))
      (initialize (depth-cell auv))
      (initialize (gps auv))
      (initialize (uhf-receiver auv))
      (initialize (auto-pilot auv)))

    (Transit_to_task_location   t)

    (Transit_to_recovery_point
     (get-recovery-point navigator) t)

    (Abort_mission
     (setf (power-status (imu auv)) 'off)
     (setf (power-status (depth-cell auv)) 'off)
     (setf (power-status (gps auv)) 'off)
```

```
    (setf (power-status (uhf-receiver auv)) 'off)
    (setf (power-status (auto-pilot auv)) 'off))

   (Get_next-waypoint
    (update-way-point navigator) t)

   (Get_GPS_fix
    (setf (gps-obtained-flag navigator) 'no)
    (get-gps-fix navigator)
    (setf (gps-obtained-flag navigator) 'yes))

   (Loiter_and_Start_Global_Replanner t)

   (Log_new_obstacle t)

   (Loiter_and_Start_Local_Replanner t)

   (t nil)))
;-----------------------------NAVIGATOR QUERIES-------------------------------
(defmethod query ((navigator navigatorclass) question)
 (update-position navigator)
  (case question
   (Is_recovery_point_obtainable
    (distance (last-position navigator) (target-way-point navigator)))

   (Initialization_complete
    (cond ((and (equalp (power-status (imu auv)) 'on)
                (equalp (power-status (depth-cell auv)) 'on)
                (equalp (power-status (gps auv)) 'on)
                (equalp (power-status (uhf-receiver auv)) 'on)
                (equalp (power-status (auto-pilot auv)) 'on))
           (write-line "NAV -->OOD: yes" ) t)
          (t (write-line "NAV -->OOD: no") nil)))

   (Initialization_aborted
     (initialize (imu auv))
     (initialize (depth-cell auv))
     (initialize (gps auv))
     (initialize (uhf-receiver auv))
     (initialize (auto-pilot auv))
     (write-line "NAV -->OOD: no") nil)
```

```
(Task_location_reached
 (cond ((target-point-reached navigator)
       (write-line "NAV -->OOD: yes") t)
      (t (write-line "NAV -->OOD: no") nil)))

(At_recovery_point
 (cond ((target-point-reached navigator)
       (write-line "NAV -->OOD: yes") t)
      (t (write-line "NAV -->OOD: no") nil)))

(Waypoint_reached
 (cond ((target-point-reached navigator)
       (write-line "NAV -->OOD: yes") t)
      (t (write-line "NAV -->OOD: no") nil)))

(Waypoint_process_OK
 (check-setpoints-modes navigator)
 (write-line "NAV -->OOD: yes" ) t)

(Got_next_waypoint
 (cond ((and (equalp (first (target-way-point navigator)) 'R)
            (null (way-point-list navigator)))
    (write-line "NAV -->OOD: no") nil)
    (t (write-line "NAV -->OOD: yes") t)))

(GPS_fix_needed
 (cond((< 60 (- (current-time auv-clock) (last-gps-time navigator)))
     (write-line "NAV -->OOD: yes" ) t)
     (t(write-line "NAV -->OOD: no") nil)))

(GPS_fix_obtained
 (cond ((equalp (gps-obtained-flag) 'yes)
       (write-line "NAV -->OOD12: yes" ) t)
       (t(write-line "NAV -->OOD13: no") nil)))

(Abort_GPS_fix
 (cond ((and (equalp (power-status gps auv) 'on)
            (equalp (power-status uhf-receiver auv) 'on))
       (write-line "NAV -->OOD: no") nil)
       (t(write-line "NAV -->OOD: yes" ) t)))

(Area_clear_of_uncharted_obstacles
 (write-line "NAV -->OOD: yes" ) t)
```

```
(New_obstacle_logged
  (write-line "NAV -->OOD: yes" ) t)

(Log_system_failure
  (write-line "NAV -->OOD: no") nil)

(Setpoints_and_modes_sent
  (cond ((check-setpoints-modes navigator)
       (write-line "NAV -->OOD: yes" ) t)
      (t(write-line "NAV -->OOD: no") nil)))

(Setpoints_and_modes_system_OK
  (cond ((check-setpoints-modes navigator)
       (write-line "NAV -->OOD: yes" ) t)
      (t(write-line "NAV -->OOD: no") nil)))

(t nil) ))
```

C. SUB.CL

```
(defclass sub-class (rigid-body)
 ((gps
  :initform (make-instance 'gps-class)
  :accessor gps)
 (uhf-receiver
  :initform (make-instance 'uhf-receiver-class)
  :accessor uhf-receiver)
 (imu
  :initform (make-instance 'imu-class)
  :accessor imu)
 (depth-cell
  :initform (make-instance 'depth-cell-class)
  :accessor depth-cell)
 (speed-sensor
  :initform (make-instance 'speed-sensor-class)
  :accessor speed-sensor)
 (compass
  :initform (make-instance 'compass-class)
  :accessor compass)
 (auto-pilot
  :initform (make-instance 'auto-pilot-class)
  :accessor auto-pilot)
 (ins
```

```
 :initform (make-instance 'insprocessclass)
 :accessor ins)
(camera
 :initform (make-instance 'strobe-camera)
 :accessor camera)
(sigma
 :initform 2
 :accessor sigma)
(kappa
 :initform 0
 :accessor kappa)
(a
 :accessor a)
(b
 :accessor b)
(c
 :accessor c)
(node-list
 :initform '((0 0 0 1)   ;0
          (-5 -2 -1 1) ;1
          (-5 2 -1 1)  ;2
          (-5 2 1 1)   ;3
          (-5 -2 1 1)  ;4
          (-4 0 1 1)   ;5
          (-2 0 1 1)   ;6
          (-4 0 3 1)   ;7
          (-4 0 -3 1)  ;8
          (-2 0 -1 1)  ;9
          (-4 0 -1 1)  ;10
          (5 2 -1 1)   ;11
          (5 2 1 1)    ;12
          (10 0 0 1)   ;13
          (10 1 0 1)   ;14
          (10 -1 0 1)  ;15
          (5 -2 -1 1)  ;16
          (5 -2 1 1)   ;17
          (1 -4 0 1)   ;18
          (3 -2 0 1)   ;19
          (1 -2 0 1)   ;20
          (1 2 0 1)    ;21
          (3 2 0 1)    ;22
          (1 4 0 1)))  ;23
```

```
 (polygon-list
  :initform '((1 2 3 4)   ;backview
              (2 11 12 3) ;rt side
              (8 9 10)    ;top fin
              (5 6 7)     ;bottom fin
              (11 12 14)  ;rt cone
              (11 14 15 16);top cone
              (2 11 16 1) ;top
              (18 19 20)  ;lt fin
              (21 22 23)  ;rt fin
              (1 4 17 16) ;lt side
              (15 16 17)  ;lt cone
              (4 3 12 17) ;bottom
              (12 14 15 17)))));bottom cone

(defmethod initialize ((auv sub-class))
  (move (camera auv) 0 (- (/ pi 2)) 0 -100 -100 -100)
  (set-sigma auv)
  (setf (transformed-node-list auv) (node-list auv))
  (setf (velocity-growth-rate auv) (update-velocity-growth-rate auv))
  (setf (posture-rate auv) (earth-velocity auv))
  (take-picture (camera auv) auv)
  (setf (time-stamp auv) (current-time auv-clock)))

;set linearization constants
(defmethod set-sigma ((auv sub-class))
  (let* (( curvature (/ 1 (sigma auv))))
    (setf (a auv) (* 3 curvature))
    (setf (b auv) (* 3 (power 2 curvature)))
    (setf (c auv) (power 3 curvature))))

(defmethod update-rigid-body ((auv sub-class))    ;Euler integration.
  (let* ((delta-t (get-delta-t auv)))
      (update-posture auv delta-t)
      (setf (H-matrix auv) (homogeneous-transform (sixth (posture auv))
        (fifth (posture auv)) (fourth (posture auv)) (first (posture auv))
        (second (posture auv)) (third (posture auv))))
      (transform-node-list auv)
      (update-velocity auv delta-t)
      (update-velocity-growth-rate-1 auv delta-t)))
```

```
(defmethod update-velocity-growth-rate-1 ((auv sub-class) delta-t)
 (setf (velocity-growth-rate auv)
  (list (update-linear-velocity auv delta-t) 0 0
      (update-p auv) (update-q auv)(update-r auv delta-t))))

(defmethod update-r ((auv sub-class) delta-t)
 (let* ((A (dk-ds auv)))
  (setf (kappa auv) (+ (kappa auv) (* A (first (velocity auv)) delta-t)))
    (* A (square (first (velocity auv))) delta-t)))

(defmethod dk-ds ((auv sub-class))
 (with-slots (kappa a b c) auv
  (neg (+ (* a  kappa)
        (* b (fee (- (angle-trans (sixth (posture auv)))
                  (angle-trans (third (commanded-track (auto-pilot auv)))))))
        (* c (delta-d auv))))))

(defmethod delta-d ((auv sub-class))
 (let* ((x (first (posture auv)))
      (y (second (posture auv)))
      (xg (first (commanded-track (auto-pilot auv))))
      (yg (second (commanded-track (auto-pilot auv))))
      (theta-desired (fee (third (commanded-track (auto-pilot auv))))))
  (+ (neg (* (- x xg) (sin theta-desired))) (* (- y yg) (cos theta-desired)))))

(defmethod update-q ((auv sub-class))
 (let*
   ((depth (third (posture auv)))
    (depth-commanded (slot-value (auto-pilot auv) 'commanded-depth))
    (k-theta (/ 1 50))
    (theta-commanded (* (- depth-commanded depth) k-theta))
    (theta (fifth (posture auv)))
    (k-q (/ 1 10)))
 (* (- theta-commanded theta) k-q)))

(defmethod update-p ((auv sub-class))
 (* (fourth (posture auv)) (/ 1 2)))
```

```
(defmethod update-linear-velocity ((auv sub-class) delta-t)
 (let*
  ((u (first (velocity auv)))
   (u-commanded (slot-value (auto-pilot auv) 'commanded-speed))
   (k-r (/ 1 1)))
 (* (* (- u-commanded u) k-r) delta-t)))

(defmethod update-auv-position ((auv sub-class))
 (dotimes (i (* (timetick auv-clock) 10) (take-picture (camera auv) auv))
        (update-rigid-body auv)
        (update-ins (ins auv))))

;*********************************************************
(defclass blackbox (rigid-body)
 ((power-status
   :initform 'off
   :accessor power-status))
)

(defmethod initialize ((box blackbox))
 (setf (power-status box) 'on))

;*********************************************************
(defclass gps-class (blackbox)
 ((satellites-in-view
   :initform nil
   :accessor satellites-in-view)
  (differential-correction
   :initform '(0 0 0)
   :accessor differential-correction)
  (updated-position
   :initform '(1 2 3)
   :accessor updated-position)
  (position-fix-obtained
   :initform 'no
   :accessor position-fix-obtained))
)

(defmethod get-gps-fix ((gps gps-class))
 (put-satellites-in-view gps)
 (setf (updated-position gps)
     (apply-differential-correction gps (firstn 3 (posture auv)))))
```

```
(defmethod apply-differential-correction ((gps gps-class) gps-position)
 (with-slots (differential-correction position-fix-obtained) gps
  (setf differential-correction (get-differential-correction (uhf-receiver auv)))
  (setf position-fix-obtained 'yes)
  (vector-add gps-position differential-correction)))

(defmethod put-satellites-in-view ((gps gps-class))
 (with-slots (satellites-in-view) gps
  (setf satellites-in-view nil)
  (do*
   ((generator  0              (random 10000))
    (in-view    satellites-in-view (if (> generator 9000) (cons 1 in-view) in-view)))
  ((> (length in-view) 2) t))))

;*********************************************************
(defclass uhf-receiver-class (blackbox)

 ((differential-correction
  :initform '(0 0 0)
  :accessor differential-correction))
)

(defmethod get-differential-correction ((uhf-receiver uhf-receiver-class))
 (update-differential-correction uhf-receiver)
 (differential-correction uhf-receiver))

(defmethod update-differential-correction ((uhf-receiver uhf-receiver-class))
 t)

;*********************************************************
(defclass imu-class (blackbox) ())

(defmethod update-imu ((imu imu-class))
 (let*
  ((phi (fourth (posture auv)))
   (theta (fifth (posture auv)))
   (u (first (velocity auv)))
   (v (second (velocity auv)))
   (w (third (velocity auv)))
   (p (fourth (velocity auv)))
   (q (fifth (velocity auv)))
   (r (sixth (velocity auv)))
   (u-dot (first (velocity-growth-rate auv)))
```

```
    (v-dot (second (velocity-growth-rate auv)))
    (w-dot (third (velocity-growth-rate auv))))
  (list
    (+ u-dot (+ (* -1 v r) (+ (* w q) (* *gravity* (sin theta)))))
    (+ v-dot (+ (* -1 w p) (- (* u r) (* *gravity* (cos theta) (sin phi)))))
    (+ w-dot (+ (* -1 u q) (- (* v p) (* *gravity* (cos theta) (cos phi)))))
    (fourth (velocity-growth-rate auv))
    (fifth (velocity-growth-rate auv))
    (sixth (velocity-growth-rate auv)))))

(defmethod initialize ((imu imu-class))
  (setf (power-status imu) 'on))
;  (setf (last-access-time imu) (current-time auv-clock)))

;*********************************************************
(defclass depth-cell-class (blackbox)

  ((pressure
    :initform 0  ;pounds/sqr inch
    :accessor pressure))
)

(defmethod get-depth ((depth-cell depth-cell-class))
  (setf depth (* (pressure depth-cell) (/ 32.0 14.7))))

;*********************************************************
(defclass speed-sensor-class (blackbox)())

(defmethod get-speed ((speed-sensor speed-sensor-class))
  (first (velocity auv)))


;*********************************************************
(defclass compass-class (blackbox)())

(defmethod get-heading ((compass compass-class))
  (sixth (posture auv)))

;*********************************************************
(defclass auto-pilot-class (blackbox)
  ((commanded-track
    :initform '(0 0 0)
    :accessor commanded-track)
```

```
  (commanded-speed
   :initform 0
   :accessor commanded-speed)
  (commanded-depth
   :initform 0
   :accessor commanded-depth)
  (commanded-dive-angle
   :initform 0
   :accessor commanded-dive-angle)))

(defmethod update-helm-commands ((auto-pilot auto-pilot-class) commands)
  (setf (commanded-track auto-pilot) (first commands))
  (setf (commanded-speed auto-pilot) (second commands))
  (setf (commanded-depth auto-pilot) (third commands))
  (setf (commanded-dive-angle auto-pilot) (fourth commands)))

;(defmethod auto-pilot-position ((auto-pilot auto-pilot-class))
;  (firstn 3 (posture auv)))

;*********************************************************
```

D. INS.CL

```
;----------------------INSprocess CLASS AND METHODS-------------------------

(defclass insprocessclass ()
  ((estimated-posture
    :initform '(0 0 0) ;xe ye ze
    :accessor estimated-posture)
   (estimated-attitude
    :initform '(0 0 0) ;phi theta psi
    :accessor estimated-attitude)
   (estimated-linear-velocity
    :initform '(0 0 0) ;u v w
    :accessor estimated-linear-velocity)
   (estimated-angular-rates
    :initform '(0 0 0) ;p q r
    :accessor estimated-angular-rates)
   (last-gps-time
    :initform 0
    :accessor last-gps-time)
   (error-vector
    :initform '(0 0 0)
```

```
   :accessor error-vector)
  (k-one
   :initform '(0.0 0.0)
   :accessor k-one)
  (k-two
   :initform 0.0
   :accessor k-two)
  (k-three
   :initform '((0.0 0.0 0.0)
               (0.0 0.0 0.0)
               (0.0 0.0 0.0))
   :accessor k-three)
  (k-four
   :initform '(1.0 1.0 1.0)
   :accessor k-four)))
(defmethod ins-position ((ins insprocessclass))
  (firstn 3 (estimated-posture ins)))

(defmethod correct-position((ins insprocessclass) position time)
  (with-slots (estimated-posture error-vector last-gps-time k-four) ins
    (let* ((delta-x (- (first position) (first estimated-posture)))
           (delta-y (- (second position) (second estimated-posture)))
           (delta-z (- (third position) (third estimated-posture)))
           (delta-t (- time last-gps-time)))
          (setf estimated-posture position)
       . (setf error-vector (vector-add error-vector (list
           (* (first k-four) (/ delta-x delta-t))
           (* (second k-four) (/ delta-y delta-t))
           (* (third k-four) (/ delta-z delta-t))))))
    (setf last-gps-time time)))

(defmethod update-ins ((ins insprocessclass))
  (with-slots (estimated-attitude estimated-angular-rates estimated-posture
               estimated-linear-velocity) ins
  (let* ((imu-output (update-imu (imu auv)))
         (speed (get-speed (speed-sensor auv)))
         (heading (get-heading (compass auv))))
        (setf estimated-angular-rates (rate-filter ins imu-output heading))
        (setf estimated-attitude (integrate-euler-rates ins))
        (setf estimated-linear-velocity (estimated-earth-velocity
                                  ins (firstn 3 imu-output) speed))
        (setf estimated-posture (integrate-velocities ins)))))
```

```
(defmethod rate-filter ((ins insprocessclass)imu-output heading)
 (with-slots (estimated-attitude k-one k-two) ins
  (let* ((theta-a (asin (/ (first imu-output) *gravity*)))
       (phi-a (neg (asin (/ (second imu-output)
                    (* *gravity* (cos (second estimated-attitude)))))))
       (phi-filter (* (first k-one) (- phi-a (first estimated-attitude))))
       (theta-filter (* (second k-one) (- theta-a (second estimated-attitude))))
       (psi-filter (* k-two (- heading (third estimated-attitude))))
       (filter-correction (list phi-filter theta-filter psi-filter))
       (unfiltered-euler-rates (post-multiply (body-rate-to-euler-rate-matrix
                                  (third estimated-attitude)
                                  (second estimated-attitude)
                                  (first estimated-attitude))
                                 (list
                                  (fourth imu-output)
                                  (fifth imu-output)
                                  (sixth imu-output)))))
      (vector-add unfiltered-euler-rates filter-correction))))

(defmethod integrate-euler-rates ((ins insprocessclass))
 (with-slots (estimated-attitude estimated-angular-rates) ins
  (vector-add estimated-attitude
        (scalar-multiply (get-delta-t auv) estimated-angular-rates))))

(defmethod estimated-earth-velocity ((ins insprocessclass) specific-forces speed)
 (with-slots (estimated-attitude estimated-linear-velocity error-vector k-three) ins
  (let* ((x-linear-accel (- (first specific-forces)
                   (* *gravity*
                     (sin (second estimated-attitude)))))
       (y-linear-accel (+ (second specific-forces)
                   (* *gravity*
                     (sin (first estimated-attitude))
                     (cos (second estimated-attitude)))))
       (z-linear-accel (+ (third specific-forces)
                   (* *gravity*
                     (cos (first estimated-attitude))
                     (cos (second estimated-attitude)))))
       (r-matrix (rotation-matrix (third estimated-attitude)
                        (second estimated-attitude)
                        (first estimated-attitude)))
       (earth-accels (post-multiply r-matrix
                          (list x-linear-accel
                              y-linear-accel
```

```
                        z-linear-accel)))
      (speed-vector (post-multiply r-matrix (list speed 0 0)))
      (filter-correction (post-multiply k-three
                  (vector-add (scalar-multiply -1 estimated-linear-velocity)
                        (vector-add speed-vector error-vector)))))
     (vector-add estimated-linear-velocity
          (scalar-multiply (get-delta-t auv)
                (vector-add filter-correction
                      earth-accels))))))

(defmethod integrate-velocities ((ins insprocessclass))
 (with-slots (estimated-posture estimated-linear-velocity) ins
  (vector-add estimated-posture
       (scalar-multiply (get-delta-t auv) estimated-linear-velocity))))

;----------------------REPLANNER CLASS AND METHODS-------------------------

;(load 'replanner.cl)

(load 'world.cl)

(defclass replannerclass()() )


(defmethod plan-route ((replanner replannerclass) obstacles beginning end)
 (list beginning end))
```

E. ROBOT-KINEMATICS.CL

```
File: robot-kinematics.cl              Franz Common LISP
;
; by Dr. McGhee for  CS4314
; ***************************************************************************

(defun transpose (matrix)        ;A matrix is a list of row vectors.
 (cond ((null (cdr matrix)) (mapcar 'list (car matrix)))
       (t (mapcar 'cons (car matrix) (transpose (cdr matrix))))))

(defun dot-product (vector-1 vector-2)  ;A vector is a list of numerical atoms.
 (apply '+ (mapcar '* vector-1 vector-2)))

(defun vector-magnitude (vector) (sqrt (dot-product vector vector)))
```

```
(defun post-multiply (matrix vector)
 (cond ((null (rest matrix)) (list (dot-product (first matrix) vector)))
       (t (cons (dot-product (first matrix) vector)
          (post-multiply (rest matrix) vector)))))

(defun pre-multiply (vector matrix)
 (post-multiply (transpose matrix) vector))

(defun matrix-multiply (A B)    ;A and B are conformable matrices.
 (cond ((null (cdr A)) (list (pre-multiply (car A) B)))
       (t (cons (pre-multiply (car A) B) (matrix-multiply (cdr A) B)))))

(defun chain-multiply (L)       ;L is a list of names of conformable matrices.
 (cond ((null (cddr L)) (matrix-multiply (eval (car L)) (eval (cadr L))))
       (t (matrix-multiply (eval (car L)) (chain-multiply (cdr L))))))

(defun cycle-left (matrix) (mapcar 'row-cycle-left matrix))

(defun row-cycle-left (row) (append (cdr row) (list (car row))))

(defun cycle-up (matrix) (append (cdr matrix) (list (car matrix))))

(defun unit-vector (one-column length)       ;Column count starts at 1.
 (do ((n length (1- n))
    (vector nil (cons (cond ((= one-column n) 1) (t 0)) vector)))
   ((zerop n) vector)))

(defun unit-matrix (size)
 (do ((row-number size (1- row-number))
    (I nil (cons (unit-vector row-number size) I)))
   ((zerop row-number) I)))

(defun concat-matrix (A B)  ;A and B are matrices with equal number of rows.
 (cond ((null A) B)
       (t (cons (append (car A) (car B)) (concat-matrix (cdr A) (cdr B))))))

(defun augment (matrix)
 (concat-matrix matrix (unit-matrix (length matrix))))

(defun normalize-row (row) (scalar-multiply (/ 1.0 (car row)) row))

(defun scalar-multiply (scalar vector)
 (cond ((null vector) nil)
```

```
        (t (cons (* scalar (car vector))
                 (scalar-multiply scalar (cdr vector))))))

(defun solve-first-column (matrix)      ;Reduces first column to (1 0 ... 0).
 (do* ((remaining-row-list matrix (rest remaining-row-list))
       (first-row (normalize-row (first matrix)))
       (answer (list first-row)
               (cons (vector-add (first remaining-row-list)
                                 (scalar-multiply (- (caar remaining-row-list))
                                                  first-row))
                     answer)))
      ((null (rest remaining-row-list)) (reverse answer))))
(defun vector-add (vector-1 vector-2) (mapcar '+ vector-1 vector-2))

(defun vector-subtract (vector-1 vector-2) (mapcar '- vector-1 vector-2))

(defun first-square (matrix)  ;Returns leftmost square matrix from argument.
 (do ((size (length matrix))
      (remainder matrix (rest remainder))
      (answer nil (cons (firstn size (first remainder)) answer)))
     ((null remainder) (reverse answer))))

(defun firstn (n list)
 (cond ((zerop n) nil)
       (t (cons (first list) (firstn (1- n) (rest list))))))

(defun max-car-firstn (n list)
 (append (max-car-first (firstn n list)) (nthcdr n list)))

(defun matrix-inverse (M)
 (do ((M1 (max-car-first (augment M))
          (cond ((null M1) nil)      ;Abort for singular matrix.
                (t (max-car-firstn n (cycle-left (cycle-up M1))))))
      (n (1- (length M)) (1- n)))
     ((or (minusp n) (null M1)) (cond ((null M1) nil) (t (first-square M1))))
     (setq M1 (cond ((zerop (caar M1)) nil) (t (solve-first-column M1))))))

(defun max-car-first (L)  ;L is a list of lists. This function finds list with
 (cond ((null (cdr L)) L) ;largest car and moves it to head of list of lists.
       (t (if (> (abs (caar L)) (abs (caar (max-car-first (cdr L))))) L
              (append (max-car-first (cdr L)) (list (car L)))))))
```

```
(defun dh-matrix (cosrotate sinrotate costwist sintwist length translate)
 (list (list cosrotate (- (* costwist sinrotate))
             (* sintwist sinrotate) (* length cosrotate))
       (list sinrotate (* costwist cosrotate)
           (- (* sintwist cosrotate)) (* length sinrotate))
       (list 0. sintwist costwist translate) (list 0. 0. 0. 1.)))

(defun homogeneous-transform (azimuth elevation roll x y z)
 (let ((spsi (sin azimuth)) (cpsi (cos azimuth)) (sth (sin elevation))
     (cth (cos elevation)) (sphi (sin roll)) (cphi (cos roll)))
    (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
              (+ (* cpsi sth cphi) (* spsi sphi)) x)
          (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
              (- (* spsi sth cphi) (* cpsi sphi)) y)
       (list (- sth) (* cth sphi) (* cth cphi) z)
       (list 0. 0. 0. 1.))))

(defun inverse-H (H)        ;H is a 4x4 homogeneous transformation matrix.
 (let* ((minus-P (list (- (fourth (first H)))
                       (- (fourth (second H)))
                       (- (fourth (third H)))))
      (inverse-R (transpose (first-square (reverse (rest (reverse H))))))
      (inverse-P (post-multiply inverse-R minus-P)))
     (append (concat-matrix inverse-R (transpose (list inverse-P)))
          (list (list 0 0 0 1)))))

(defun rotation-matrix (azimuth elevation roll)
 (let ((spsi (sin azimuth)) (cpsi (cos azimuth)) (sth (sin elevation))
     (cth (cos elevation)) (sphi (sin roll)) (cphi (cos roll)))
    (list (list (* cpsi cth) (- (* cpsi sth sphi) (* spsi cphi))
              (+ (* cpsi sth cphi) (* spsi sphi)))
          (list (* spsi cth) (+ (* cpsi cphi) (* spsi sth sphi))
              (- (* spsi sth cphi) (* cpsi sphi)))
       (list (- sth) (* cth sphi) (* cth cphi)))))

(defun body-rate-to-euler-rate-matrix (azimuth elevation roll)
 (let ((sth (sin elevation)) (cth (cos elevation)) (tth (tan elevation))
     (sphi (sin roll)) (cphi (cos roll)))
    (list (list 1 (* tth sphi) (* tth cphi))
       (list 0 cphi (- sphi))
       (list 0 (/ sphi cth) (/ cphi cth)))))
```

```
(setf T '((1 2) (3 4)))
(setf M '((1 2 3) (4 5 6) (7 8 9)))
(setf x '(1 2 3))
```

F. EULER-ANGLE-RIGID-BODY.CL

```
File: euler-angle-rigid-body.cl          Franz Common LISP
;
; by Dr. McGhee for  CS4314
; ***************************************************************

(defclass rigid-body
   ()
  ((posture             ;The vector (xe ye ze phi theta psi).
   :initform '(0 0 0 0 0 3.14)
   :initarg :posture
   :accessor posture)
  (posture-rate ;The vector (xe-dot ye-dot ze-dot phi-dot theta-dot psi-dot).
   :initarg :posture-rate
   :accessor posture-rate)
  (velocity             ;The six-vector (u v w p q r) in body coordinates.
   :initform '(0 0 0 0 0 0)
   :initarg :velocity
   :accessor velocity)
  (velocity-growth-rate  ;The vector (u-dot v-dot w-dot p-dot q-dot r-dot).
   :accessor velocity-growth-rate)
  (forces-and-torques    ;The vector (Fx Fy Fz L M N) in body coordinates.
   :initform (list 0 0 (- *gravity*) 0 0 0)
   :accessor forces-and-torques)
  (moments-of-inertia    ;The vector (Ix Iy Iz) in principal axis coordinates.
   :initform '(1 1 1)
   :initarg :moments-of-inertia
   :accessor moments-of-inertia)
  (mass
   :initform 1
   :initarg :mass
   :accessor mass)
  (node-list    ;(x y z 1) in body coord for each node. Starts with (0 0 0 1).
   :initform  '((0 0 0 1) (4 0 0 1) (2 0 0 1) (-4 0 0 1) (-5 0 -2 1)
              (-6 -1.5 -2 1) (-6 1.5 -2 1) (-2 6 -2 1) (-2 -6 -2 1)
              (-2 0 0 1)) ;Defines a simple "airplane" as default rigid body.
   :initarg :node-list
   :accessor node-list)
```

```
  (polygon-list
   :initform '((1 3 4 5 4 3) (4 6) (7 2 8 9))
   :initarg :polygon-list
   :accessor polygon-list)
  (transformed-node-list  ;(x y z 1) in earth coord for each node in node-list.
   :accessor transformed-node-list)
  (H-matrix
   :initform (unit-matrix 4)
   :accessor H-matrix)
  (time-stamp
   :accessor time-stamp)))

(defmethod initialize ((body rigid-body))
 (setf (transformed-node-list body) (node-list body))
 (setf (velocity-growth-rate body) (update-velocity-growth-rate body))
 (setf (posture-rate body) (earth-velocity body))
 (setf (time-stamp body) (get-internal-real-time)))

(defmethod move-body ((body rigid-body) azimuth elevation roll x y z)
 (setf (posture body) (list x y z roll elevation azimuth))
 (setf (H-matrix body)
  (homogeneous-transform azimuth elevation roll x y z))
 (transform-node-list body))

(defmethod get-delta-t ((body rigid-body)) 0.1)

(defmethod update-rigid-body ((body rigid-body))    ;Euler integration.
 (let* ((delta-t (get-delta-t body)))
    (update-posture body delta-t)
    (setf (H-matrix body) (homogeneous-transform (sixth (posture body))
      (fifth (posture body)) (fourth (posture body)) (first (posture body))
      (second (posture body)) (third (posture body))))
    (transform-node-list body)
    (update-velocity body delta-t)
    (update-velocity-growth-rate body)))

(defmethod update-velocity-growth-rate ((body rigid-body))
 (setf (velocity-growth-rate body) ;Assumes principal axis coordinates with
  (multiple-value-bind          ;origin at center of gravity of body.
   (Fx Fy Fz L M N  u v w p q r  Ix Iy Iz) ;Declares local variables.
   (values-list                 ;Values assigned.
    (append
     (forces-and-torques body) (velocity body) (moments-of-inertia body)))
```

```
    (list (+ (* v r) (* -1 w q) (/ Fx (mass body))
             (* *gravity* (first (third (H-matrix body)))))
          (+ (* w p) (* -1 u r) (/ Fy (mass body))
             (* *gravity* (second (third (H-matrix body)))))
          (+ (* u q) (* -1 v p) (/ Fz (mass body))
             (* *gravity* (third (third (H-matrix body)))))
          (/ (+ (* (- Iy Iz) q r) L) Ix)
          (/ (+ (* (- Iz Ix) r p) M) Iy)
          (/ (+ (* (- Ix Iy) p q) N) Iz)))))  ;Value returned.

(defmethod update-velocity ((body rigid-body) delta-t) ;Euler integration.
  (setf (velocity body)
    (vector-add (velocity body)
                (scalar-multiply delta-t (velocity-growth-rate body)))))

(defmethod update-posture ((body rigid-body) delta-t) ;Euler integration.
  (setf (posture-rate body) (earth-velocity body))
  (setf (posture body)
    (vector-add (posture body) (scalar-multiply delta-t (posture-rate body)))))

(defmethod transform-node-list ((body rigid-body))
  (setf (transformed-node-list body)
    (mapcar #'(lambda (node-location)
                (post-multiply (H-matrix body) node-location))
            (node-list body))))

(defconstant *gravity* 32.2185)

(defmethod earth-velocity ((body rigid-body))
  (let* ((translational-velocity (list(first (velocity body))
                                      (second (velocity body))
                                      (third (velocity body))))
         (rotational-velocity (list(fourth (velocity body))
                                   (fifth (velocity body))
                                   (sixth (velocity body)))))
    (append (post-multiply (rotation-matrix (sixth (posture body))
                                            (fifth (posture body))
                                            (fourth (posture body)))
                           translational-velocity)
            (post-multiply (body-rate-to-euler-rate-matrix (sixth (posture body))
                                                           (fifth (posture body))
                                                           (fourth (posture body)))
                           rotational-velocity))))
```

```
(defun test-rigid-body ()
 (setf airplane-1 (make-instance 'rigid-body))
 (initialize airplane-1)
 (setf camera-1 (make-instance 'strobe-camera))
 (move camera-1 0 (- (/ pi 2)) 0 0 0 -30)
 (take-picture camera-1 airplane-1)
 (dotimes (i 20 'done) (update-rigid-body airplane-1))
 (take-picture camera-1 airplane-1))
```

G. WIND-TRI.CL

```
(defconstant two-pi 6.283185307179586)  ;pi multiplied by two

(defconstant pi-two 1.5707963267948966) ;pi divided by two

;;**************************clock class **********************************

(defclass clock-class ()

 ((time-count
   :initform 0
   :accessor time-count)

  (timetick
   :initform 1
   :accessor timetick)))

(defmethod current-time ((clock clock-class))
 (time-count clock))

(defmethod increment-time ((clock clock-class))
 (with-slots (time-count timetick) clock
 (setf time-count (+ time-count timetick))))

;;*************************************************************************

(defun radian-true-course (from-point to-point)
 (positive-radians (atan (- (second to-point) (second from-point))
                         (- (first to-point) (first from-point)))))

(defun positive-radians (radians)
 (if (< radians 0) (+ radians two-pi)  radians));
```

```
(defun calculate-heading (speed course current)
 (cond ((zerop speed) course)
       (t (let* ((A (-  course (second current)))
                 (b (first current))
                 (a speed)
                 (B (asin (/ (* b (sin A)) a))))
                (+ course B)))))

(defun square (x) (* x x))

(defun deg-to-rad (deg) (* deg (/ pi 180)))

(defun rad-to-deg (rad) (* rad (/ 180 pi)))

(defun distance (from-point to-point)
 (let*((x-delta (- (first to-point)
                   (first from-point)))
       (y-delta (- (second to-point)
                   (second from-point))))
      (sqrt (+ (square x-delta) (square y-delta)))))

(defun find-vector (from-point to-point)
 (list (distance from-point to-point)
       (radian-true-course from-point to-point)))

(defun firstj (n list) ;improved version
 (cond ((or(zerop n)(null list)) nil)
       (t (cons (first list) (firstj (1- n) (rest list))))))

(defun power (exponent base)
 (do* ((i exponent (1- i))
       (result base (* result base)))
      ((= i 1) result)))

(defun neg (number)
 (* -1 number))

(defun angle-trans (heading)
 (if (> heading pi) (- heading two-pi) heading))

;normalizes alpha to between pi and -pi
(defun fee (alpha)
 (if (> alpha pi) (fee(- alpha two-pi))
```

```
  (if (<= alpha (neg pi)) (fee(+ alpha two-pi)) alpha)))
```

H. CAMERA.CL

```
; File: camera.cl                    Franz Common LISP
;
; ** CAMERA CLASS DEFINITION **
; A Camera  "takes a picture" of rigid-body class objects
; and displays the image.  A sequence of images may be
; displayed by superimposing them or by first erasing the display
; window and then creating and displaying the next image.
;
; Requires: rigid-body.cl
;
; by Shirley Isakari  CS4314 Winter 1994  Final Project
; Modifications & enhancements to Prof. McGhee's Strobe-Camera CLOS code
; ***********************************************************************

(require :xcw)

(use-package :cw)  ; Note that this is required for use of mouse and color.
                   ; This forced renaming of some original functions, i.e.
                   ; move and translate. Causes some problem when compiling.

(cw:initialize-common-windows)

(defclass camera (rigid-body)
  ((focal-length
    :accessor focal-length
    :initform 6)
   (posture
    :accessor posture   ; azim elev roll x y z
    :initform (list 0 0 0 -300 0 0))
   (camera-window
    :accessor camera-window
    :initform (cw:make-window-stream :borders 5
                                     :left 300
                                          :bottom 300
                                             :width 300
                                             :height 400
                                     :title "Right Arm Articulation"
                                     :background-color blue
                                     :foreground-color white
```

```
                                :activate-p t))
  (H-matrix
   :initform (homogeneous-transform 0 0 0 -300 0 0))
  (inverse-H-matrix
   :accessor inverse-H-matrix
   :initform (inverse-H (homogeneous-transform 0 0 0 -300 0 0)))
  (enlargement-factor
   :accessor enlargement-factor
   :initform 900)))

(defun create-camera-1 ()
  (setf camera-1 (make-instance 'camera))
  (queue-mouse camera-1))

(defmethod queue-mouse ((camera camera))
  (cw:modify-window-stream-method (camera-window camera) :left-button-down
     :after 'mouse-handler)
  (cw:modify-window-stream-method (camera-window camera) :middle-button-down
     :after 'mouse-handler)
  (cw:modify-window-stream-method (camera-window camera) :right-button-down
     :after 'mouse-handler))

; Note that mouse-handler requires names of instantiated objects:
; camera-1 jack-1.  Unable to modify argument list of this event-handler.
(defun mouse-handler (wstream cw:mouse-state &optional event)
  ;(format t "In mouse-handler button: ~a~%" (mouse-button-state))
  (cond ((eql (cw:mouse-button-state) 128)      ; Left-click
         (rotate-camera camera-1 -10)
          ;  (format t "Mouse Event: Left-click => rotate-camera~%")
        )
        ((eql (cw:mouse-button-state) 129)      ; Left-click & CNTRL key
          (rotate-camera camera-1 10)
          ;   (format t "Mouse Event: CNTRL+Left-click => rotate-camera~%")
        )
        ((eql (cw:mouse-button-state) 64)           ; Middle-click
          (zoom-camera camera-1 10)
       ;  (format t "Mouse Event: Middle-click => zoom-camera~%")
        )
        ((eql (cw:mouse-button-state) 65)       ; Middle-click & CNTRL key
          (zoom-camera camera-1 -10)
       ;  (format t "Mouse Event: CNTRL+Middle-click => zoom-camera~%")
        )
        ((eql (cw:mouse-button-state) 32)           ; Right-click
```

```
        (tilt-camera camera-1 -10)
      ;   (format t "Mouse Event: Right-click => tilt-camera~%")
       )
       ((eql (cw:mouse-button-state) 33)        ; Right-click & CNTRL key
        (tilt-camera camera-1 10)
      ;   (format t "Mouse Event: CNTRL+Right-click => tilt-camera~%")
       )
       (t nil))
  (new-picture camera-1 jack-1 jack-color))

; *** Defined global color constants ************************************
; To be used as the draw-color argument in take-picture and new-picture
; functionss (and also jack-picture, jack-video, jack-movie functions)

(defconstant *white* 0)
(defconstant *yellow* 1)
(defconstant *red* 2)
(defconstant *green* 3)
(defconstant *black* 4)
(defconstant *cyan* 5)
(defconstant *magenta* 6)
(defconstant *blue* 7)

; *** Draw picture functions ********************************************

(defmethod take-picture ((camera camera) (body rigid-body) draw-color)
  (let ((camera-space-node-list (mapcar #'(lambda (node-location)
       (post-multiply (inverse-H-matrix camera) node-location))
                (transformed-node-list body))))
    (dolist (polygon (polygon-list body))
      (clip-and-draw-polygon camera polygon camera-space-node-list draw-color))))

(defmethod erase-camera-window ((camera camera))
  (cw:clear (camera-window camera)))

(defmethod new-picture ((camera camera) (body rigid-body) draw-color)
   (erase-camera-window camera)
   (take-picture camera body draw-color))

(defmethod clip-and-draw-polygon
  ((camera camera) polygon node-coord-list draw-color)
  (do* ((initial-point (nth (first polygon) node-coord-list))
       (from-point initial-point to-point)
```

```
    (remaining-nodes (rest polygon) (rest remaining-nodes))
    (to-point (nth (first remaining-nodes) node-coord-list)
          (if (not (null (first remaining-nodes)))
            (nth (first remaining-nodes) node-coord-list))))
   ((null to-point)
      (draw-clipped-projection camera from-point initial-point draw-color))
   (draw-clipped-projection camera from-point to-point draw-color)))

(defmethod draw-clipped-projection ((camera camera)
       from-point to-point draw-color)
 (cond ((and (<= (first from-point) (focal-length camera))
         (<= (first to-point) (focal-length camera))) nil)
     ((<= (first from-point) (focal-length camera))
      (draw-line-in-window camera
       (perspective-transform camera (from-clip camera from-point to-point))
       (perspective-transform camera to-point) draw-color))
     ((<= (first to-point) (focal-length camera))
      (draw-line-in-window camera
       (perspective-transform camera from-point)
       (perspective-transform camera (to-clip camera from-point to-point))
          draw-color))
     (t (draw-line-in-window camera
       (perspective-transform camera from-point)
       (perspective-transform camera to-point) draw-color))))

(defmethod from-clip ((camera camera) from-point to-point)
 (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                (- (first to-point) (first from-point)))))
    (list (+ (first from-point)
          (* scale-factor (- (first to-point) (first from-point))))
        (+ (second from-point)
          (* scale-factor (- (second to-point) (second from-point))))
        (+ (third from-point)
          (* scale-factor (- (third to-point) (third from-point)))) 1)))

(defmethod to-clip ((camera camera) from-point to-point)
 (from-clip camera to-point from-point))

(defmethod draw-line-in-window ((camera camera) start end draw-color)
  (cond ((= 0 draw-color) (cw:draw-line (camera-window camera)
          (cw:make-position :x (first start) :y (second start))
          (cw:make-position :x (first end) :y (second end))
          :brush-width 5 :color white))
```

```
        ((= 1 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color yellow))
        ((= 2 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color magenta))
      ((= 3 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color green))
      ((= 4 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color red))
      ((= 5 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color cyan))
      ((= 6 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color black))
      ((= 7 draw-color) (cw:draw-line (camera-window camera)
           (cw:make-position :x (first start) :y (second start))
           (cw:make-position :x (first end) :y (second end))
           :brush-width 5 :color blue))))

(defmethod perspective-transform ((camera camera) point-in-camera-space)
  (let* ((enlargement-factor (enlargement-factor camera))
         (focal-length (focal-length camera))
         (x (first point-in-camera-space))  ;x axis is along optical axis
         (y (second point-in-camera-space)) ;y is out right side of camera
         (z (third point-in-camera-space))) ;z is out bottom of camera
        (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
                 150)                      ;to right in camera window
              (+ 150 (round (* enlargement-factor (/ (* focal-length (- z)) x))
                            ))))  ;up in camera window

; *** Position camera functions ****************************************

(defmethod move-camera ((camera camera) azimuth elevation roll x y z)
```

```
 (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
 (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera)))
 (format t "camera: ~a " (posture camera)) )

(defmethod zoom-camera ((camera camera) zoom-amount)
 (setf (slot-value camera 'enlargement-factor)
     (+ (slot-value camera 'enlargement-factor) zoom-amount)))


; Rotation in x-y plane about origin
(defmethod rotate-camera ((camera camera) angle-increment) ; in degrees
 (let* ((new-position (posture camera))
       (radius (sqrt (+ (* (fourth new-position) (fourth new-position))
                 (* (fifth new-position) (fifth new-position)))))
       (heading (atan (fourth new-position)
         (fifth new-position)))
       (angle (deg-to-rad angle-increment))
         (new-heading (+ heading angle)))
       (setf (first new-position) (- (first new-position) angle)
           (fourth new-position) (* radius (sin new-heading))
           (fifth new-position) (* radius (cos new-heading))
           (posture camera) new-position
           (H-matrix camera) (homogeneous-transform (first new-position)
           (second new-position) (third new-position) (fourth new-position)
           (fifth new-position) (sixth new-position))
           (inverse-H-matrix camera) (inverse-H (H-matrix camera)))))

; Vertical tilting about origin in a plane perpendicular to x-y plane
; Max tilt (90 or -90 deg) when top or bottom view of x-y plane is achieved
(defmethod tilt-camera ((camera camera) angle-increment) ; in degrees
 (let* ((new-position (posture camera))
       (radius (sqrt (+ (* (fourth new-position) (fourth new-position))
                   (* (fifth new-position) (fifth new-position))
                 (* (sixth new-position) (sixth new-position)))))
       (tilt (atan (sixth new-position)
           (sqrt (+  (* (fourth new-position) (fourth new-position))
                (* (fifth new-position) (fifth new-position))))))
       (heading (atan (fourth new-position)
           (fifth new-position)))
       (angle (deg-to-rad angle-increment))
         (new-tilt (cond ((< (abs (+ tilt angle)) tilt-limit) (+ tilt angle))
                   (t (cond ((minusp (+ tilt angle)) (* -1 tilt-limit))
                        (t tilt-limit))))))
```

```
      (setf (second new-position) new-tilt
       (fourth new-position)
            (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
                 (fourth new-position))
                 (t (* radius (sin heading) (cos new-tilt))))
       (fifth new-position)
         (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
                 (fifth new-position))
                 (t (* radius (cos heading) (cos new-tilt))))
       (sixth new-position)
         (cond ((= (abs tilt) (abs new-tilt) tilt-limit)
                 (sixth new-position))
                 (t (* radius (sin new-tilt))))
       (posture camera) new-position
       (H-matrix camera) (homogeneous-transform (first new-position)
       (second new-position) (third new-position) (fourth new-position)
       (fifth new-position) (sixth new-position))
       (inverse-H-matrix camera) (inverse-H (H-matrix camera)))))

(defun deg-to-rad (angle) (* .017453292519943295 angle))
(defconstant tilt-limit (deg-to-rad 89.9))

; *** Auxiliary functions ************************************************

(defun kill ()
  (cw:kill-common-windows))

(defun reset-windows ()
  (kill)
  (cw:initialize-common-windows))
```

I. STROBE-CAMERA.CL

```
File: strobe-camera.cl                Franz Common LISP
;
; ** STROBE-CAMERA CLASS DEFINITION **
; A Camera  "takes a picture" of rigid-body class objects
; and displays the image.  A sequence of images may be
; displayed by superimposing them.
; Requires: rigid-body.cl
;
; by Dr. McGhee for  CS4314
; ************************************************************************
```

```
(require :xcw)
(cw:initialize-common-windows)

(defclass strobe-camera (rigid-body)
 ((focal-length
  :accessor focal-length
  :initform 6)
  (camera-window
  :accessor camera-window
  :initform (cw:make-window-stream :borders 5
                        :left 500
                              :bottom 500
                                 :width 700
                                 :height 700
                                 :title "AUV"
                                 :activate-p t))
  (H-matrix
   :initform (homogeneous-transform 0 (/ pi 2) 0 0 0 150))
  (inverse-H-matrix
   :accessor inverse-H-matrix
   :initform (inverse-H (homogeneous-transform 0 (/ pi 2) 0 0 0 150)))
  (enlargement-factor
   :accessor enlargement-factor
   :initform 30)))
(defmethod move ((camera strobe-camera) azimuth elevation roll x y z)
 (setf (H-matrix camera) (homogeneous-transform azimuth elevation roll x y z))
 (setf (inverse-H-matrix camera) (inverse-H (H-matrix camera))))

(defmethod take-picture ((camera strobe-camera) (body rigid-body))
 (let ((camera-space-node-list (mapcar #'(lambda (node-location)
      (post-multiply (inverse-H-matrix camera) node-location))
            (transformed-node-list body))))
  (dolist (polygon (polygon-list body))
   (clip-and-draw-polygon camera polygon camera-space-node-list))))

(defmethod clip-and-draw-polygon
 ((camera strobe-camera) polygon node-coord-list)
 (do* ((initial-point (nth (first polygon) node-coord-list))
     (from-point initial-point to-point)
     (remaining-nodes (rest polygon) (rest remaining-nodes))
     (to-point (nth (first remaining-nodes) node-coord-list)
          (if (not (null (first remaining-nodes)))
            (nth (first remaining-nodes) node-coord-list))))
```

```
    ((null to-point)
        (draw-clipped-projection camera from-point initial-point))
    (draw-clipped-projection camera from-point to-point)))

(defmethod draw-clipped-projection ((camera strobe-camera) from-point to-point)
 (cond ((and (<= (first from-point) (focal-length camera))
             (<= (first to-point) (focal-length camera))) nil)
       ((<= (first from-point) (focal-length camera))
        (draw-line-in-camera-window camera
          (perspective-transform camera (from-clip camera from-point to-point))
          (perspective-transform camera to-point)))
       ((<= (first to-point) (focal-length camera))
        (draw-line-in-camera-window camera
          (perspective-transform camera from-point)
          (perspective-transform camera (to-clip camera from-point to-point))))
       (t (draw-line-in-camera-window camera
          (perspective-transform camera from-point)
          (perspective-transform camera to-point)))))

(defmethod from-clip ((camera strobe-camera) from-point to-point)
 (let ((scale-factor (/ (- (focal-length camera) (first from-point))
                        (- (first to-point) (first from-point)))))
    (list (+ (first from-point)
             (* scale-factor (- (first to-point) (first from-point))))
          (+ (second from-point)
             (* scale-factor (- (second to-point) (second from-point))))
          (+ (third from-point)
             (* scale-factor (- (third to-point) (third from-point)))) 1)))

(defmethod to-clip ((camera strobe-camera) from-point to-point)
 (from-clip camera to-point from-point))

(defmethod draw-line-in-camera-window ((camera strobe-camera) start end)
 (cw:draw-line (camera-window camera)
               (cw:make-position :x (first start) :y (second start))
               (cw:make-position :x (first end) :y (second end))
               :brush-width 0))

(defmethod perspective-transform ((camera strobe-camera) point-in-camera-space)
 (let* ((enlargement-factor (enlargement-factor camera))
        (focal-length (focal-length camera))
        (x (first point-in-camera-space))  ;x axis is along optical axis
        (y (second point-in-camera-space)) ;y is out right side of camera
```

```
        (z (third point-in-camera-space))) ;z is out bottom of camera
       (list (+ (round (* enlargement-factor (/ (* focal-length y) x)))
                150)                    ;to right in camera window
        (+ 150 (round (* enlargement-factor (/ (* focal-length (- z)) x))
                      )))))  ;up in camera window

(defun test-camera () ;Produces top view of default rigid-body.
  (setf airplane-1 (make-instance 'rigid-body))
  (initialize  airplane-1)
  (setf camera-1 (make-instance 'strobe-camera))
  (move camera-1 0 (- (/ pi 2)) 0 0 0 -30)
  (take-picture camera-1 airplane-1))
```

APPENDIX D: Replanner Simulation Source Code (LISP)

A. REPLANNER.CL

```
(defconstant cw -1)
(defconstant ccw 1)
(defconstant infinity 32000)

(defconstant invisible-tangent (list (list infinity infinity infinity)
                                     (list infinity infinity infinity)))

(load 'world.cl)

;----------------------REPLANNER CLASS AND METHODS-------------------------

(defclass replannerclass()

  ((world
   :initform ()
   :accessor world)

   (start
    :initform ()
    :accessor start)

   (goal
    :initform ()
    :accessor goal)

   (start-tangents
    :initform ()
    :accessor start-tangents)

   (polygon-tangents
    :initform ()
    :accessor polygon-tangents)

   (goal-tangents
    :initform ()
    :accessor goal-tangents)
```

```
  (dykstra-table
   :initform ()
   :accessor dykstra-table)))

(defmethod plan-route ((replanner replannerclass) obstacles beginning end)
  (with-slots (start goal world start-tangents polygon-tangents goal-tangents) replanner
    (setf start beginning)
    (setf goal end)
    (setf world obstacles)
    (setf start-tangents ())
    (setf polygon-tangents ())
    (setf goal-tangents ())
  (find-tangents replanner world start goal)
  (find-visible-tangents replanner)
  (find-shortest-path replanner))) ;returns the path specified in vertices

(defmethod find-tangents ((replanner replannerclass) world start goal)
  (find-start-tangents replanner)
  (find-all-polygon-tangents replanner)
  (find-goal-tangents replanner world goal))

(defmethod find-visible-tangents ((replanner replannerclass))
  (with-slots (start-tangents goal-tangents) replanner
    (setf start-tangents (find-visible-first-tangents replanner start-tangents))
    (find-visible-polygon-tangents replanner)
    (setf goal-tangents (find-visible-last-tangents replanner goal-tangents))))


;---------- REPLANNER FUNCTIONS ----------

(load 'tangents.cl)


;---------- VISIBLE POLYGON TANGENTS ----------

(load 'visible-polygon.cl)

;---------- START OR GOAL TANGENTS ----------

(load 'visible-tangents.cl)

;----------DYKSTRA SEARCH----------
```

```
(load 'dykstra-search.cl)

;----------MISCELLANEOUS----------

(load 'replan-functions.cl)
```

B. WORLD.CL

```
(setf v11 (list 1 -150 70))
(setf v12 (list 2 -30 10))
(setf v13 (list 3 -10 50))
(setf v14 (list 4 -70 80))
(setf v21 (list 1 -30 80))
(setf v22 (list 2 60 10))
(setf v23 (list 3 90 60))
(setf v31 (list 1 -100 90))
(setf v32 (list 2 80 90))
(setf v33 (list 3 80 110))
(setf v34 (list 4 -100 110))

(setf start (list 0 0 0))
(setf goal (list 0 -50 160))

;Polygons
(setf b1 (list v11 v12 v13 v14))
(setf b2 (list v21 v22 v23))
(setf b3 (list v31 v32 v33 v34))

(setf world (list b1 b2 b3))
```

C. TANGENTS.CL

```
;Find all tangents from the start to all polygons in the world

(defmethod find-start-tangents ((replanner replannerclass))
  (with-slots (start goal world start-tangents) replanner
    (do* ((n 0 (1+ n)))
      ((= n (length world)))
        (setf start-tangents
          (append start-tangents (list
            (list (tangent-from-start-polygon start (nth n world) ccw)
```

```lisp
                (tangent-from-start-polygon start (nth n world) cw))))))
   (setf start-tangents
       (append start-tangents
       (list (list (list start goal) (list start goal)))))))

(defun tangent-from-start-polygon (point polygon mode)
  (do* ((vertex (first polygon) (if (= minus-orientation mode)
                                 plus-next minus-next))
      (minus-next (next vertex polygon (neg mode))
                (next vertex polygon (neg mode)))
      (plus-next (next vertex polygon mode)
               (next vertex polygon mode))
      (minus-orientation (orientation point vertex minus-next)
                      (orientation point vertex minus-next))
      (plus-orientation  (orientation point vertex plus-next)
                      (orientation point vertex plus-next)))
     ((and
       (= minus-orientation mode)
       (/= plus-orientation (neg mode))) (list point vertex))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Find all tangents from the goal to all polygons in the world
(defmethod find-goal-tangents ((replanner replannerclass) world goal)
  (with-slots (goal-tangents) replanner
   (do* ((n 0 (1+ n)))
     ((= n (length world)))
       (setf goal-tangents
        (append goal-tangents (list
          (list (tangent-from-goal-polygon goal (nth n world) ccw)
               (tangent-from-goal-polygon goal (nth n world) cw))))))))

(defun tangent-from-goal-polygon (point polygon mode)
  (do* ((vertex (first polygon) (if (= minus-orientation mode)
                                 plus-next minus-next))
      (minus-next (next vertex polygon (neg mode))
                (next vertex polygon (neg mode)))
      (plus-next (next vertex polygon mode)
               (next vertex polygon mode))
      (minus-orientation (orientation point vertex minus-next)
                      (orientation point vertex minus-next))
      (plus-orientation  (orientation point vertex plus-next)
```

```
                    (orientation point vertex plus-next)))
        ((and
          (= minus-orientation mode)
          (/= plus-orientation (neg mode))) (list vertex point))))
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;Find all of the tangents from one polygon to all other polygons in the world
(defmethod find-all-polygon-tangents ((replanner replannerclass))
  (with-slots (world polygon-tangents) replanner
    (do* ((n 0 (1+ n))
          (from-poly (first world) (nth n world))
          (rest-of-world (rest world) (remove-nth n world)))
         ((= n (length world)))
      (setf polygon-tangents
            (append polygon-tangents (list
              (find-one-polygons-tangents from-poly rest-of-world)))))))

(defun find-one-polygons-tangents (polygon world)
  (let* ((poly-index (first polygon)))
    (do* ((target-polygon (first world) (first remaining-world))
          (remaining-world (rest world) (rest remaining-world))
          (tangent-list
            (list (find-tangents-of-all-modes polygon target-polygon))
            (append tangent-list
                    (list (find-tangents-of-all-modes polygon target-polygon)))))
         ((= 0 (length remaining-world)) tangent-list))))

(defun find-tangents-of-all-modes (poly1 poly2)
  (list (tangent-between-two-polygons poly1 poly2 ccw ccw)
        (tangent-between-two-polygons poly1 poly2 cw ccw)
        (tangent-between-two-polygons poly1 poly2 ccw cw)
        (tangent-between-two-polygons poly1 poly2 cw cw)))

(defun tangent-between-two-polygons (B1 B2 u1 u2)
  (let* ((v1 (first B1))
         (v2 (first B2))
         (flag1 nil)
         (flag2 nil)
         (minus-u2-orientation nil)
         (u2-orientation nil)
         (minus-u1-orientation nil)
         (u1-orientation nil))
```

```
(loop
 (setf minus-u2-orientation (orientation v1 v2 (next v2 B2 (neg u2))))
 (setf u2-orientation (orientation v1 v2 (next v2 B2 u2)))
 (setf minus-u1-orientation (orientation v2 v1 (next v1 B1 (neg u1))))
 (setf u1-orientation (orientation v2 v1 (next v1 B1 u1)))
 (if (= u2 minus-u2-orientation)
  (if (/= (neg u2) u2-orientation) (setf flag2 t) (setf v2 (next v2 B2 u2)))
  (setf v2 (next v2 B2 (neg u2))))
 (if (= u1-orientation (neg u1))
  (if (/= u1 minus-u1-orientation) (setf flag1 t) (setf v1 (next v1 B1 (neg u1))))
  (setf v1 (next v1 B1 u1)))
(if (and flag1 flag2) (return (list v1 v2))))))
```

D. VISIBLE-POLYGONS.CL

```
(defmethod find-visible-polygon-tangents ((replanner replannerclass))
 (with-slots (world polygon-tangents) replanner
 (let* ((list-length (1- (length polygon-tangents))))
  (do* ((n 0 (1+ n))
        (new-polygon-tangents
         (list (single-polygon-visible-tangents (first polygon-tangents)
                                                (remove-nth n world)))
         (append new-polygon-tangents (list
               (single-polygon-visible-tangents (nth n polygon-tangents)
                                                (remove-nth n world))))))
       ((= n list-length)
         (setf polygon-tangents new-polygon-tangents))))))

(defun  single-polygon-visible-tangents (polygon-tangent-list check-world)
 (let* ((list-length (1- (length polygon-tangent-list))))
  (do* ((n 0 (1+ n))
        (visible-list
         (list (visible-line-segments (first polygon-tangent-list)
                                      check-world))
         (append visible-list (list
               (visible-line-segments (nth n polygon-tangent-list)
                                      check-world)))))
       ((= n list-length) visible-list))))

(defun visible-line-segments (tangent-set check-world)
 (do* ((n 0 (1+ n))
       (visible-lines (if (test-if-invisible (first tangent-set) check-world)
```

```
                (list  invisible-tangent)
                (list (first tangent-set)))
              (if (test-if-invisible (nth n tangent-set) check-world)
                (append visible-lines (list invisible-tangent))
                (append visible-lines (list (nth n tangent-set))))))
    ((= n 3) visible-lines)))

;returns t if invisible
(defun  test-if-invisible (tangent check-world)
 (let* ((list-length (1- (length check-world))))
  (do* ((n 0 (1+ n))
      (check-poly (first check-world) (nth n check-world))
      (invisible-flag (check-line-against-poly tangent check-poly)
                (check-line-against-poly tangent check-poly)))
     ((or invisible-flag (= n list-length)) invisible-flag))))

;returns t if invisible
(defun check-line-against-poly (tangent check-poly)
 (do* ((n 0 (1+ n))
     (segment (list (first check-poly)
                (second check-poly))
           (list (nth n check-poly)
                (next (nth n check-poly)
                     check-poly ccw)))
     (crossing-flag (segment-crossing-test tangent segment)
                (segment-crossing-test tangent segment))
     (invisible-flag (if (= crossing-flag -1) nil
                  (if (= crossing-flag 1) t
                   (if (invisible-end-point-test
                        tangent segment
                        (next (nth n check-poly) check-poly cw))
                       t nil)))
                 (if (= crossing-flag -1) nil
                  (if (= crossing-flag 1) t
                   (if (invisible-end-point-test
                        tangent segment
                        (next (nth n check-poly) check-poly cw))
                       t nil)))))
    ((or invisible-flag (= n (1- (length check-poly)))) invisible-flag)))

(defun invisible-end-point-test (tangent segment previous-point)
  (if (and (= 0 (segment-crossing-test tangent segment))
        (= 0 (segment-crossing-test tangent (list previous-point
```

```
                                    (first segment)))))
  (let* ((previous-orientation (orientation (first tangent)
                                    (second tangent)
                                    previous-point))
         (next-orientation (orientation (first tangent)
                                    (second tangent)
                                    (second segment))))
        (if (or (and (>= previous-orientation 0)
                     (>= next-orientation 0))
                (and (<= previous-orientation 0)
                     (<= next-orientation 0)))
            nil t))
  nil))
```

E. VISIBLE-TANGENTS.CL

```
(defmethod find-visible-first-tangents ((replanner replannerclass) tangent-list)
 (let* ((list-length (1- (length tangent-list))))
  (do* ((n 0 (1+ n))
        (check-list (rest tangent-list) (remove-nth n tangent-list))
        (new-visible-tangents
         (list (first-point-visible-tangents (first tangent-list) check-list))
         (append new-visible-tangents
                 (list (first-point-visible-tangents (nth n tangent-list)
                                                     check-list)))))
       ((= n list-length) new-visible-tangents))))


(defmethod find-visible-last-tangents ((replanner replannerclass) tangent-list)
 (let* ((list-length (1- (length tangent-list))))
  (do* ((n 0 (1+ n))
        (check-list (rest tangent-list) (remove-nth n tangent-list))
        (new-visible-tangents
         (list (last-point-visible-tangents (first tangent-list) check-list))
         (append new-visible-tangents
                 (list (last-point-visible-tangents (nth n tangent-list)
                                                    check-list)))))
       ((= n list-length) new-visible-tangents))))

(defun first-point-visible-tangents (tangent-set check-list)
 (do* ((n 0 (1+ n))
       (visible-lines (if (test-first-invisible (first tangent-set) check-list)
```

```
                    (list invisible-tangent)
                    (list (first tangent-set)))
                (if (test-first-invisible (second tangent-set) check-list)
                    (append visible-lines (list invisible-tangent))
                    (append visible-lines (list (nth n tangent-set))))))
    ((= n 1) visible-lines)))

(defun last-point-visible-tangents (tangent-set check-list)
 (do* ((n 0 (1+ n))
     (visible-lines (if (test-last-invisible (first tangent-set) check-list)
                    (list invisible-tangent)
                    (list (first tangent-set)))
                (if (test-last-invisible (second tangent-set) check-list)
                    (append visible-lines (list invisible-tangent))
                    (append visible-lines (list (nth n tangent-set))))))
    ((= n 1) visible-lines)))

(defun test-first-invisible (tangent check-list) ;returns t if invisible
 (let* ((list-length (1- (length check-list))))
  (do* ((n 0 (1+ n))
      (segment (list (second (first (first check-list)))
                (second (second (first check-list))))
            (list (second (first (nth n check-list)))
                (second (second (nth n check-list)))))
      (invisible-flag (if (< 0 (segment-crossing-test tangent segment)) t nil)
                (if (< 0 (segment-crossing-test tangent segment))
                   t invisible-flag)))
    ((or invisible-flag
       (= n list-length)) invisible-flag))))

(defun test-last-invisible (tangent check-list) ;returns t if invisible
 (let* ((list-length (1- (length check-list))))
  (do* ((n 0 (1+ n))
      (segment (list (first (first (first check-list)))
                (first (second (first check-list))))
            (list (first (first (nth n check-list)))
                (first (second (nth n check-list)))))
      (invisible-flag (if (< 0 (segment-crossing-test tangent segment)) t nil)
                (if (< 0 (segment-crossing-test tangent segment))
                   t invisible-flag)))
    ((or invisible-flag
       (= n list-length)) invisible-flag))))
```

F. DYKSTRA-SEARCH.CL

```
(defstruct dir-poly
  mark
  cost
  land
  previous
  leave
  area)

;High level function which implements the dykstra search.
(defmethod find-shortest-path ((replanner replannerclass))
  (build-dykstra-table replanner) ;method to build the table slot
  (dykstra-search replanner)
  (return-shortest-path replanner))

;Builds a table which contains state information for each of the directed
;polygons in the world.
(defmethod build-dykstra-table ((replanner replannerclass))
   (with-slots (start world dykstra-table) replanner
   (setf dykstra-table (list (make-dir-poly :mark 1
                                    :cost 0
                                    :land start
                                    :previous start
                                    :area 0)))
   (do ((n 0 (1+ n)))
     ((= n (* 2 (length world))) (setf dykstra-table (append dykstra-table
                                          (list (make-dir-poly
                                                :mark 0
                                                :cost infinity
                                                :leave -1
                                                :area 0)))))
     (setf dykstra-table (append dykstra-table (list (make-dir-poly
                                               :mark 0
                                               :cost infinity
                                               :leave infinity
                                               :area 0)))))))

(defmethod dykstra-search ((replanner replannerclass))
  (with-slots (dykstra-table) replanner
    (do* ((z-index (go-from-z replanner 0)(go-from-z replanner z-index)))
       ((equal z-index (1- (length dykstra-table)))))))

(defmethod go-from-z ((replanner replannerclass) z-index)
```

```
 (dotimes (n (polygon-nodes replanner z-index) (find-next-z replanner))
  (mark-landings replanner z-index n)))

(defmethod find-next-z ((replanner replannerclass))
  (with-slots (dykstra-table) replanner
  (let* ((min-cost-index 0)
        (min-cost infinity))
    (do* ((index 1 (1+ index)))
        ((= index (length dykstra-table))
         (setf (dir-poly-mark (nth min-cost-index dykstra-table)) 1)
         min-cost-index)
         (if (and (/= (dir-poly-mark (nth index dykstra-table)) 1)
                  (< (dir-poly-cost (nth index dykstra-table)) min-cost))
                    (and (setf min-cost (dir-poly-cost (nth index dykstra-table)))
                         (setf min-cost-index index)))))))

(defmethod mark-landings ((replanner replannerclass) z-index vertice-index)
  (with-slots (dykstra-table) replanner
  (let* ((leaving-vertice (which-vertice replanner z-index vertice-index))
        (landing-list (return-landings replanner z-index leaving-vertice))
        (vertice-cost (boundary-distance replanner
                       z-index
                       (dir-poly-land (nth z-index dykstra-table))
                       leaving-vertice))
        (landing-vertice (dir-poly-land (nth z-index dykstra-table)))
        (z-area (dir-poly-area (nth z-index dykstra-table))))
  (dotimes (n (length landing-list) ())
   (let*
    ((z-prime (first (nth n landing-list)))
     (z-prime-index (second (nth n landing-list)))
     (z-prime-area (dir-poly-area (nth z-prime-index dykstra-table)))
     (w (+ (dir-poly-cost (nth z-index dykstra-table))
           vertice-cost
           (distance (rest leaving-vertice) (rest z-prime))))
     (a0 (+ z-area (big-d (rest landing-vertice) (rest leaving-vertice))
                   (big-d (rest leaving-vertice) (rest z-prime)))))
     (if (or (= (dir-poly-cost (nth z-prime-index dykstra-table)) infinity)
          (and (= (sign (mode replanner z-prime-index))
                  (sign (+ z-prime-area
                           (big-d
                            (rest (dir-poly-land (nth z-prime-index dykstra-table)))
                            (rest z-prime))
                           (neg  a0))))
```

```
            (< (+ w
                  (boundary-distance replanner
                    z-prime-index
                    z-prime
                    (dir-poly-land (nth z-prime-index dykstra-table))))
                (dir-poly-cost (nth z-prime-index dykstra-table))))
          (and (/= (sign (mode replanner z-prime-index))
                   (sign (+ z-prime-area
                            (big-d
                              (rest (dir-poly-land (nth z-prime-index dykstra-table)))
                              (rest z-prime))
                            (neg  a0))))
               (< (- w
                     (boundary-distance replanner
                       z-prime-index
                       (dir-poly-land (nth z-prime-index dykstra-table))
                       z-prime))
                  (dir-poly-cost (nth z-prime-index dykstra-table)))))
        (and
         (setf (dir-poly-cost (nth z-prime-index dykstra-table)) w)
         (setf (dir-poly-land (nth z-prime-index dykstra-table)) z-prime)
         (setf (dir-poly-previous (nth z-prime-index dykstra-table)) z-index)
         (setf (dir-poly-leave (nth z-prime-index dykstra-table)) leaving-vertice)
         (setf (dir-poly-area (nth z-prime-index dykstra-table)) a0))))))))

(defmethod which-vertice ((replanner replannerclass) table-index vertice-index)
 (with-slots (start goal dykstra-table) replanner
  (if (= table-index 0) start
   (if (= table-index (1- (length dykstra-table))) goal
    (let* ((new-vertex (dir-poly-land (nth table-index dykstra-table))))
     (dotimes (n vertice-index new-vertex)
      (setf new-vertex (jump-one replanner table-index new-vertex)))))))))

(defmethod boundary-distance ((replanner replannerclass) table-index s-point vertice)
 (with-slots (dykstra-table) replanner
  (if (or (= table-index 0)
          (= table-index (1- (length dykstra-table)))
          (equal vertice s-point)) 0
   (do* ((old-vertex s-point
                     new-vertex)
         (new-vertex (jump-one replanner table-index s-point)
                     (jump-one replanner table-index new-vertex))
         (distance-total (distance (rest old-vertex) (rest new-vertex))
```

```
                (+ distance-total
                  (distance (rest old-vertex) (rest new-vertex)))))
      ((equal new-vertex vertice) distance-total)))))

(defmethod return-landings ((replanner replannerclass) z-index vertice)
 (with-slots (polygon-tangents) replanner
  (let* ((polygon (if (= 0 z-index) 0 (floor (/ (- z-index 1) 2))))
       (landing-list (if (> z-index 0) (check-one-poly vertice polygon
                                  (nth polygon polygon-tangents))
                  (check-start-leaves replanner)))
       (goal-landings (check-goal-landings replanner vertice polygon)))
      (if (equal goal-landings nil) landing-list
                  (append landing-list goal-landings)))))

(defmethod check-goal-landings ((replanner replannerclass) vertex polygon)
 (with-slots (goal-tangents dykstra-table) replanner
  (let* ((goal1 (if (equal vertex (first (first (nth polygon goal-tangents))))
           (list (second (first (nth polygon goal-tangents)))
              (1- (length dykstra-table))) nil))
       (goal2 (if (equal vertex (first (second (nth polygon goal-tangents))))
           (list (second (second (nth polygon goal-tangents)))
              (1- (length dykstra-table))) nil)))
      (if (equal goal1 nil)
        (if (equal goal2 nil) nil (list goal2))
        (if (equal goal2 nil) (list goal1) (list goal1 goal2))))))

(defun check-one-poly  (vertex polygon tangent-list)
  (let* ((partial-list nil))
   (dotimes (n (length tangent-list) partial-list)
      (if (equal vertex (first (nth 0 (nth n tangent-list))))
       (setf partial-list (append partial-list
           (list (list
           (second (nth 0 (nth n tangent-list)))
           (if (>= n polygon) (- (* (+ n 2) 2) 1)
                  (- (* (+ n 1) 2) 1)))))))
      (if (equal vertex (first (nth 1 (nth n tangent-list))))
       (setf partial-list (append partial-list
           (list (list
           (second (nth 1 (nth n tangent-list)))
           (if (>= n polygon) (- (* (+ n 2) 2) 1)
                  (- (* (+ n 1) 2) 1)))))))
      (if (equal vertex (first (nth 2 (nth n tangent-list))))
       (setf partial-list (append partial-list
```

```
            (list (list
            (second (nth 2 (nth n tangent-list)))
            (if (>= n polygon) (* (+ n 2) 2)
                        (* (+ n 1) 2)))))))
       (if (equal vertex (first (nth 3 (nth n tangent-list))))
        (setf partial-list (append partial-list
            (list (list
            (second (nth 3 (nth n tangent-list)))
            (if (>= n polygon) (* (+ n 2) 2)
                        (* (+ n 1) 2)))))))))))

(defmethod check-start-leaves ((replanner replannerclass))
 (with-slots (start start-tangents) replanner
  (let* ((partial-list ()))
   (dotimes (n (length start-tangents) partial-list)
     (if (equal start (first (first (nth n start-tangents))))
      (setf partial-list (append partial-list (list (list
            (second (first (nth n start-tangents)))
            (- (* (+ n 1) 2) 1))))))
     (if (equal start (first (second (nth n start-tangents))))
      (setf partial-list (append partial-list (list (list
            (second (second (nth n start-tangents)))
            (* (+ n 1) 2)))))))))

(defmethod polygon-nodes ((replanner replannerclass) table-index)
 (with-slots (dykstra-table) replanner
  (if (or (= table-index 0) (= table-index (1- (length dykstra-table)))) 1
   (length (find-polygon replanner table-index)))))

(defmethod mode ((replanner replannerclass) table-index)
 (with-slots (dykstra-table) replanner
  (if (or (= 0 table-index)
       (= table-index (1- (length dykstra-table))))
   0 (if (even table-index) cw ccw))))

(defmethod find-polygon ((replanner replannerclass) table-index)
 (with-slots (start goal world dykstra-table) replanner
  (if (= 0 table-index) start
   (if (= table-index (1- (length dykstra-table))) goal
     (nth (floor (/ (- table-index 1) 2)) world)))))

(defmethod jump-one ((replanner replannerclass) table-index vertex)
 (next vertex (find-polygon replanner table-index) (mode replanner table-index)))
```

```
(defmethod return-shortest-path ((replanner replannerclass))
 (with-slots (goal world dykstra-table) replanner
  (let* ((path-list (list goal)))
  (do* ((cur-index (+ (* (length world) 2) 1)
              (dir-poly-previous (nth cur-index dykstra-table)))
       (previous-index (dir-poly-previous (nth cur-index dykstra-table))
                 (dir-poly-previous (nth cur-index dykstra-table))))
      ((= cur-index 0) (reverse path-list))
       (if (equal (dir-poly-leave (nth cur-index dykstra-table))
              (dir-poly-land (nth previous-index  dykstra-table)))
         (setf path-list  (append  path-list (list
                              (dir-poly-leave (nth cur-index dykstra-table)))))
         (setf path-list  (append path-list
                     (traverse-perimeter replanner
                       previous-index
                       (dir-poly-leave (nth cur-index dykstra-table))
                       (dir-poly-land (nth previous-index dykstra-table)))))))))

(defmethod traverse-perimeter ((replanner replannerclass) index leave land)
  (with-slots (world dykstra-table) replanner
  (do* ((old-vertex leave  new-vertex)
       (new-vertex (next old-vertex (find-polygon replanner index) (neg (mode replanner
index)))
             (next old-vertex (find-polygon replanner index) (neg (mode replanner index))))
       (vertex-list (list new-vertex leave) (cons new-vertex vertex-list)))
  ((equal new-vertex land) (reverse vertex-list)))))
```

G. REPLAN-FUNCTIONS.CL

```
(defconstant limit 0.000001)

(defun sign (x)
 (if (< x (neg limit)) -1
     (if (> x limit) 1 0)))

(defun orientation(p1 p2 p3)
  (let ((area (* 0.5 (- (* (- (second p2) (second p1))
                    (- (third p3) (third p1)))
                 (* (- (second p3) (second p1))
                    (- (third p2) (third p1)))))))
  (if (> area 0.0) ccw
    (if (< area 0.0) cw 0))))
```

```
(defun next(vertex polygon mode)
  (let ((i (first vertex)))
  (if (> mode 0)
    (if (= i (length polygon)) (first polygon)
                    (nth i polygon))
    (previous vertex polygon))))

(defun previous(vertex polygon)
  (let ((i (first vertex)))
  (if (= i 1) (nth (1- (length polygon)) polygon)
          (nth (- i 2) polygon))))

(defun from-n-on (n list) ;elements numbered 0 1 2 ...
  (cond ((>= n (length list)) ())
      (t (cons (nth n list) (from-n-on (1+ n) list)))))

(defun remove-nth (n list) ;elements numbered 0 1 2 ...
  (append (firstj n  list) (from-n-on (1+ n) list)))

(defun make-line-segment(v1 v2)
  (list v1 v2))

(defun segment-crossing-test(s1 s2)
  (let* ((o1 (orientation (first s1) (second s1) (first s2)))
       (o2 (orientation (first s1) (second s1) (second s2)))
       (o3 (orientation (first s2) (second s2) (first s1)))
       (o4 (orientation (first s2) (second s2) (second s1))))
  (if (crossing-test s1 s2 o1 o2 o3 o4) 1
    (if (touching-test s1 s2 o1 o2 o3 o4) 0
      (if (overlap-test s1 s2 o1 o2 o3 o4) 0 -1)))))

(defun crossing-test(s1 s2 o1 o2 o3 o4)
  (if (and (/= o1 0) (/= o2 0) (/= o1 o2)
        (/= o3 0) (/= o4 0) (/= o3 o4)) t ()))

(defun touching-test(s1 s2 o1 o2 o3 o4)
  (if (or (and (or (= o1 0) (= o2 0)) (and (/= o3 0) (/= o4 0) (/= o3 o4)))
        (and (or (= o3 0) (= o4 0)) (and (/= o1 0) (/= o2 0) (/= o1 o2)))
        (and (and (or (= o1 0) (= o2 0)) (/= o1 o2)) (and (or (= o3 0)
                                  (= o4 0)) (/= o3 o4)))) t ()))

(defun overlap-test(s1 s2 o1 o2 o3 o4)
```

```
  (if (= (and o1 o2 o3 o4) 0)
      (let* ((f1 (linearize (first s1) s1))
          (f2 (linearize (second s1) s1))
          (f3 (linearize (first s2) s1))
          (f4 (linearize (second s2) s1)))
        (if (or (and (< f3 0.0) (< f4 0.0))
              (and (> f3 f2) (> f4 f2))) nil
          (if (or (and (<= f3 f2) (>= f3 f1))
                (and (<= f4 f2) (>= f4 f1))) t ())))))

(defun linearize(v l)
  (let* ((f1 (* (- (second v) (second (first l)))
          (- (second (second l)) (second (first l)))))
      (f2 (* (- (third v) (third (first l)))
          (- (third (second l)) (third (first l))))))
   (+ f1 f2)))

(defun big-d (v1 v2)
  (* 0.5 (- (* (first v1) (second v2)) (* (first v2) (second v1)))))


(defun even  (number)
 (if (= 0 (mod number 2)) t nil))
```

H. WIND-TRI.CL

```
(defconstant two-pi 6.283185307179586)  ;pi multiplied by two

(defconstant pi-two 1.5707963267948966) ;pi divided by two

(defun radian-true-course (from-point to-point)
  (positive-radians (atan (- (second to-point) (second from-point))
                (- (first to-point) (first from-point)))))

(defun positive-radians (radians)
  (if (< radians 0) (+ radians two-pi)  radians));

(defun calculate-heading (speed course current)
  (cond ((zerop speed) course)
      (t (let* ((A (-  course (second current)))
            (b (first current))
            (a speed)
            (B (asin (/ (* b (sin A)) a))))
```

```
            (+ course B)))))

(defun square (x) (* x x))

(defun deg-to-rad (deg) (* deg (/ pi 180)))

(defun rad-to-deg (rad) (* rad (/ 180 pi)))

(defun distance (from-point to-point)
 (let*((x-delta (- (first to-point)
                   (first from-point)))
       (y-delta (- (second to-point)
                   (second from-point))))
      (sqrt (+ (square x-delta) (square y-delta)))))

(defun find-vector (from-point to-point)
 (list (distance from-point to-point)
       (radian-true-course from-point to-point)))


(defun firstj (n list) ;improved version
 (cond ((or(zerop n)(null list)) nil)
       (t (cons (first list) (firstj (1- n) (rest list))))))

(defun power (exponent base)
 (do* ((i exponent (1- i))
       (result base (* result base)))
      ((= i 1) result)))

(defun neg (number)
 (* -1 number))

(defun angle-trans (heading)

 (if (> heading pi) (- heading two-pi) heading))

;normalizes alpha to between pi and -pi
(defun fee (alpha)
 (if (> alpha pi) (fee(- alpha two-pi))
  (if (<= alpha (neg pi)) (fee(+ alpha two-pi)) alpha)))
```

APPENDIX E: Tattletale Source Code (TxBASIC)

A. TATTLETALE CODE

Written by Walter Schubert for the towfish experiment.

```
 3 SLEEP 0
  4 SLEEP 2000
  5 PRINT "ATA"
  6 SLEEP 3000
 10 A=16384
 20 B=A
 30 BURST B,8,2
 40 IF B < (A+128) GOTO 30
 50 OFFLD A,A+127,100,C
 60 GOTO 20
```

LIST OF REFERENCES

Bergem, O., "A Multibeam Sonar Based Positioning System for an AUV," *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology*, Durham, New Hampshire, September 27-29 1993, pp. 291-299.

Bowditch, N., *American Practical Navigator, Vol. 1 and 2*, Defense Mapping Agency Hydrographic/Topographic Center, 1984.

Brown, R.G and Hwang, P.Y.C., *Introduction to Random Signals and Applied Kalman Filters*, 2nd Edition, John Wiley and Sons, New York, 1992.

Brutzman, D.P., *A Virtual World for an Autonomous Underwater Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, December, 1994. Available at *http://www.sh.nps.navy.mil/ubrutzman/dissertation*.

Byrnes, R.B., Nelson, M.L., McGhee, R.B., Kwak, S.H. and Healey, A.J., "Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control Of Autonomous Underwater Vehicles," *Proceedings of the Eighth International Symposium on Unmanned Untethered Submersible Technology (UUST)*, September 27-29 1993, Durham, New Hampshire, pp. 160-178.

Cox, I.J. and Wilfong, G.T., *Autonomous Robot Vehicles*, Springer-Verlag, New York, 1990.

Clynch, J.R., "Error Characteristics of GPS Differential Positions and Velocities," *Proceedings Institute of Navigation GPS-92*, Albuquerque, New Mexico, September, 1992.

Davidson, S.L., *An Experimental Comparison of CLOS and C++ Implementations of an Object-Oriented Graphical Simulation of Walking Robot Kinematics*, Master's Thesis, Naval Postgraduate School, Monterey, California, March, 1993.

Dutton, B., *Dutton's Navigation and Piloting, 14thEdition*, Naval Institute Press, Annapolis, Maryland, 1985.

Fu, K.S., Gonzalez, R.C. and Lee, C.S.G., *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, Inc., New York, 1987.

Grose, B.L., "The Application of the Correlation Sonar to Autonomous Underwater Vehicle Navigation," *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology*, IEEE Oceanic Engineering Society, June 2-3 1992, Washington, D.C., pp. 298-303.

Healey, A.J. "Evaluation of the NPS PHOENIX Autonomous Underwater Vehicle Hybrid Control System," *Proceedings of ACC'95 Conference*, Seattle, Washington, June, 1995.

Healey, A.J. and Lienard, D., "Multivariable Sliding Mode Control for Autonomous Diving and Steering of Unmanned Underwater Vehicles," *IEEE Journal of Oceanic Engineering*, vol. 18 no. 3, July, 1993.

Kanayama, Y., *CS4313: Lecture Notes Introduction to Motion Planning*, March, 1995.

Koschmann, T.D., *The Common Lisp Companion*, John Wiley and Sons, New York, 1990.

Kwak, S.H., Stevens, C.D., Clynch, J.R., McGhee, R.B. and Whalen, R.H., "An Experimental Investigation of GPS/INS integration for Small AUV Navigation," *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology (UUST)*, September 27-29 1993, Durham, New Hampshire, pp. 239-251.

Leu, C.T., Chao, J.J. and Lee, T.S., "GPS Based Underwater Positioning - A System Design," *Proceedings of The Institute of Navigation GPS-93*, Salt Lake City, Utah, September 22-24 1993, pp. 745-754.

Logsdon, T., *The Navstar Global Positioning System*, Van Nostrand Reinhold, New York, 1992.

LTC-1290 A-D Converter, Linear Technology Corporation, Milpitas, California.

McGhee, R.B., Clynch, J.R., Healey, S.H., Kwak, S.H., Brutzman, D.P., Yun, X.P., Norton, N.A., Whalen, R.H., Bachmann, E.R., Gay, D.L. and Schubert, W.R., "An Experimental Study of an Integrated GPS/INS System for Shallow-Water AUV Navigation (SANS), *Proceedings of the Ninth International Symposium on Unmanned Untethered Submersible Technology (UUST)*, September 25-27 1995, Durham, New Hampshire.

McKeon, J.B., *Integration of GPS into a Small Underwater Navigation System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March, 1992.

Motorola GPS Receiver Technical Reference Manual, Motorola Inc, October, 1993, pp. 4-11, 22-27.

Motorola GPS Evaluation Kit Quick Start Guide, Motorola Inc, June, 1993, pp. 2-1 to 4-17, A-1 to A-15.

Moya, D.C. and Elchynski, J.J., "Evaluation of the Worlds's Smallest Integrated Embedded GPS/INS, the H-764G", *The Institute of Navigation Proceedings of 1993 National Technical Meeting*, San Francisco, California, January 20-22 1993, pp. 275-286.

Nagengast, S., *Correction of Inertial Measurements Using GPS Update for Underwater Navigation*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992, pp. 4-6.

Norton, N.A., *Evaluation of Hardware and Software for a Small Autonomous Underwater Vehicle Navigation System (SANS)*, Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1994.

Parkinson, B.W., "Overview," *Global Positioning System*, Vol. 1, The Institute of Navigation, Washington, D.C., 1980, pp. 1-2.

Practical Pocket Modem 14400FX Operating Manual, Practical Peripherals Corporation, 1992.

Ramadage, R. and Wohnam, W.M., "The Control of Discrete Event Systems," *Proceedings of the IEEE*, vol. 77, no. 1, January, 1989.

Schubert, W.R. and Whalen, R.H., "Design and Implementation of an Integrated GPS/INS System for Shallow-Water AUV Navigation," *Technical Report No. CS-95-003*, Naval Postgraduate School, Monterey, California 93943, July, 1995.

Souen, K., and Nishida, T., "The World's Smallest 8-Channel GPS Receiver," *Proceedings of The Institute of Navigation GPS-92*, Albuquerque, New Mexico, September 16-18 1992, pp. 707-713.

Stevens, C.D., A *Software Architecture for a Small Autonomous Underwater Vehicle Navigation System*, Master's Thesis, Naval Postgraduate School, Monterey, California, June, 1993.

Systron-Donner Model MP-GCCCQAAB MotionPaK IMU, Systron-Donner, Inc., Concord, California.

The Tattletale Model 5F-LCD Manual, Onset Computer Corporation, February, 1994, pp. 1-9.

Tattletale Application Notes, Onset Computer Corporation, February, 1994, pp. 2-17.

Tuohy, S.T., Patrikalakis, N.M., Leonard, J.J., Bellingham, J.G. and Chryssostomidis, C., "AUV Navigation Using Geophysical Maps with Uncertainty," *Proceedings of the 8th International Symposium on Unmanned Untethered Submersible Technology (UUST)*, Durham, New Hampshire, September 27-29 1993, pp. 265-276.

Tritech DS30 Precision Doppler Sonar Technical Manual, Tritech International Limited, Aberdeen AB32 6JL, United Kingdom.

Van Dierendonck, A.J., Russell, S.S., Kopitzke, E.R. and Birnbaum, M., "The GPS Navigation Message," Global Positioning System, vol. 1, *The Institute of Navigation*, Washington, DC, 1980, pp.55-73.

Wooden, W. H., "NAVSTAR Global Positioning System: 1985", *Proceedings of the First International Symposium on Precise Positioning with the Global Positioning System*, April 1985, pp. 23-32.

Youngberg, J.W., "A Novel Method for Extending GPS to Underwater Applications", *NAVIGATION, Journal of The Institute of Navigation*, vol. 38, no. 3, Fall 1991.

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

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 013 — 2
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Director, Training and Education — 1
   MCCDC, Code C46
   1019 Elliot Rd.
   Quantico, VA 22134-5027

4. Chairman, Code CS — 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Dr. Robert B. McGhee, Code CS/Mz — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Dr. Don Brutzman, Code UW/Br — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

7. Dr. James Clynch, Code OC/CL — 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

8. Lt. Eric Bachmann — 2
   1281 Spruance Rd.
   Monterey, CA 93940

9. LCDR David Gay — 2
   P.O. Box 2448
   Deming, NM 88031

10. Frank Kelbe 1
7108 Gisele Dr. NE
Albuquerque, NM 87109-3798

11. Russ Whalen, Code CS 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

12. Guy Oliver 1
University of California, Santa Cruz
233 Northrop Place
Santa Cruz, CA 95060

13. Dr. Anthony J. Healey, Code ME/Hy 1
Department of Mechanical Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

14. Dr. Doug Troy 1
System Analysis Department
Miami University
Oxford, OH 95056

15. Mr. Norman Caplen 1
National Science Foundation
BES, Room 565
4201 Wilson Blvd.
Arlington, VA 22230