NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

TESTING THE HG1700 INERTIAL MEASUREMENT UNIT FOR IMPLEMENTATION INTO THE ARIES UNMANNED UNDERWATER VEHICLE

by

Joel Gow

June 2005

Thesis Advisor: Anthony J. Healey
Co-Advisor: Edward B. Thornton

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2005	Master's Thesis

4. TITLE AND SUBTITLE: Testing the HG1700 Inertial Measurement Unit for Implementation into the ARIES Unmanned Underwater Vehicle	5. FUNDING NUMBERS
6. AUTHOR(S) Joel Gow	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

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

13. ABSTRACT (maximum 200 words)

The ARIES Unmanned Underwater Vehicle (UUV) currently uses an Inertial Measurement Unit (IMU) with an inherent rotation rate error bias of 10 degrees/hour. Then need for a more accurate IMU for long term missions has led to the purchase of the Honeywell HG1700 IMU. The HG1700 is a ring laser gyroscope designed specifically as part of the navigation software in multiple U.S. missiles. The objective of this research is to perform numerous bench tests on the HG1700 to test its capabilities and to begin the process of implementing the IMU into the ARIES unmanned underwater vehicle. Specifically, the IMU is tested for correct setup configurations, angle of rotation accuracies, the rotation rate error bias, and positional accuracies. Also, guidelines for integrating the IMU with the current software in the ARIES vehicle are discussed.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Inertial Measurement Unit, Unmanned Underwater Vehicle, ARIES Vehicle	81
	16. PRICE CODE

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

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

TESTING THE HG1700 INERTIAL MEASUREMENT UNIT FOR IMPLEMENTATION INTO THE ARIES UNMANNED UNDERWATER VEHICLE

Joel A. Gow
Ensign, United States Navy
B.S., United States Naval Academy, 2004

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED SCIENCE
(PHYSICAL OCEANOGRAPHY)

from the

NAVAL POSTGRADUATE SCHOOL
June 2005

Author: Joel Gow

Approved by: Anthony J. Healey
Thesis Advisor

Edward B. Thornton
Co-Advisor

Mary L. Batteen
Chairman, Department of Oceanography

Donald P. Brutzman
Chair, Undersea Warfare Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The ARIES Unmanned Underwater Vehicle (UUV) currently uses an Inertial Measurement Unit (IMU) with an inherent rotation rate error bias of 10 degrees/hour. The need for a more accurate IMU for long term missions led to the purchase of the Honeywell HG1700 IMU. The HG1700 is a ring laser gyroscope designed specifically as part of the navigation software in multiple U.S. missiles. The objective of this research is to perform numerous bench tests on the HG1700 to test its capabilities and to begin the process of implementing the IMU into the ARIES unmanned underwater vehicle. Specifically, the IMU is tested for correct setup configurations, angle of rotation accuracies, the rotation rate error bias, and positional accuracies. Also, guidelines for integrating the IMU with the current software in the ARIES vehicle are discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis could not have been completed without the help and guidance of Professor Healey. Special thanks also go to the Benjamin Wring, who performed all the electrical setup for the tests, and Doug Horner, who was always willing to lend his assistance. Finally, I would like thank my loving fiancée, Kristen Ford, for consistently offering support and encouragement and for understanding and respecting the time I needed to commit to this work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION TO RING LASER GYROSCOPES

A. HISTORY OF THE GYROSCOPE

A gyroscope is any device used to create a fixed direction in space or measure the angular rate and position of the platform to which it is mounted with respect to a fixed reference frame [1]. The idea of the gyroscope originated from the concept and observations of the spinning top. Throughout history the top was used mainly for entertainment, but in the late 18^{th} century, scientists began to notice the top for its unique balancing capabilities. The top was first used as a navigational tool in the 1740s by an English scientist named Serson. His idea was to use the top in conjunction with the sextant as an artificial horizon for ships at sea when the weather made visibility of the true horizon impossible. Although the experiment failed, the potential of the top continued to grow [2].

In 1810, G.C.Bohnenberger invented the first modern gyroscope, but since it lacked any scientific purposes, the credit for the first gyroscope usually goes to the French scientist Jean-Bernard-Leon Foucault. In 1852, Foucault used a wheel mounted in gimbal rings to successfully measure the rotation of the earth, and he was the first to coin the term 'gyroscope' – a combination of the Greek words "gyros" (revolution) and "skopein" (to see). From that time on, the gyroscope has been used as a navigational tool and a stabilizer in numerous platforms including torpedoes, ships, airplanes, satellites, and unmanned vehicles [2].

B. HISTORY OF RING LASER GYROSCOPE

Since its inception, the gyroscope has taken on many forms that have deviated from the original spinning-mass mechanical system. Out of a need to reduce the required maintenance on such a rapidly spinning piece of equipment and increase the production and installation times for the original gyroscope, scientist began looking for alternative methods to measure fixed direction and angular rate. As a result, in 1962, Warren Macek, from the Sperry Corporation, developed the first "gyroscope" without any moving parts: the Ring Laser Gyroscope (RLG) [3].

The RLG's functionality is based on the Sagnac effect, named after the scientist who first successfully demonstrated the effect in 1913 [4]. The Sagnac effect holds that if two identical beams of light (equal wavelength and phase) are sent in opposite directions around a stationary closed path, the two beams will arrive simultaneously at the opposite end of the enclosed path, since the speed of light is constant. If, however, the closed path undergoes a rotation while the light waves are traveling along the path, then the ray traveling in the direction of the rotation will take longer to travel around the path than the ray traveling opposite the direction of rotation. This time delay due to the rotation of the path causes the two beams of light to be out of phase upon reaching the end of the path, and this phase difference can be measured to return the rotation rate [1]. The mathematical derivation of the Sagnac effect can be seen below.

1. Derivation of the Sagnac Effect

Since most RLG's use a triangular path, the derivation will begin with the geometric representation of an RLG shown in Figure 1.

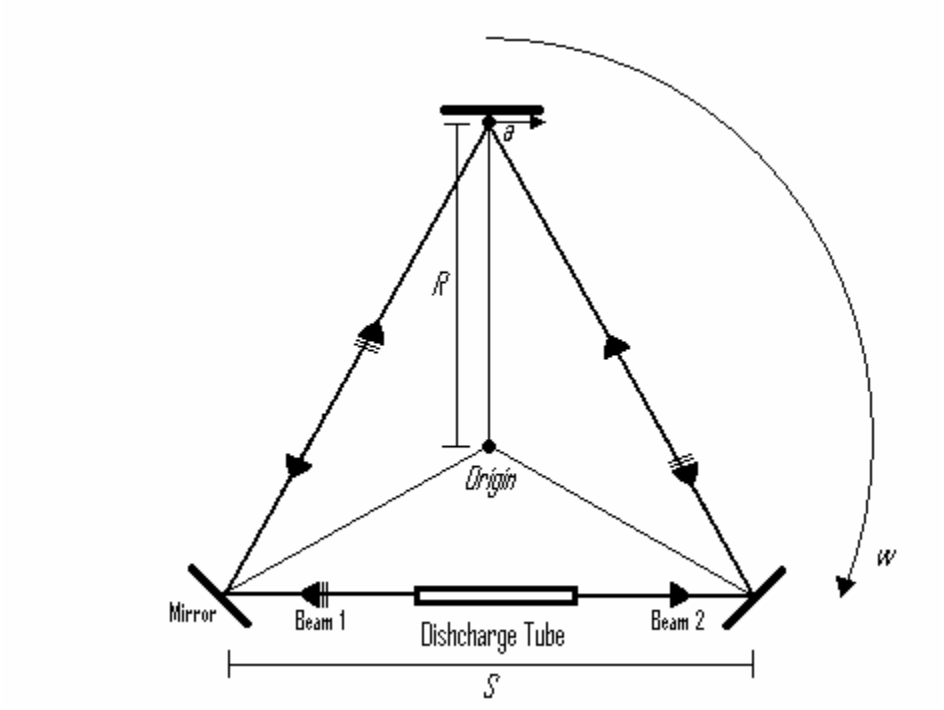

Figure 1. Equilateral Triangle (After Ref. [4])

Given that S is the side length, and $P = 3S$ is the total perimeter, then the time it takes for one beam to travel the complete path of the triangle is

$$t = \frac{3S}{c} = \frac{P}{c}$$

where c is the speed of light.

The velocity of point *a*, given an angular rotation rate of *w* (rad/sec), is

$$v = wR$$

Note that *R* is the distance from the origin of the triangle to point *a*, and can be redefined as

$$\cos(30°) = \frac{\left(\frac{S}{2}\right)}{R}$$

$$R = \frac{S}{2}\left(\frac{1}{\cos(30°)}\right) = \frac{S}{2}\sec(30°) = \frac{S}{\sqrt{3}}$$

Thus for a rotation rate *w*, point *a* moves a distance

$$d = vt = v\frac{P}{c} = \left(w\frac{P}{c}\right)\frac{S}{\sqrt{3}}$$

The change in path length of beam 1 is simply the component of the distance traveled by point *a* along the direction of the beam path.

$$\Delta P = d\cos(60°) = \frac{d}{2} = \frac{wPS}{2c\sqrt{3}} = \frac{wS^2\sqrt{3}}{2c}$$

This equation can be simplified given that the area of the triangle is defined as follows:

$$A = \frac{1}{2}S(S\sin(60°)) = \frac{S^2\sqrt{3}}{4}$$

Therefore

$$\Delta P = \frac{2wA}{c}$$

Since the path length must have an integral number of wavelengths, P must also be defined as

$$P = n\lambda \qquad n = 1,2,3...$$

Thus, when the path length changes by ΔP, the wavelength changes by

$$\Delta\lambda = \frac{\Delta P}{n} = \frac{\lambda \Delta P}{P}$$

Likewise the frequency change can be defined as

$$\frac{\Delta f'}{f'} = \frac{\Delta\lambda}{\lambda} = \frac{\Delta P}{P}$$

But since both beams experience the same frequency shift in opposite directions, the total beat frequency is given by

$$f = 2\Delta f' = 2\left(\frac{\Delta P}{P}\right) f'$$

Finally, substituting for ΔP gives

$$f = \frac{4wAf'}{cS} = \frac{4wA}{\lambda S}$$

This equation gives the final derivation of the Sagnac effect [4].

C. HG1700 INTRODUCTION

The HG1700 Inertial Measurement Unit (IMU) is an RLG made by Honeywell International Incorporated. It is a six Degree-of-Freedom (DOF) system with the capability to measure acceleration, angular rotation rate, change in velocity, and change in angle, all in a three-dimensional coordinate reference frame. The IMU was originally made for the Joint Direct Attack Munition (JDAM) and the Wind Corrected Munitions Dispenser (WCMD), but it can be modified to function in other platforms. The following

chapters will discuss the configurations to set up the IMU, the software to run it, the tests done on the outputs to confirm their accuracy, and finally the steps taken to implement the system into the ARIES Unmanned Underwater Vehicle (UUV).

II. HARDWARE CONFIGURATIONS

A. HONEYWELL TEST BOX CONFIGURATION

The Honeywell test box contains all the necessary hardware components within the box to power and read transmissions from the IMU. A Honeywell card associated with the test box had to be installed in a CPU equipped with Windows95, and a special 30-pin connector sent from Honeywell had to connect the test box to the CPU. The test box is powered through a standard 110V outlet.

Figure 2. Test Box Hardware

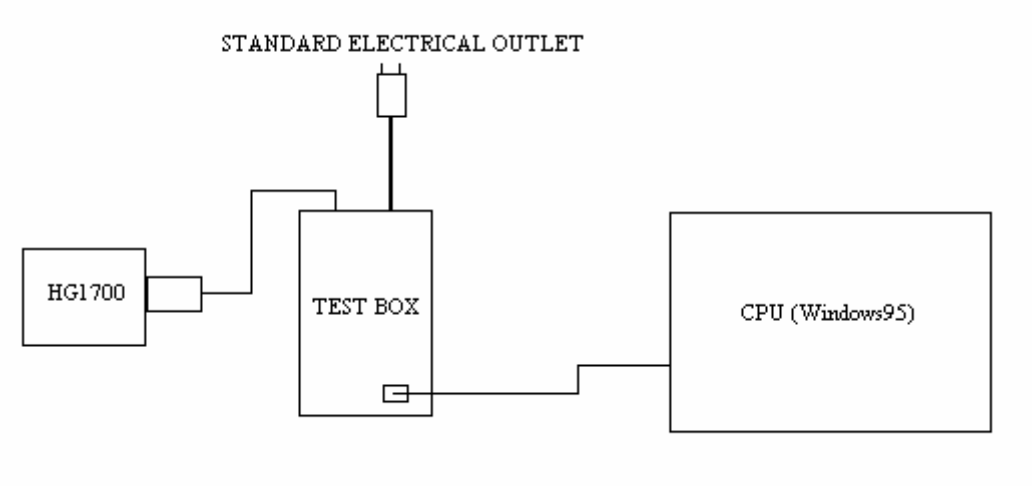

Figure 3. Test Box Block Diagram

B. C++ CONFIGURATION

The next configuration is the hardware configuration that will be used to run the IMU when it is installed on the ARIES vehicle. Throughout this research it will be referred to as simply the C++ configuration because a C++ program is used to process the transmissions from the IMU. An Ultralife battery set at 30V DC powers the IMU. The positive voltage from the battery is run through a 2A fuse (slow blow) before both the positive and negative voltages are run through a Datel DC/DC converter. The converter outputs +/-15V DC (1A) and +5V DC (5A) and a common ground. The outputs and ground from the converter run through a 6 pin Molex connector before attaching to the IMU. A HI and LO output from the IMU connect to a B & B Electronics 422-to-232 converter in order to match the interface of a computer serial port. Finally, a short ribbon cable (straight cable) connects from the 422-to-232 converter to the serial port.

Figure 4. C++ Hardware

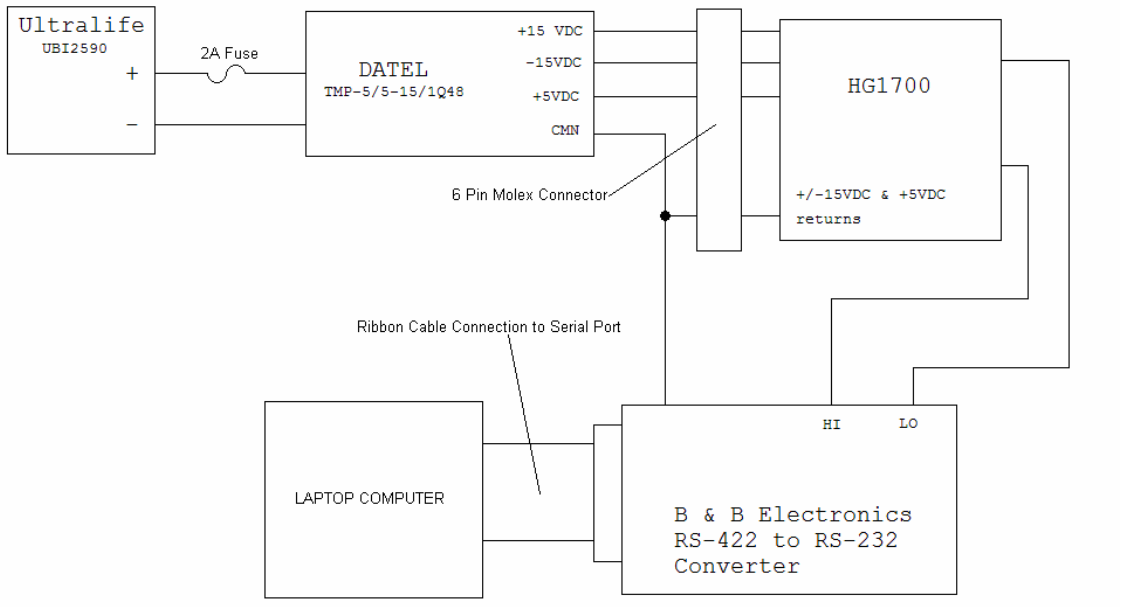

Figure 5. C++ Block Diagram

C. ORIENTATION OF IMU

Now that the two different setup configurations have been explained, it is appropriate to define the orientation of the IMUs reference frame. The IMU was tested and will be installed into the ARIES vehicle in the special orientation shown in the previous two sections. Given that setup, the positive x axis points straight ahead, the y axis points 90° to the left, and the z axis points straight up. The rotation rates follow the left hand rule where clockwise rotations are positive and counterclockwise rotations are negative. For the sake of standardization, the rotation about the x axis will be called roll, the rotation about the y axis will be called pitch, and the rotation about the z axis will be called yaw.

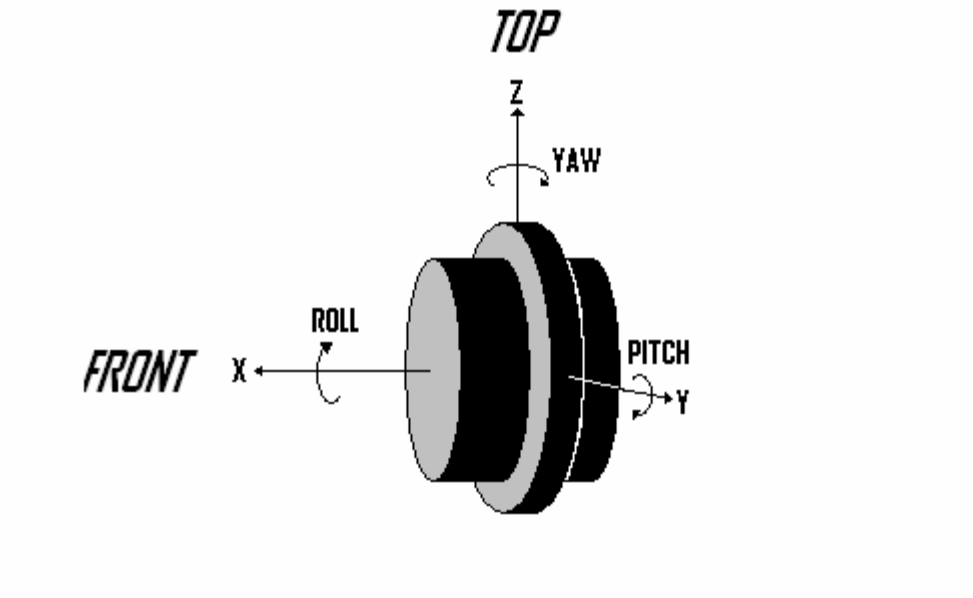

Figure 6. Illustrated Orientation of IMU

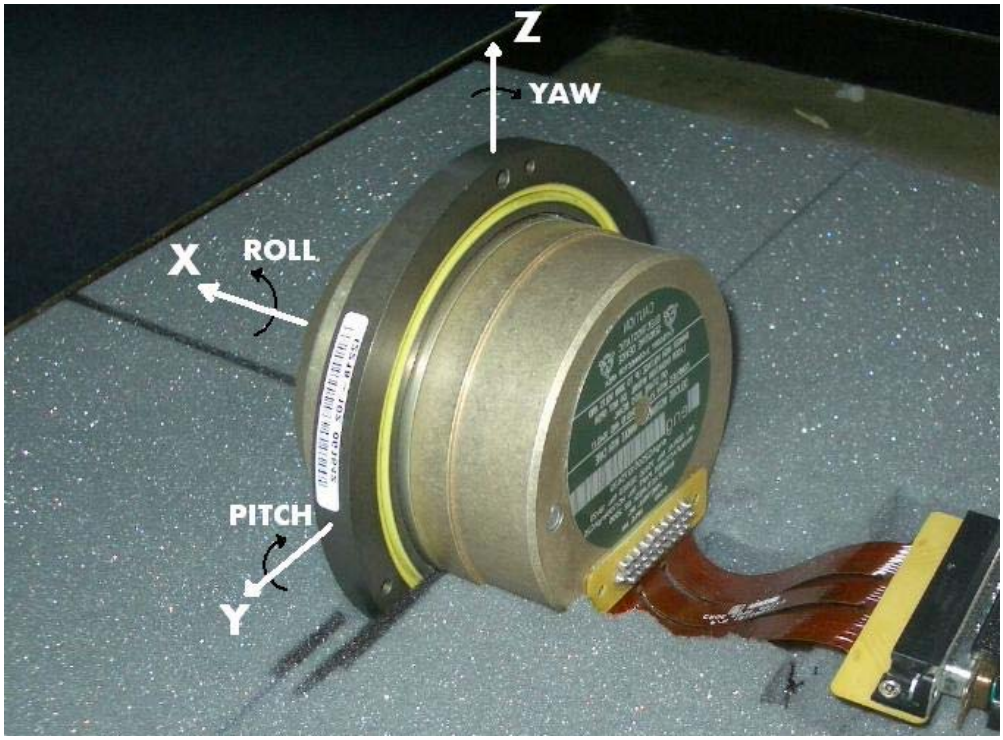

Figure 7. Photo Orientation of IMU

This reference frame orientation is modeled after the output from the text box configuration with one major modification: the y and z axes are swapped. In other words, the output from the test box shows the y axis as pointing straight up and the z axis as pointing 90° to the left. This swap is done easily in the MATLAB code by renaming the y and z vectors.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SOFTWARE USED TO TEST AND RUN THE HG1700 IMU

A. HG1700 OUTPUT

The HG1700 used in this research is an asynchronous model that outputs a single message at a rate of 100 Hz. The output can be broken down into two sections, where the first section contains all of the flight control data, and the second section is compromised of the status words and the inertial data.

1. Flight Control Data

The flight control data includes the angular rotation rates along the x, y, and z axes, and the linear accelerations in the x, y, and z directions. Each parameter output is composed of a two-byte word, making a total of 12 bytes for the entire flight control section, and it is measured at a rate of 600 Hz. Only every sixth flight control message is transmitted, however, due to the 100 Hz serial data interface. In order to get units of radians/second, the Least Significant Bits (LSBs) of the angular rotation rate parameters must be multiplied by a factor of $2^{-20} * 600$, and in order to get units of feet/second^2, the LSBs for the linear acceleration parameters must be multiplied by a factor of $2^{-14} * 600$ [5].

2. Status Words and Inertial Data

The second section includes two status words and the inertial data. The inertial data is made up of the change in angle along the x, y, and z axes, and the change in velocity in the x, y, and z directions. The first status word displays the accelerometer temperature in degrees Celsius as well as an IMU pass/fail test (outputted as a 0 or 1 respectively) in a two byte word, and the second

status word displays individual component pass/fail test results in a two byte word. Specifically, the second status word includes a processor test, a memory test, an accelerometer test, a gyro test, an "other" test, and it also outputs the software version number. The change in angle and change in velocity parameters are each composed of four bytes, making a total of 28 bytes for all of the second section. In order to get units of radians for the change in angle parameters, the LSBs must be multiplied by a factor of 2^{-33}, and in order to get units of feet/second for the change in velocity parameters, the LSBs must be multiplied by a factor of 2^{-27}. Unlike the flight control data, the status words and the inertial data are measured and transmitted at 100 Hz [5].

3. Summary of Output Message

Thus, summing up the components of the flight control data, the status words, and the inertial data, the entire message is made up of a 40-byte string. Before the message is sent, however, each transmission begins with a sync byte of A5h (or 165 in decimal format) followed by a message id byte of 2, and after each message a 2-byte checksum is transmitted. This means that 44 bytes are being transmitted at 100 Hz, or 4,400 bytes/second. Converting the bytes to bits (8 bits in a byte), the HG1700 transmits 35,200 bits/second. In order to receive all the data, the manual states a baud rate of 115.2 kHz set at 8 data bits, 1 stop bit, and no parity [5]. See the figures and tables below for a graphical summary of the HG1700 output.

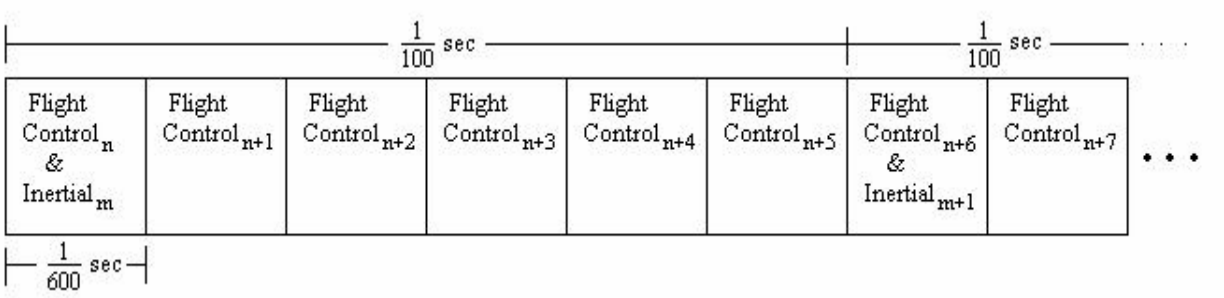

Figure 8. IMU Rate of Measurement and Transmission (After Ref. [5])

Parameter	Description			
Shift Clock	None - Asynchronous			
Data Communication	Asynchronous Mode: Baud Rate = 115KHz ±2% 8 Data bits 1 stop bit No Parity			
Data Ordering	Least Significant (LS) bit first; LS byte first; LS 16 bit word first			
Message Format	Message from Host to IMU	Message from IMU to Host		
	NONE USED	Description	Bytes	Value/Para.
		Sync Byte	1	A5h
		IMU Msg. ID	1	2
		Data	40	Per below
		Checksum	2	16-Bit

Table 1. Asynchronous Flight Control and Serial Data Interface Characteristics (From Ref. [5])

Item	Parameter	No. of Bytes	LSB Value	Units
1	Angular Rate Body X	2	$2^{-20} \times 600$	rad/sec
2	Angular Rate Body Y	2	$2^{-20} \times 600$	rad/sec
3	Angular Rate Body Z	2	$2^{-20} \times 600$	rad/sec
4	Linear Acceleration Body X	2	$2^{-14} \times 600$	ft/sec^2
5	Linear Acceleration Body Y	2	$2^{-14} \times 600$	ft/sec^2
6	Linear Acceleration Body Z	2	$2^{-14} \times 600$	ft/sec^2
7	Status Word 1[1]	2	N/A	N/A
8	Status Word 2[2]	2	N/A	N/A
9	Delta angle - body X *	4	2^{-33}	radians
10	Delta angle - body Y *	4	2^{-33}	radians
11	Delta angle - body Z *	4	2^{-33}	radians
12	Delta velocity - body X	4	2^{-27}	feet/sec
13	Delta velocity - body Y	4	2^{-27}	feet/sec
14	Delta velocity - body Z	4	2^{-27}	feet/sec

Notes

1) Status Word 1 is defined as follows:
   - Bit 15 (MSB) - 8 = Accelerometer (a) Temperature (LSB = 1°C)
   - Bit 7 = a-axis RLG in PLC reset (=1)
   - Bit 6 = b-axis RLG in PLC reset (=1)
   - Bit 5 = c-axis RLG in PLC reset (=1)
   - Bit 4 = IMU Failed (=1) (if set, remains during contiguous power)
   - Bit 3,2,1,0 = 4-bit Counter

2) Status Word 2 is defined as follows:
   - Bit 15 (MSB) = Processor Tests Failed
   - Bit 14 = Memory Tests Failed
   - Bit 13 = Other Tests Failed
   - Bit 12 = Accel Tests Failed
   - Bit 11 = Gyro Tests Failed
   - Bit 10 = Reserved
   - Bit 9 = Reserved
   - Bit 8 = Reserved
   - Bit 7-0 = Software Version Number

Table 2. Data Message Contents (From Ref. [5])

B. SOFTWARE USED TO READ HG1700 OUTPUT

Two different software packages were used to read and evaluate transmission data from the HG1700: a test box with the associated MS-DOS program sent from Honeywell and a C++ program written by the AUV lab.

1. Test Box Software

The Honeywell test box was used to confirm that the HG1700 was transmitting accurate data. A MS-DOS program, Menu_25.pif, was used to read, display, and save the transmitted data from the test box, and it can only be run on a Windows95 CPU. When running the program, the user must first select one of five options: exit program, scale outputs, display data to screen, record data to a file, or perform noise test. For all the testing done on the test

box in this research, the option to record data to a file was selected. Next, the user could chose to display the 600 Hz flight control data, but since the asynchronous model of the HG1700 only transmits at 100 Hz, this option was never selected. Finally, the user must enter the rate at which to display and save the data, create a file name with a *.dat extension to save the data, and enter a comment line on that file. After the comment line is entered, the program will begin gathering data.

If the 'scale outputs' option is selected, the user can change the gyro scale factor to either one (LSB), radians, radians/second, degrees, degrees/second, or degrees/hour, and the user can change the acceleration scale factor to one (LSB), feet/second^2, g's, or meters/second^2. For all of the testing done with the test box, the gyro scale factor was set to degrees/hour, and the acceleration scale factor was set to g's.

The textbox worked great for short periods of testing, but it consistently froze up after approximately two minutes or less regardless of the rate at which the data was displayed. Thus, the test box was useful for testing the angular measurements in a short rotation test, but it was impossible to do any long-term tests with the test box to determine the error bias in the drift rate. The actual test results for tests done with the test box will be discussed in the next chapter.

2. C++ Software

The C++ program, DECODE4, was written only as a test program to read saved data from a text file and output the results to the screen and save them to a new text file. A terminal emulation program such as Procomm Plus or

Hyperterminal reads in the binary data from the HG1700 and saves it in ASCII format to a file. DECODE4 then reads the saved file one ASCII character at a time (or one byte at a time) searching for the sync byte followed directly by the message id. Once it finds the 165-2 pair, the program takes the following 40 bytes and combines them into words to get the correct parameters in exactly the same way as discussed in II.A.1 and II.A.2 above. After completely evaluating and displaying one message, the program begins looking for the next 165-2 pair and repeats the entire process. This code can be found in the appendix.

DECODE4 will be modified and used to actually run the IMU in the ARIES vehicle. The primary modification will be to read the transmission from the HG1700 directly into the program instead of reading the data from a saved file. When making this modification however, the program must also include a command to clear the buffer after each transmission in order to prevent the buffer from becoming overloaded. When the buffer can no longer hold all the transmitted data, data will be lost, and even the loss of one byte in a transmission will greatly skew the outputted results. This very problem occurred during the testing in the research and will be covered in more detail in the next chapter.

IV. TESTING ON THE HG1700 IMU

A. EXPLANATION OF TESTS

Once installed, the HG1700 IMU will be used primarily for determining initial compass heading on the ARIES vehicle as well as tracking the heading of the vehicle throughout its mission. Therefore, to determine the accuracy of each of these uses, two separates tests needed to be run and evaluated: a rotation test and an idle test. As an added check, a third test was done to track the IMU's position based on the measured accelerations and angular rotation rates, and these results will be discussed as well.

B. ROTATION TEST

The rotation test was done using the foam mount on a flat table with the test box and Honeywell software as discussed and shown in chapter II. To reiterate what was stated in that chapter, in this setting (which is the orientation in which the IMU will be mounted in the ARIES vehicle), the positive x axis points directly ahead, the y axis points 90° to the left, and the z axis points straight up. With this configuration, the goal of the rotation test was to rotate the IMU about its z axis 90° to the right (+90°), rotate it back to the 0° position, and then repeat the processes in the opposite direction (to -90° and back to 0°).

1. MATLAB Programming for Rotation Test

After gathering and saving the data from the test box, the data output file was loaded into the MATLAB program IMUtest_rotation.m. This m.file loads the data file and first plots the acceleration along the three axes with

respect to time. This graph is simply a check to insure that the z axis is reading a -1 g and the x and y axes are reading approximately zero acceleration. Next, the program plots the x, y, and z rotation vectors with respect to time. Since the rotation vectors were outputted from the IMU test box in degrees/hour, it is difficult to determine any specific results from this plot. Therefore, the program uses a discrete integration method to integrate the degrees/hour data with respect to time in order to achieve the angle of rotation at each time step. The equation for the discrete integration is shown below:

$$rotation\_value_t(\deg rees) = \sum_{t=0}^{t=n} rotation\_rate_t \times \Delta t$$

where $\Delta t = \frac{1}{measured\_rate}$. The measuring rate for this test was 10Hz. Thus, integrating the x, y, and z rotation vectors with respect to time results in the roll, pitch, and yaw respectively, and these values are plotted versus time in the last plot of the program. IMUtest_rotation.m can be seen in the appendix.

2. MATLAB Plots for Rotation Test

The following are the three plots in order as discussed in the section above.

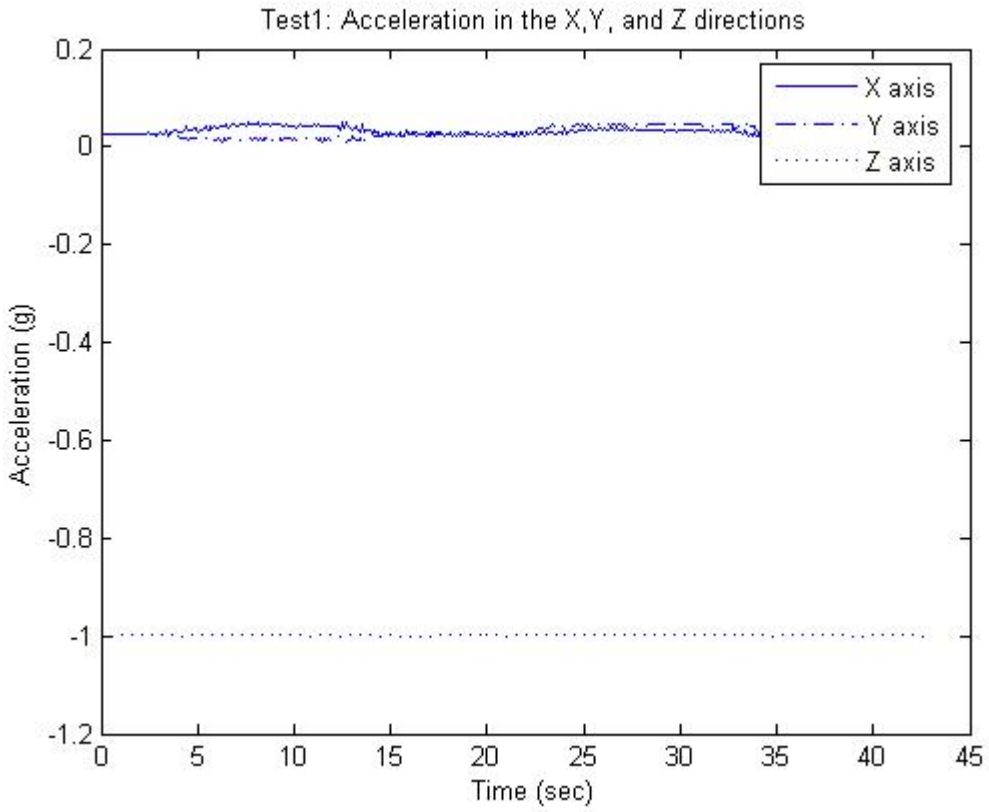

Figure 9. Rotation Test: Accelerations vs. Time

Figure 9 confirms an accurate acceleration reading since the acceleration in the z direction reads -1 g and the accelerations in the x and y directions read basically 0 g's.

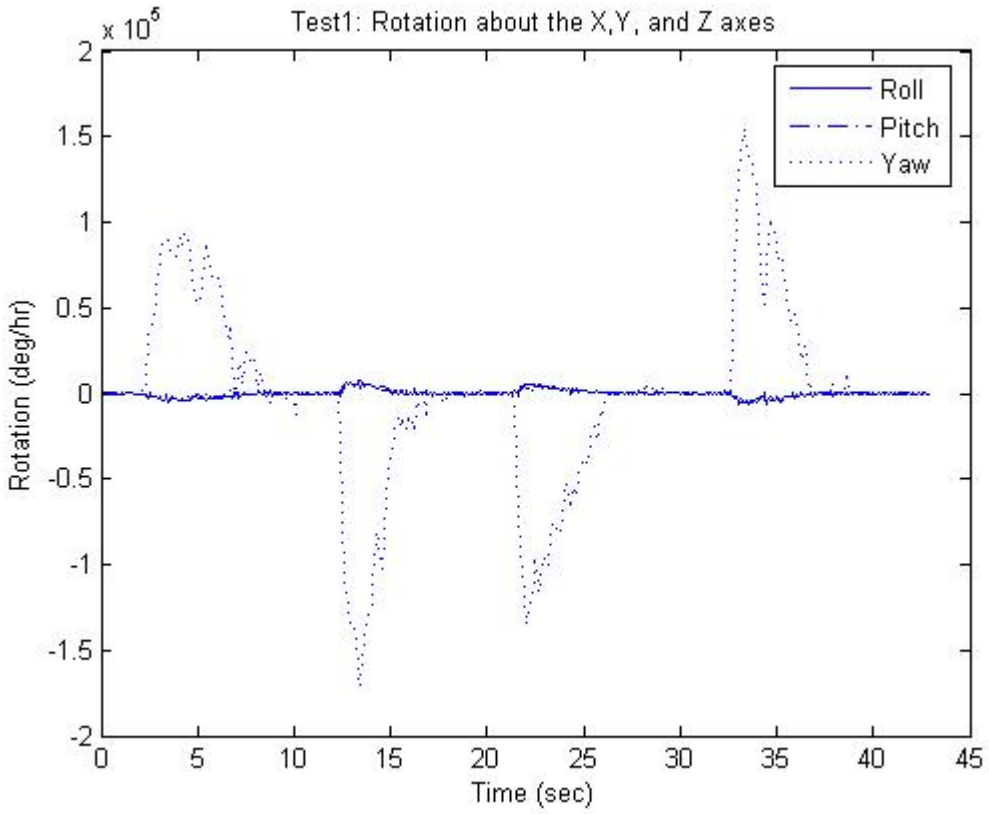

Figure 10. Rotation Test: Rotation Rates vs. Time

As discussed above, Figure 10 is difficult to interpret precisely, but it is clear that there is first a positive rotation rate about the z axis (clockwise), then a negative rotation rate (counterclockwise), then another negative rotation rate, and finally a positive rotation rotate. These are the rotation rates expected for this test.

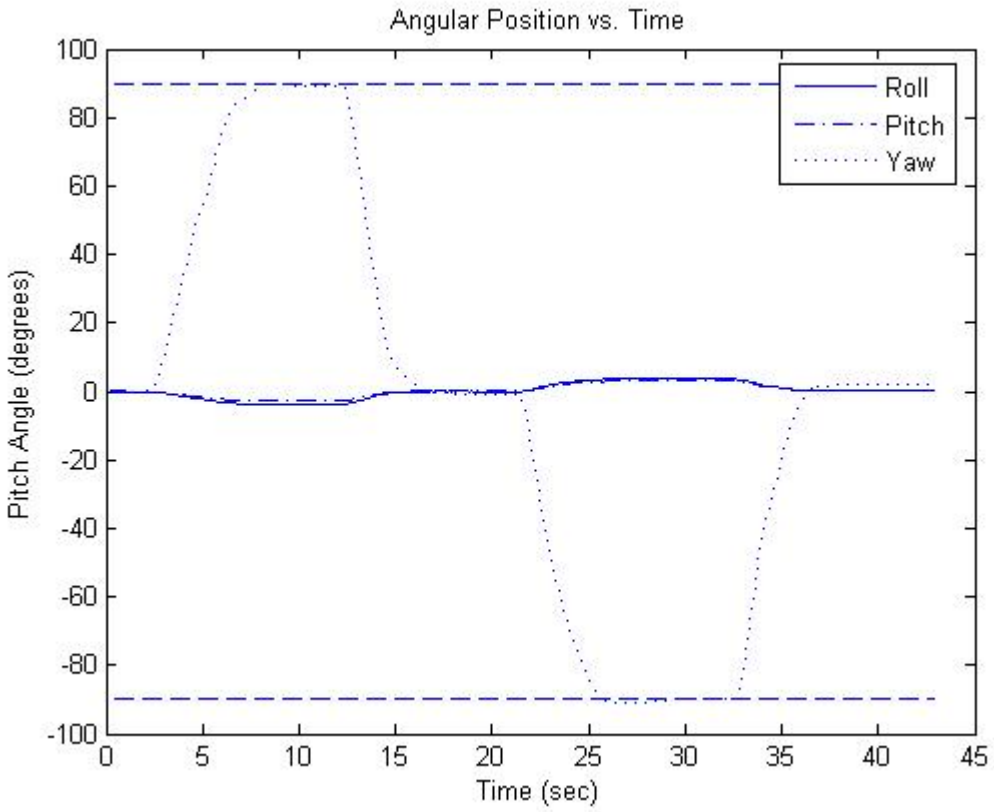

Figure 11. Rotation Test: Angular Position vs. Time

As can be seen in Figure 11, the yaw does in fact rotate up to 90°, back to 0°, down to -90°, and then back to 0°. Based on this test we know that the output of the IMU can be integrated to accurately display the heading of the vehicle as it is carrying out its mission.

C. IDLE TEST

The second test, the idle test, proved to be much more difficult of a test to complete. The purpose of the idle test was to let the IMU sit motionless and run for a certain period of time in order to determine and factor out the drift rate caused by the rotation of the earth and ultimately determine the error bias associated with the HG1700 itself. The initial heading of the IMU could be calculated from the idle test. In a tactical mission, the

idle test will be used solely to determine initial heading since the earth rate and error bias will have been tested and factored out of the measurements. For testing purposes, however, the idle test was used to measure all three: earth rotation rate, error bias, and initial heading.

The idle test was set up exactly the same as the rotation test with the IMU sitting in the foam mount on a flat table.

1. Calculating the Earth Rotation Rate

The rotation rate associated with the spinning of the earth is equal to 360 degrees/day or 15 degrees/hour counterclockwise, and since the IMU reads counterclockwise rotations as negative, the rotation of the earth is -15 degrees/hour. When the IMU is run for a period of time while sitting idle, the only changes to the outputted data should be due to this earth drift rate and the error bias. Thus, by knowing and factoring out the earth drift rate, the error bias rate can be determined.

In order to find and filter out the earth drift rate, the x, y and z axes must be defined in relation to the axis of rotation of the earth. If the IMU were set facing north or south directly on the equator, the earth drift rate would appear only in the roll vector because both the pitch vector and the yaw vector would be perpendicular to the axis of rotation (only the x axis would be rotating). But when the IMU is operating at any other latitude or any other orientation, all three of its axes have component vectors parallel to the axis of rotation. These drift rate components can easily be found if the latitude and compass heading are known. For simplicity, assume that the IMU is facing due north (compass heading 0.00°), but at a

different latitude than the equator. This setup is illustrated in Figure 12. (The x axis points north (N). The y axis points west (W), and the z axis points up from the center of the earth (U).) The angle of latitude is labeled λ.

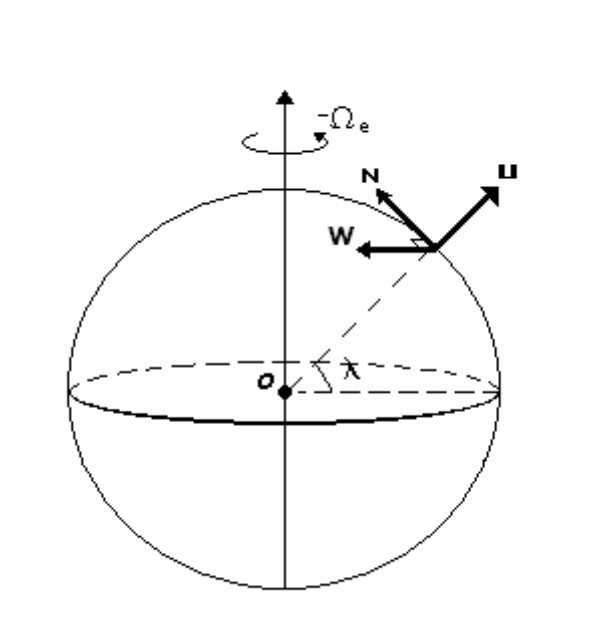

Figure 12. Earth Rotation

This configuration can be reduced trigonometrically to a two-dimensional figure (Figure 13), where the axis of rotation is shifted to the origin of the IMU, the angle of latitude has been transferred by trigonometric identities, and the z axis has been shifted in order to complete the N and U components of the axis of rotation.

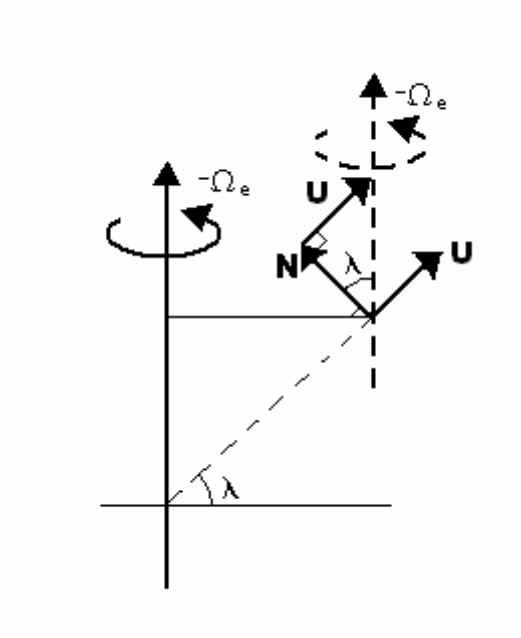

Figure 13. Shifted Earth Rotation Axis

From this figure, the following rotation equations can be deduced:

$$U = -\Omega_e \sin\lambda$$
$$N = -\Omega_e \cos\lambda$$

Now assume that the IMU is at the same latitude as before, but the heading is rotated to some angle other than due north. This assumption is illustrated in Figure 14, where the x and y axes are rotated clockwise (positive) from due north and the z axis points straight up.

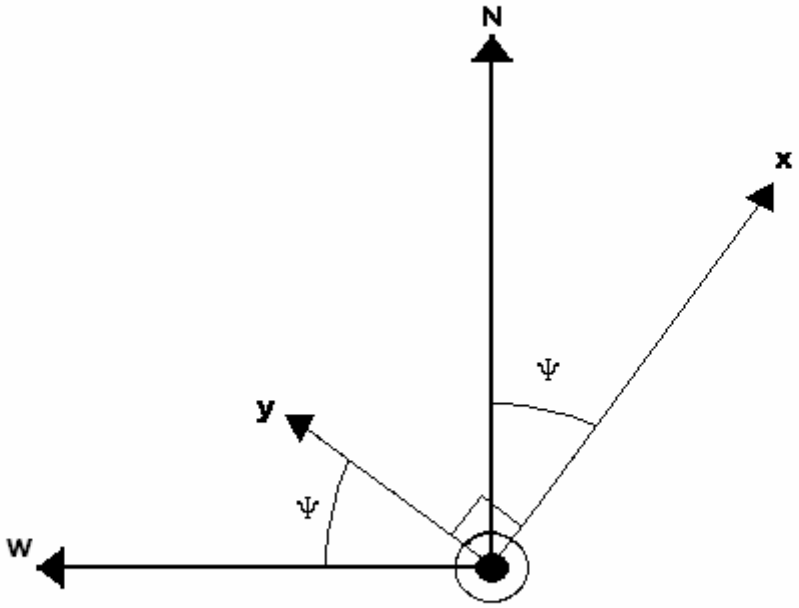

Figure 14. X/Y Axes Rotated From N/W Directions

From this figure, the rotation equation $N = -\Omega_e \cos\lambda$ must be redefined in x and y components where

$$\varpi_x = N\cos\psi$$
$$\varpi_x = -\Omega_e \cos\lambda\cos\psi$$

and

$$\varpi_y = N\sin\psi$$
$$\varpi_y = -\Omega_e \cos\lambda\sin\psi$$

Thus, in summary, the equations for the rotation of the earth measured about each axis are

$$\varpi_x = -\Omega_e \cos\lambda\cos\psi$$
$$\varpi_y = -\Omega_e \cos\lambda\sin\psi$$
$$\varpi_z = -\Omega_e \sin\lambda$$

2. MATLAB Programming for Idle Test

A MATLAB script file was written to display the average rotation rate about each axis in degrees/hour during the idle test compared to the theoretical earth rotation rates about each axis based on the derived equations above. The program, IMUtest_idle##.m, (## represents the test run number) outputs eight columns of data related to the experimental and calculated earth rotation rates. The first 3 columns of output are the experimental data gathered from the HG1700, and columns 4, 5, and 6 are the calculated values using the equations derived above with a latitude of 36.5859°, a compass heading of 0.00°, and an earth rotation rate of (-15.00 degrees/hour). The last two columns related to the earth rotation rate are the total experimental and calculated rotation rates, respectively, found by using the three dimensional application of Pythagoras' theorem:

$$\omega_t = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$$

After producing the eight columns related to the earth's rotation rate, the program outputs two more columns related to the initial heading. Looking again at Figure 14, the heading angle ψ can be found by taking the arctangent of ω_y/ω_x. Thus, the first of these two columns is the initial heading calculated from the experimental data, and the second column is the initial heading calculated using the calculated data. IMUtest_idle##.m can be seen in the appendix.

All of the idle tests were done using the Honeywell test box, but, as discussed in chapter III, the test box software had a data storage problem making it impossible to

do any long-term testing with it. Seven idle tests were done with the test box, each at a different sampling rate, in order to try to maximize storage capacity. The results are shown below.

```
IMUtest_idle4
TEST 4: 1 Hz Data Rate. Duration - 70sec
   wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
  -9.3419   -2.9024   -9.8148  -12.0439         0   -8.9412   13.8574   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Experiment      True
 -162.7406         0



IMUtest_idle5
TEST 5: 10 Hz Data Rate. Duration - 102sec
  wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
-10.6108   -5.9588   -8.8306  -12.0439         0   -8.9412   15.0358   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Experiment      True
   29.3176         0



IMUtest_idle6
TEST 6: 100 Hz Data Rate. Duration - 69.1sec
  wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
-11.4164   -2.1896   -9.9025  -12.0439         0   -8.9412   15.2705   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Experiment      True
   10.8571         0



IMUtest_idle7
TEST 7: 1 Hz Data Rate. Duration - 70sec
  wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
-11.2792   -3.2644   -8.5668  -12.0439         0   -8.9412   14.5350   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Experiment      True
   16.1413         0



IMUtest_idle8
TEST 8: .2 Hz Data Rate. Duration - 75sec
  wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
-10.5452   -3.7608   -8.8667  -12.0439         0   -8.9412   14.2815   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Experiment      True
   19.6279         0
```

```
IMUtest_idle9
TEST 9: 10 Hz Data Rate. Duration - 48.2sec
  wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
-12.2331   -3.7360   -9.7339  -12.0439         0   -8.9412   16.0734   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Experiment      True
   16.9830         0


IMUtest_idle10
TEST 10: 20 Hz Data Rate. Duration - 15.95sec
  wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal    wt_exp    wt_cal
-10.3930   -1.0867   -8.7919  -12.0439         0   -8.9412   13.6562   15.0000

    Computed Heading by taking ATAN of Wy/Wx
   Exp (deg)       True (deg)
    5.9690         0
```

Table 3. MATLAB Output From Idle Tests

3. Conclusions for Idle Tests

According to manufacture specifications, the error drift rate of the HG1700 should be less than 1 degrees/hour, meaning that the experimental rotation rates from the seven tests should range between 14.9° and 15.1° [5]. From the seven tests, however, the total experimental rotation rates range from 13.6562° to 16.0734° with a standard deviation of 0.8486. Thus these seven tests prove that over a short period of time, the HG1700 fails to meet the expected specifications. This outcome was expected, however, since the output from the IMU must be measured over a long period of time (~30 minutes) in order to accurately calculate the drift rate. Therefore, a series of longer idle tests must be done.

Another important point to notice in the short idle tests is the headings calculated from the experimental data. The heading angle should read approximately 0.00°, but in fact it ranges from -162.7406° to 29.3176°. This is

further proof that enough data cannot be gathered in two minutes in order to accurately calculate the error drift rate or the heading.

The C++ program DECODE4 should have provided the capabilities to perform long-term tests with the IMU, but a data acquisition problem prevented the program from ever being able to read and display accurate information. By analyzing the capture files from whichever terminal emulation program was used, it was revealed that the sync byte and the message id (165, 2) were not showing up consistently in the data string. This seems to indicate that at certain times the buffer was overwhelmed by bursts of data resulting in the terminal emulation program not being able to capture all that data [6]. This information is good in the sense that the setup of the hardware components is correct, but it also means that more work needs to be done to make the software compatible at the serial data port interface.

D. POSITIONAL TESTING

The rotation test proved that it is possible to integrate the rate of rotation in order to get angle of rotation, and since the IMU also measures the acceleration along each axis, it would seem that by double integrating the acceleration components in a constant frame of reference, the position of the IMU could be recorded.

The positional testing involved moving the IMU along four different routes and running the MATLAB program IMUtest_position.m to plot the position of each track. The four routes were as follows: a complete circle with a radius of approximately two feet, a complete square with the length of each side being approximately four feet, a

ten foot long straight line parallel to the x axis of the constant reference frame, and a ten foot line that drifted two feet at a constant rate off the x axis of the constant reference frame while the IMU remained facing the constant reference frame.

In order to derive the equations for this test, the concept of a constant frame of reference must be explained. When the IMU is first initialized, it is placed in a certain direction and orientation. This configuration will be called the initial reference frame. As the IMU is moved and/or rotated, however, the reference frame associated with the IMU will deviate from the initial reference frame, but the IMU's reference frame can always be translated back to the initial reference frame by using the rotation and acceleration data gathered throughout the run. When the IMU's reference frame is translated back to the initial reference frame, the movement of the IMU in that reference frame can be tracked. Thus, in order to track the position of the IMU, the IMU's reference frame must be translated back to the initial reference frame. This idea of tracking the position with respect to the initial reference frame is know as using a constant frame of reference. The equations and diagrams below will explain how to translate the IMU's reference frame back to the initial reference frame.

1. Derivations for Calculating Position

To track position, the only parameters of importance are the accelerations along the x, y, and z axes of the constant reference frame because these accelerations can be double integrated with respect to time to produce position. In order to derive these accelerations, the accelerations

associated with the IMU's reference frame must be translated back to the constant reference frame using simple trigonometry.

Consider the two-dimensional figures below:

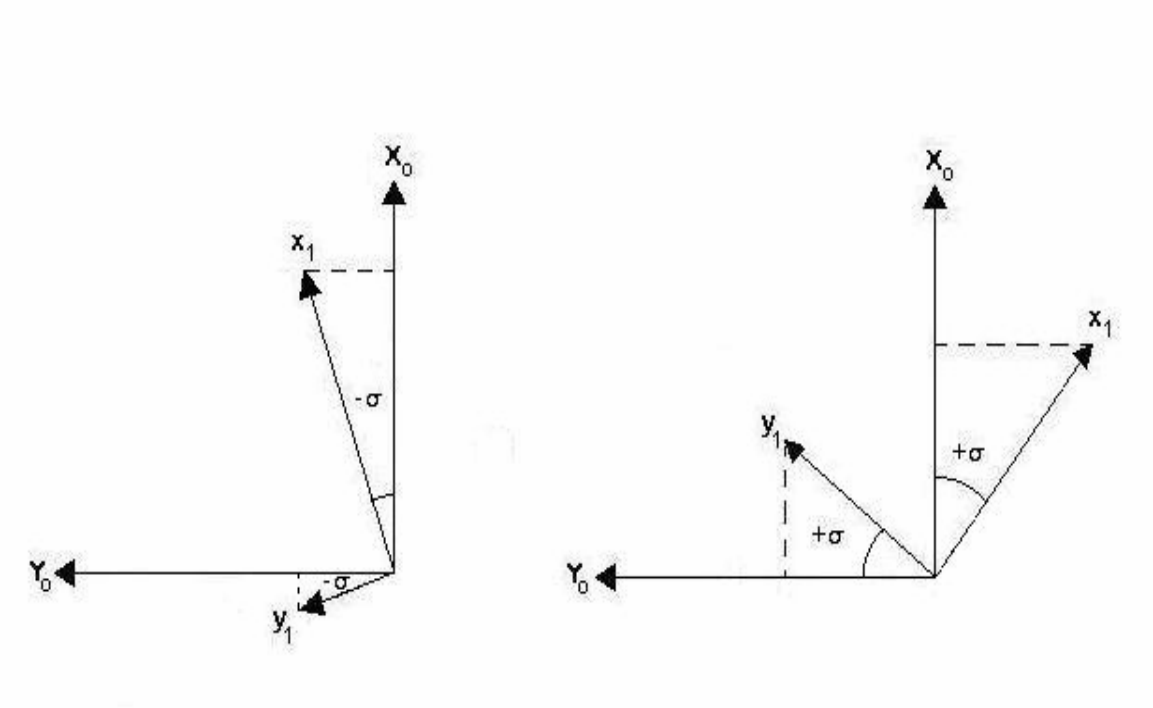

Figure 15. Translation to Reference Frame

Looking at the first figure, the angle of rotation σ is marked negative because of the counterclockwise rotation about the z axis of the IMU. If x_1 and y_1 are acceleration vectors in this figure, and all of the vectors point in the positive direction, then the components of x_1 and y_1 can be related back to x_o and y_o by the equations

$$accel(x_0) = accel(x_1)\cos(-\sigma) + accel(y_1)\sin(-\sigma)$$
$$accel(y_0) = accel(y_1)\cos(-\sigma) - accel(x_1)\sin(-\sigma)$$

Likewise, in the second figure the angle of rotation σ is positive since the rotation is in the clockwise direction. Because of the sign change on the angle, the equations to

relate x_1 and y_1 back to x_o and y_o are exactly the same. And since all of the positioning tests were done in the x-y plane (no change in z), these two equations will suffice.

For situations in which the IMU is rotated about more than one axis, the rotation rates must be converted to Euler rotation rates before being integrated into angles (Euler angles). Even though the positioning tests only involved rotations about the z axis, the method of calculating the Euler angles was used to ensure that the correct roll, pitch, and yaw angles were being recorded and to simply become familiar with the method since it must be used when the IMU is implemented into the ARIES vehicle. The rotation rates transmitted from the IMU can be converted to Euler rotation rates by the transformation matrix

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

where p, q, and r are the transmitted rotation rates about the x, y, and z axes respectively, and ϕ, θ, and ψ are the roll, pitch and yaw respectively [7]. The specifics as to how this transformation matrix must be implemented into MATLAB will be discussed later in this section.

Once the acceleration components have been translated to the constant reference frame, the acceleration along the x and y axes must be double integrated to get position along these axes. Integrating once yields

$$\int a\,dt = at + c_1 = at + v_0 = v$$

and a second integration yields

$$\int v dt = vt + c_1 = vt + x_0 = x$$

Now, converting the integral to a discrete integration form,

$$v(n) = \sum_{i=1}^{i=n} a_{i-1}\Delta t + v_{i-1}$$

and

$$x(n) = \sum_{i=1}^{i=n} v_{i-1}\Delta t + x_{i-1}$$

These derivations provide enough background to now sufficiently discuss the MATLAB programming results.

2. MATLAB Programming for Positioning Tests

The equations derived above were used in the program IMUtest_position.m to first derive the Euler angles, then calculate the acceleration vectors in the constant frame of reference, and finally integrate the acceleration vectors twice to get an x and y position at each time interval. Specifically, Figure 16 below shows the loop used to calculate the Euler angles at each time interval.

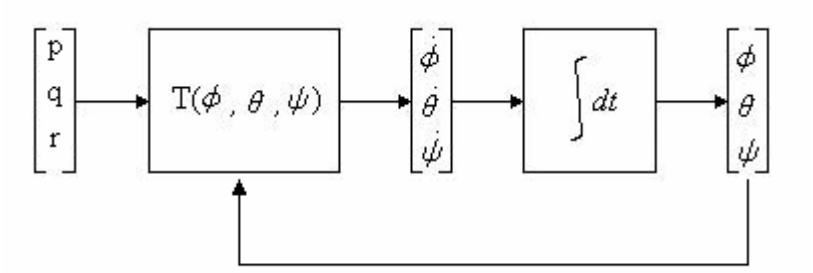

Figure 16. Euler Angle Loop

The rotation rates from the IMU are multiplied by the transformation matrix to get the Euler rotation rates. These rotation rates are then integrated to get the Euler angles, and the Euler angles are fed back into the

transformation matrix in order to calculate the Euler rotation rates at the next time interval. This loop is similar to the loop that will be used when the IMU is implemented into the ARIES vehicle, with the only difference being that the rotation of the earth will be factored out of the initial rotation rates before they are multiplied by the transformation matrix. Since all of these position tests were less than one minute in duration, however, it was not necessary to factor out the rotation of the earth. After the Euler angles have all been calculated, the program then translates the acceleration vectors back to the constant reference frame and double integrates to get position.

3. MATLAB Plots for Positioning Tests

IMUtest_position.m displays two plots per route. The first plot is a plot of the roll, pitch, and yaw versus time in order to confirm that the Euler angles were calculated correctly. The second plot is the primary plot of interest for this test: the actual track of the IMU as integrated from the accelerometers. These plots are shown in the figures below. IMUtest_position.m is given in the appendix.

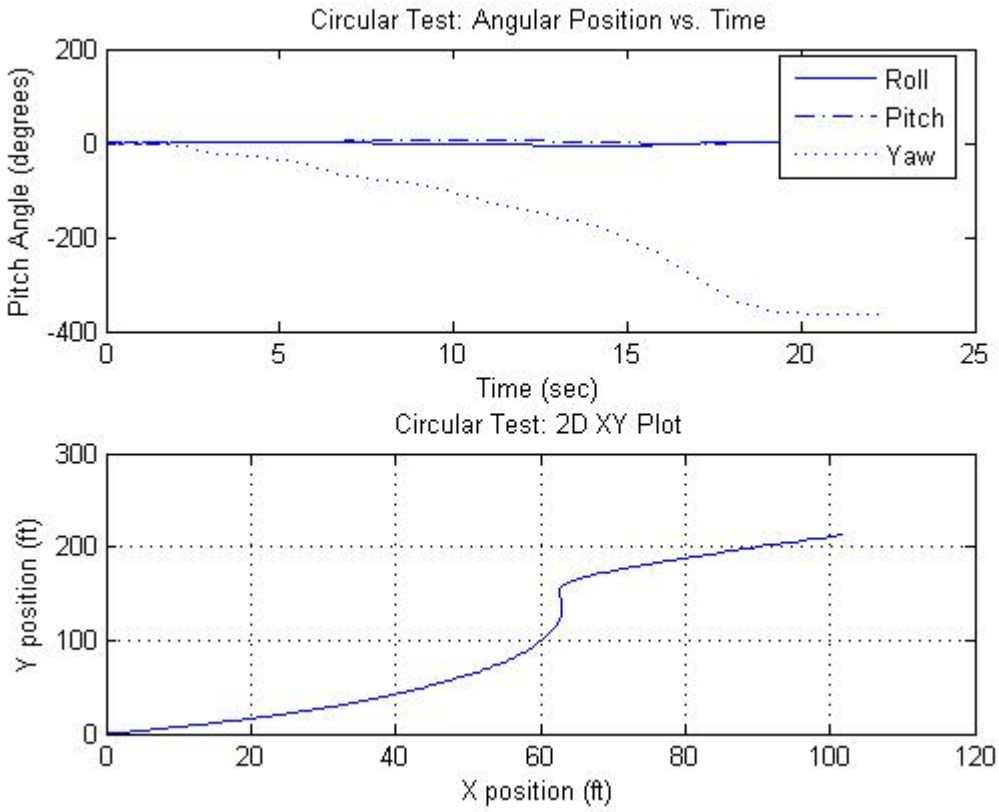

Figure 17. Positioning Tests: Circular Route

The circular test shows that the Euler angles have been calculated correctly (Figure 17, upper panel). The roll and pitch remain at approximately 0° while the yaw changes gradually from 0° to -360° indicating a circular pattern. However, the track obtained from double integration of the accelerometers deviates greatly from a circular track (Figure 17, lower panel). It appears that the IMU failed to measure accurate acceleration values throughout the route.

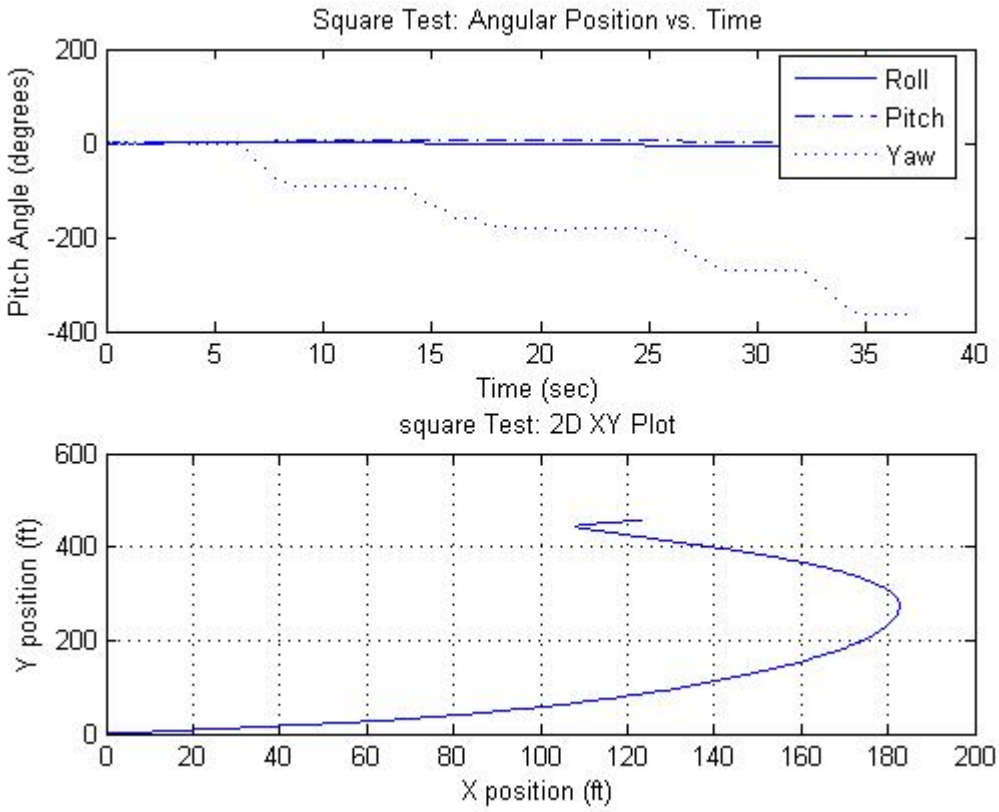

Figure 18. Positioning Test: Square Route

The plots from the square test (Figure 18) show similar results. Again, the square test shows an accurate output of Euler angles. The roll and pitch stay at approximately 0° while the yaw decreases in a stair step pattern from 0° to -360° indicating a square pattern. Like the circular test, though, the square test failed to plot an accurate route.

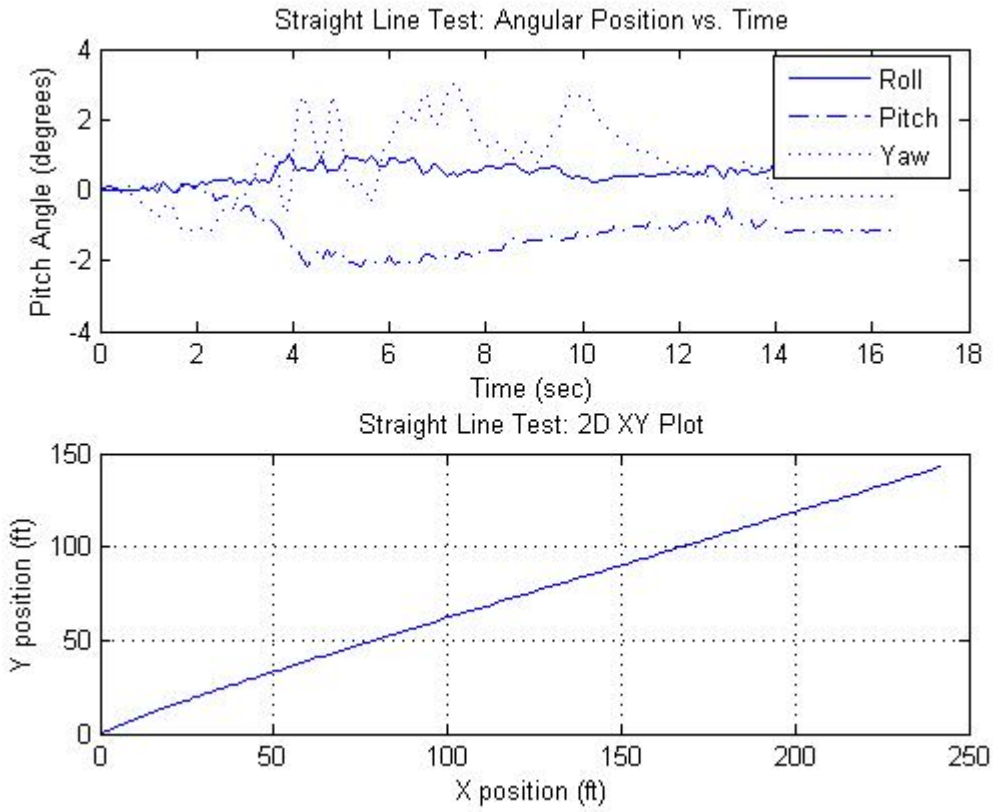

Figure 19. Positioning Tests: Straight Route

The straight line test shows very little change in the roll, pitch, and yaw, as expected, and, consistent with the previous tests, the position plot in greatly skewed (Figure 19).

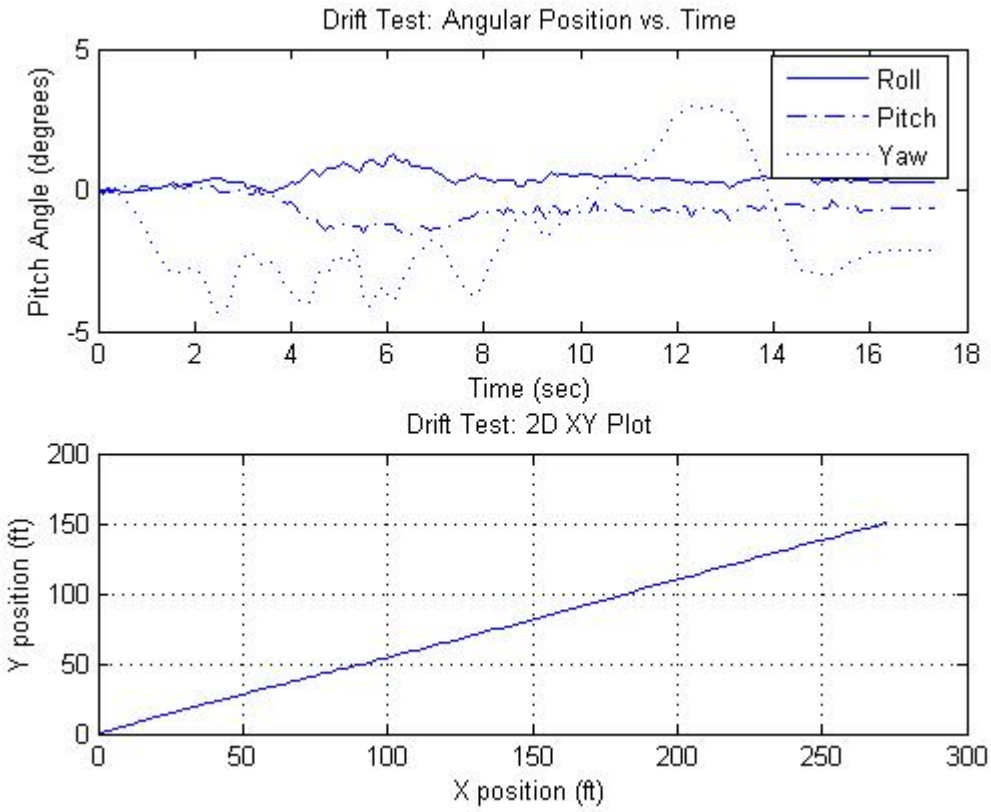

Figure 20. Positioning Tests: Drift Route

The drift test was designed to see if the IMU could measure the effects of the ocean currents during a mission. Based on the results shown in Figure 20, however, it is clear that such a capability does not exist. The drift test displays the same results as the other tests: accurate Euler angles with an extremely inaccurate position plot.

4. Conclusions for Positioning Tests

Based on the results from these four figures, it is obvious that the HG1700 cannot be used to track its position at very low accelerations. The reason for this is that first, each of the accelerometers in the IMU has an error bias associated with it. When you double integrate the acceleration to get position, the error bias integrates as well, causing the error in the accelerometers to grow at

an exponential rate over time. Thus in a very short amount of time, the error bias begins to dominate the data. Second, when translating the acceleration vectors to a constant frame of reference, the translated vectors are often smaller than the noise in the vectors, causing a very wide standard deviation for a very small acceleration value. Third, if the x and y axes of the IMU are not oriented exactly perpendicular to gravity, components of the acceleration due to gravity will appear in the x and/or y acceleration vectors. All these reasons combine to explain why the IMU at low accelerations fails to produce accurate positioning data.

THIS PAGE INTENTIONALLY LEFT BLANK

V. PLANS FOR IMPLEMENTATION

When the HG1700 is implemented into the ARIES vehicle, it should incorporate the same filter used in the Small AUV Navigation System (SANS) software. The SANS requires a GPS/DGPS receiver, IMU, compass, water speed sensor, water depth sensor, and a data processing computer - all of which are found on the ARIES vehicle. The current SANS navigation software uses the twelve-state complementary filter shown in Figure 21. The twelve state variables are as follows: the outputs of the three integrator blocks, estimated current in both the north and east directions, and the error bias estimates for the angular rate readings. A more detailed list of these state variables is shown in Table 4. Note also that in Figure 21, $R(\phi,\theta,\psi)$ is a rotation matrix and $T(\phi,\theta,\psi)$ is an Euler transformation matrix [8].

Euler Angles	ϕ, θ, ψ
North & East Velocity	$\dot{x}_e$, $\dot{y}_e$
North & East Position	x_e, y_e
Apparent Current	$\dot{x}_c$, $\dot{y}_c$
Angular Rate Bias Estimates	p_b, q_b, r_b

Table 4. State Variables (From Ref. [8])

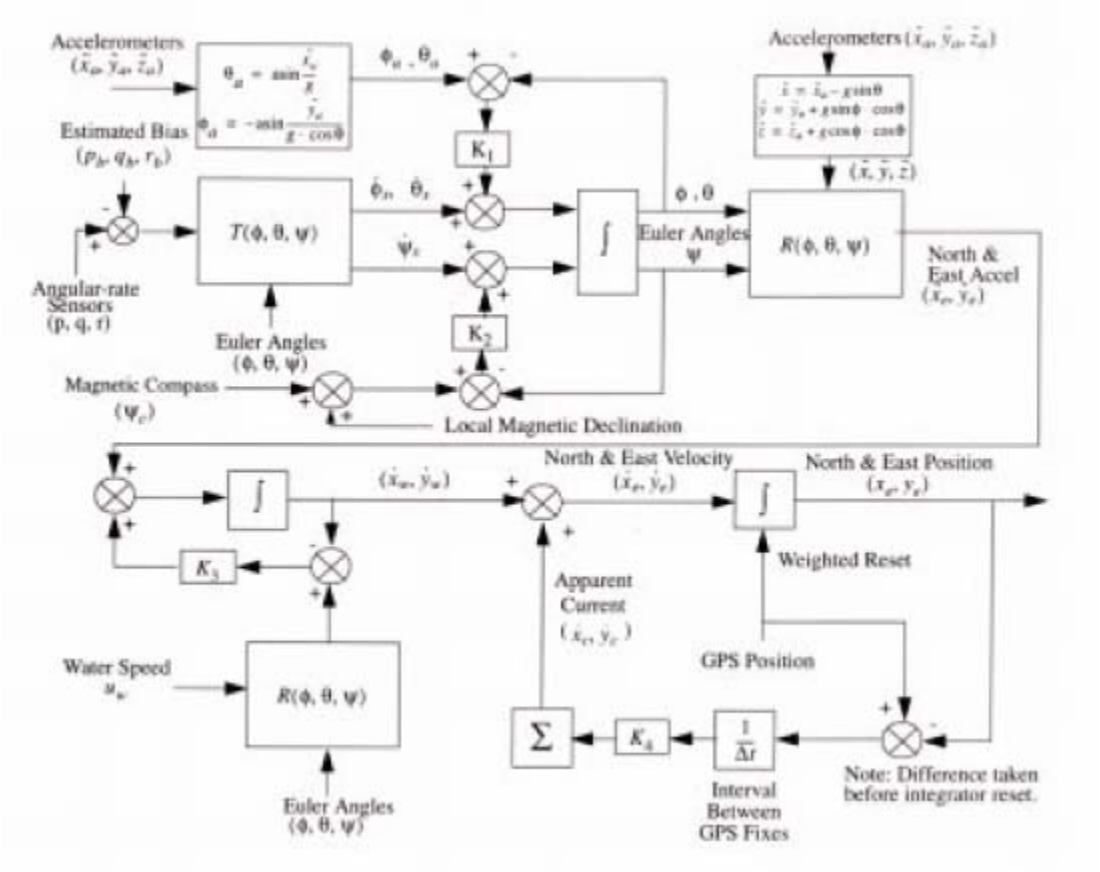

Figure 21. SANS Navigation Software Filter (From Ref. [8])

The filter gains K_1, K_2, K_3, and K_4 and constant. They must initially be determined by bandwidth considerations, but they can be modified and corrected afterwards by experimental tuning. For more information on how to tune the filter gains, see [8]. The only change needed to be made to the SANS software is to filter out the rotation of the Earth from the Angular-rate Sensors (p, q, r) using the equations discussed in section IV.C.1.

VI. CONCLUSION

It was the objective of this thesis to evaluate the accuracy of the HG1700 in order to determine its usefulness as part of the navigational components in the ARIES vehicle, and that goal was met in multiple categories. First, this thesis allowed for the two different setup configurations associated with the IMU to be examined. Second, through the rotation test, the idle test, and the positioning test, the inertial data from the IMU was evaluated for accuracy and consistency. And finally, this thesis showed the need for specific further evaluation, and the necessary steps to be taken when implementing the IMU into the UUV. While not everything is this thesis can be called successful, it certainly was useful and should be viewed as a necessary stepping stone in the process of implementing the HG1700 into the ARIES vehicle.

Throughout all the testing done on the IMU, the capabilities and difficulties related to the test box and C++ setup were examined. The test box proved useful for testing the IMU, only because the C++ setup never worked properly, but it still had two major limitations. First, the test box could only run on a Windows95 CPU, which prevented the mobility of testing on a laptop computer. And second, the test box software could not store more than two minutes worth of data regardless of the rate at which the data was displayed. This fact was shown especially true in the idle tests. The C++ software and setup has worked in the past, but due to problems with the serial data acquisition it never displayed accurate data during the testing time period for this research. Before the IMU can

be implemented into the vehicle, this problem must be corrected because long term testing must still be done and also because a similar setup and software will be used for the IMU when it operates on the ARIES vehicle.

In the rotation test, the accuracy of the rotation rates was tested and the ability to be able to integrate those rotation rates to get angles was evaluated. This test proved successful in that the IMU measured accurate rotation rates and those rates were integrated to reflect accurate angle measurements. Therefore, the IMU can and certainly will be used to measure the heading of the ARIES vehicle once it is installed. It must be remembered, however, that when the IMU is implemented into the vehicle, the earth rates must be factored out of the measured rotation rates, and then those rotation rates must be transformed to Euler rates before integrating into angle measurements.

In the idle test, the measured earth rates and error bias were examined. These tests had limited success due to the memory storage error associated with the test box setup and the data acquisition problems associated with the C++ setup. The short term testing done with the test box proved that neither the Earth rate, nor the error bias, nor the compass heading could be accurately measured in a short period time. Thus, the limited results from these tests necessitate a long term test in order to factor out an accurate earth rate measurement and to calculate the inherent error bias associated with the IMU. The accuracy of the calculated compass heading must also be checked in a long term test.

The positioning tests assessed the accuracy of double integrating the acceleration vectors in a constant frame of reference to get the position of the IMU at each discrete time step. These tests were successful in transforming the transmitted rotation rates to Euler rates, but other than that they were largely unsuccessful. They failed the primary purpose of the test: to produce an accurate two dimensional plot of the position of the IMU. Based on the results of this test, the position of the ARIES vehicle cannot be evaluated solely by the HG1700 nor can the IMU be used to measure the ocean currents. When used with the SANS navigation software filter, however, the IMU can be used to assist in the measurement of the velocity and acceleration of the vehicle.

As stated already, before the IMU can be implemented, the C++ setup must be debugged and sufficient long term testing must be done. When implementing the IMU, it should be incorporated with the SANS twelve-state complementary filter or something similar to it, and the C++ program DECODE4 must be modified to read the transmissions directly from the IMU. These are the necessary evaluations and modifications still needing attention before implementation.

Based on all these conclusions, this thesis has made great strides in taking an IMU designed for the high speed JDAM missile and installing it onto the slow moving ARIES UUV, but the process is far from complete. The test results have been made clear and the outline for future work well defined. With that, this thesis is concluded with the hope that it will be used as a guide for follow-on work to complete the implementation process.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] "Gyroscope." *McGraw-Hill Encyclopedia of Science and Technology.* Vol. 8. 2002 ed.

[2] "Brief History of Gyroscopes." <http://einstein.stanford.edu/content/education/EducatorsGuide/Page30.html> 19 FEB 2005.

[3] "Heading Sensors." <http://www.eng.yale.edu/ee-labs/morse/other/pos96ch2.pdf> 19 FEB 2005.

[4] Sharp, James. "Laser Gyroscopes." <http://www.mech.gla.ac.uk/~sharpj/lectures/lasers/notes/laser_gyro.pdf> 19 FEB 2005.

[5] "Critical Item Development Specification HG1700AG Inertial Measurement Unit – Asynchronous Serial Protocol" Honeywell International Inc., 2002.

[6] Dobrokhodov, Vladimir. Personal Interview on Serial Data Acquisition Problems. Monterey, CA. 08 JUN 2005.

[7] Healey, Tony. "Dynamics and Control of Mobile Robotic Vehicles." Class Notes, Naval Postgraduate School. Monterey, CA. Winter 2001.

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

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

A. C++ PROGRAM DECODE4.CPP

```
/* DECODE4.cpp
      This program reads binary data from a text file and outputs the
14 measurements that the HG1700 IMU transmits with each message */

#include<stdio.h>
#include<stdlib.h>



/* Processes word1.
  Format of word1 is:
      bits 15-8: Accelerometer temperature (centigrade)
      bit 7: a axis RLG in PLC reset (=1)
      bit 6: b axis RLG in PLC reset (=1)
      bit 5: c axis RLG in PLC reset (=1)
      bit 4: IMU failure
      bits 0-3: 4-bit counter*/

/* Output format is (IMU pass/fail, temperature (Celsius degrees),
counter (0-15) */

char word1(double decimal)
{   int bitvalue=32768, bitnumber=15, counter=0, bit=0, temperature=0;
    char test;

    while(bitnumber>=0) //scans and processes individual bits
    {   if((decimal-bitvalue)>=0)
        {   bit=1;
            decimal=decimal-bitvalue;
            if(bitnumber<=3)
               counter = counter + bitvalue;
            else if(bitnumber==4)
                 test = 70;
            else if(bitnumber>=8)
               temperature = temperature+bitvalue/256;
            else;
        }else
            if(bitnumber==4)
                test = 80;
            else;
        bitnumber = bitnumber - 1;
        bitvalue = bitvalue/2;
    }
    printf("(%c, %d, %d)\t",test,temperature,counter);

      return test; // SPK 5/24/05: added to make this compile
}
```

```
/* Processes word2.
  Format of word2 is:
      bits 7-0: software version number
      bits 8-10: reserved
      bits 11-15: process tests */

/* Tests (0=pass, 1=fail):
      bit 15: Processor
      bit 14: Memory
      bit 13: Other
      bit 12: Accelerometer
      bit 11: Gyroscope */

/* Output format:  If all tests pass:  "Pass [version number]";  if
test(s) failed: "Fail: [process]" */
void word2(double decimal)
{   double error=0;
    int bitvalue=32768, version=0, bit=0, temperature=0;
    char bitnumber=15;
    while(bitnumber>=0) scans and processes individual bits
    {   if((decimal-bitvalue)>=0)
        {   bit=1;
            decimal=decimal-bitvalue;
            if(bitnumber<=7)
               version = version + bitvalue;
            else
                error = error + bitvalue;
        }else;
        bitnumber = bitnumber - 1;
        bitvalue = bitvalue/2;
    }
    if(error==0)  // read test output and display results to screen
        printf("(Pass %d)\t",version);
    else if(error==32768)
        printf("(Fail: Processor)\t");
    else if(error==16384)
        printf("(Fail: Memory)\t");
    else if(decimal==8192)
        printf("(Fail: Other)\t");
    else if(error==4096)
        printf("(Fail: Accel)\t");
    else if(error==2048)
        printf("(Fail: Gyro)\t");
    else
        printf("(Fail: Multi.)\t");
}


FILE *IMUfp;
FILE *SAVEfp;

void main()
{   unsigned char bytevalue;
    int i=0, count=0, size;
    char filename[64], savename[64];
    double lsb, value;
```

```
    const double lsb1=0.00057220458984375, lsb2=0.03662109375,
lsb3=0.000000001164153218269348144531250,
lsb4=0.000000074505805969238281250;
/*(least significant bit (LSB) values according to unit spec. sheet)*/

    system("cls");
    printf("\n**********************************\n** Set Screen
buffer width to 300 **\n**********************************\n");
    printf("\nPossible data files in current directory:\n\n");
    system("dir *.cap /od /b");         // Possible file extensions
    system("dir *.txt /od /b");         // to display to screen
    printf("\nChoose a data file to view: ");
    scanf("%s",&filename);                // Selected file to view
    printf("Output file name: ");
    scanf("%s",&savename);                // Selected file to save
    printf("Enter number of data segments: ");
    scanf("%d",&size);
    printf("Filename: %s   Number of data segments:
%d\n\n",filename,size);

    printf("Angular Rate X\tAngular Rate Y\tAngular Rate Z\tLinear
accel X\tLinear accel Y\tLinear accel Z\tStatus Word1\tStatus
Word2\tDelta angle X\tDelta angle Y\tDelta angle Z\tDelta vel X\tDelta
vel Y\tDelta vel Z\n");

    printf("  (rad/sec)\t  (rad/sec)\t  (rad/sec)\t  (ft/sec^2)\t
(ft/sec^2)\t  (ft/sec^2)\t(P/F, deg.C)\t (Pass/Fail)\t (radians)\t
(radians)\t (radians)\t (ft/sec)\t (ft/sec)\t (ft/sec)\n\n");

    SAVEfp = fopen(savename,"wb");
    IMUfp = fopen(filename,"rb");
    while(count<size)   // Constant loop looking for (165,2) combo
    {   fread(&bytevalue,1,1,IMUfp);
        printf("%d\n",bytevalue);
        if(bytevalue==165)    //finds  sync. byte/message id (165, 2)
        {   fread(&bytevalue,1,1,IMUfp);
            if(bytevalue==2)
            {   while(i<40)   // evaluates next 40 bytes
                {   fread(&bytevalue,1,1,IMUfp); // Combines bytes
                    value = bytevalue;
                    ++i;
                    fread(&bytevalue,1,1,IMUfp);
                    value = value + 256*bytevalue;
                    ++i;
                    if(i>=18)
                    {   fread(&bytevalue,1,1,IMUfp);
                        value = value + 256*256*bytevalue;
                        ++i;
                        fread(&bytevalue,1,1,IMUfp);
                        value = value + 256*256*256*bytevalue;
                        ++i;
                        if(value>=2147483648)
                            value = value - 2147483648 - 2147483648;
                        }else
                        if(value>=32768)
                            value = value - 65536;
                        else;
```

```
                        if(i<=6)                //applies lsb values
                        value = value*lsb1;
                        else if(i<=12)
                            value = value*lsb2;
                        else if(i<=14);         //word1
                        else if(i<=16);         //word2
                        else if(i<=28)
                            value = value*lsb3;
                        else if(i<=44)
                            value = value*lsb4;
                        else;

                        if(i==14)
                           word1(value);        // Calls word1 function
                        else if(i==16)
                           word2(value);        // Calls word2 function
                        else
                        {   printf("%f\t",value);
                            fprintf(SAVEfp,"%f ",value);
                        }
                    }

                    printf("\n");
                    fprintf(SAVEfp,"\r\n");
                    ++count;
                    i=0;
                }else;
            }else;
        }
    }
```

B. MATLAB PROGRAM IMUTEST_ROTATION.M

```
% IMUtest_rotation.m
% This progam integrates the rotation rates to get angle of rotation.
% It also plots the accelerations vs. time, the rotation rates vs.
% time, and the angle of rotation vs. time.
% 042105
% Joel Gow

clear all


test1 = load('042105_02mod.txt');       % 10Hz test


%////////////////// Rotation Test \\\\\\\\\\\\\\\\\\\\
t1 = 0:0.1:42.9;                        % Time vector in seconds

% Plot accelerations vs. time
figure(1)
plot(t1,test1(:,1),'-')         % aX
hold on
plot(t1,test1(:,3),'-.')        % aY
plot(t1,test1(:,2),':')         % aZ
title('Test1: Acceleration in the X,Y, and Z directions')
xlabel('Time (sec)')
ylabel('Acceleration (g)')
legend('X axis','Y axis','Z axis')

% Plot rotation rates vs. time
figure(2)
plot(t1,test1(:,4),'-')         % rotation about x
hold on
plot(t1,test1(:,6),'-.')        % rotation about y
plot(t1,test1(:,5),':')         % rotation about z
title('Test1: Rotation about the X,Y, and Z axes')
xlabel('Time (sec)')
ylabel('Rotation (deg/hr)')
legend('Roll','Pitch','Yaw')



%////////////////// Calculations \\\\\\\\\\\\\\\\\\\\
angular_position(1) = 0;        % initialize angular position
tO = 0.1/3600;                  % time step (hrs)
for i=1:430
    integral_stepPitch(i) = test1(i,6)*tO;   % integrate each time step
    integral_stepRoll(i) = test1(i,4)*tO;
    integral_stepYaw(i) = test1(i,5)*tO;

    angular_positionPitch(i) = sum(integral_stepPitch); % sum steps
    angular_positionRoll(i) = sum(integral_stepRoll);
    angular_positionYaw(i) = sum(integral_stepYaw);
end
```

```
% Plot angle of rotation vs. time
figure(3)
plot(t1,angular_positionRoll,'-')
hold on
plot(t1,angular_positionPitch,'-.')
plot(t1,angular_positionYaw,':')
title('Angular Position vs. Time')
xlabel('Time (sec)')
ylabel('Pitch Angle (degrees)')
legend('Roll','Pitch','Yaw')


% Plot lines at +/- 90 degrees for check
for i = 1:430
    y1(i) = 90;
    y2(i) = -90;
end

plot(t1,y1,'--')
plot(t1,y2,'--')
```

C. MATLAB PROGRAM IMUTEST_IDLE##.M

```
% IMUtest_idle4.m
% This program measures rotation rates over time compares with known
% earth rotation rates
% 042105
% Joel Gow


clear all

% These are the only inputs that change from each idle test
test1 = load('042105_04mod.txt');       % 1Hz test

file_length = 70;       % Length of file in seconds
rate = 1;               % Period of each time step

%////////////////// Test 1 \\\\\\\\\\\\\\\\\\\\

t1 = 0:1/rate:(file_length - 1)/rate;     % Create time vector (sec)

% Plot acceleration vs. time
figure(1)
plot(t1,test1(:,1),'g')
hold on
plot(t1,test1(:,3),'r')
plot(t1,test1(:,2),'b')
title('Test1: Acceleration in the X,Y, and Z directions')
xlabel('Time (sec)')
ylabel('Acceleration (g)')
legend('X axis','Y axis','Z axis')

% Plot rotation rates vs. time
figure(2)
plot(t1,test1(:,4),'g')
hold on
plot(t1,test1(:,6),'r')
plot(t1,test1(:,5),'b')
title('Test1: Rotation about the X,Y, and Z axes')
xlabel('Time (sec)')
ylabel('Rotation (deg/hr)')
legend('Roll','Pitch','Yaw')


wx = test1(:,4);        % Rotation rate vector about x
wy = test1(:,6);        % Rotation rate vector about y
wz = test1(:,5);        % Rotation rate vector about z


% mean values for rotation rates
wx_exp = mean(wx);
wy_exp = mean(wy);
wz_exp = mean(wz);
```

```
% Total experimental rotation rate
w_total_exp = sqrt(wx_exp^2 + wy_exp^2 + wz_exp^2);

lat = 36.5896;  % Latitude in degrees
we = 15;        % Earth's rotation rate in degrees/hour
N_offset = 0;   % IMU heading in degrees

% Calculate earth rotation rates about x, y, and z axes
wx_cal = -we*cos(lat*pi/180)*cos(N_offset*pi/180);
wy_cal = -we*cos(lat*pi/180)*sin(N_offset*pi/180);
wz_cal = -we*sin(lat*pi/180);

% Total calculated rotation rate
w_total_cal = sqrt(wx_cal^2 + wy_cal^2 + wz_cal^2);

% Display data to screen
disp('TEST 4: 1 Hz Data Rate. Duration - 70sec')
disp('    wx_exp    wy_exp    wz_exp    wx_cal    wy_cal    wz_cal
wt_exp    wt_cal')
disp([wx_exp wy_exp wz_exp wx_cal wy_cal wz_cal w_total_exp
w_total_cal])

disp('    ')
disp('    Computed Heading by taking ATAN of Wy/Wx')
disp('   Experiment      True')

% Calculate heading
heading = atan2(wy_exp,wx_exp)*180/pi;
heading_check = atan(wy_cal/wx_cal)*180/pi;


% Display data to screen
disp([heading heading_check])
```

D. MATLAB PROGRAM IMUTEST_POSITION.M

```
% IMUtest_position.m
% This program double integrates the acceleration vectors in the x
% and y axes to get position, and it plots the results.
% 04APR05
% Joel Gow

clear all

% Test one had corrupted data. Only test 2-5 were used.

test2 = load('040605_07mod.txt');       % 10Hz circular path test
test3 = load('040605_09mod.txt');       % 10Hz square path test
test4 = load('040605_11mod.txt');       % 10HZ straight path test
test5 = load('040605_12mod.txt');       % 10HZ drift path test

% Initial Variables
tOs = 0.1;                  % Time step for 10Hz = 0.1 second
tOh = 0.1/3600;             % Convert time step to hours
g = 32.2;                   % Force of gravity in ft/sec^2
rad = pi/180;               % conversion factor for degrees to radians


% ///////////////// Test 2 \\\\\\\\\\\\\\\\\
% Circular Path Test
t2 = 0:.1:22.4;             % Length of test 2 in seconds

% Acceleration vectors
aX = test2(:,1)*g;
aY = test2(:,3)*g;
aZ = test2(:,2)*g;

% Initialize angle of rotation values
Roll2(1) = 0;
Pitch2(1) = 0;
Yaw2(1) = 0;

% Calculate Euler angles
for i = 1:224
    % Transformation matrix
    T = [1 sin(Roll2(i)*rad)*tan(Pitch2(i)*rad)
cos(Roll2(i)*rad)*tan(Pitch2(i)*rad); 0 cos(Roll2(i)*rad) -
sin(Roll2(i)*rad); 0 sin(Roll2(i)*rad)/cos(Pitch2(i)*rad)
cos(Roll2(i)*rad)/cos(Pitch2(i)*rad)];

    % Euler rotation rates
    Euler_angle = T*[test2(i,4); test2(i,6); test2(i,5)];
    Ex = Euler_angle(1);
    Ey = Euler_angle(2);
    Ez = Euler_angle(3);

    % Integrate rotation rates to get angles
    Roll2(i+1) = Ex*tOh + Roll2(i);
    Pitch2(i+1) = Ey*tOh + Pitch2(i);
```

```
    Yaw2(i+1) = Ez*tOh + Yaw2(i);
end


figure(1)

% Plot angular position vs. time
subplot(2,1,1)
plot(t2,Roll2,'-')
hold on
plot(t2,Pitch2,'-.')
plot(t2,Yaw2,':')
title('Circular Test: Angular Position vs. Time')
xlabel('Time (sec)')
ylabel('Pitch Angle (degrees)')
legend('Roll','Pitch','Yaw')

% Transfer acceleration vectors to constant reference frame
net_aX = aX'.*cos(Yaw2*rad) + aY'.*sin(Yaw2*rad);
net_aY = -aX'.*sin(Yaw2*rad) + aY'.*cos(Yaw2*rad);

% Initialize position and velocity values
pX2(1) = 0;
pY2(1) = 0;
vX2 = 0;
vY2 = 0;

% get position by double integration of acceleration
for i=2:225
    vX2 = vX2 + net_aX(i-1)*tOs;
    vY2 = vY2 + net_aY(i-1)*tOs;

    pX2(i) = pX2(i-1) + vX2*tOs;
    pY2(i) = pY2(i-1) + vY2*tOs;
end

% Plot track of IMU
subplot(2,1,2)
plot(pX2,pY2)
title('Circular Test: 2D XY Plot')
xlabel('X position (ft)')
ylabel('Y position (ft)')
grid



% ///////////////// Test 3 \\\\\\\\\\\\\\\\\\
% Square Path Test
t3 = 0:.1:37.2;             % Length of test 3 in seconds

% Acceleration vectors
aX3 = test3(:,1)*g;
aY3 = test3(:,3)*g;
aZ3 = test3(:,2)*g;
```

```
% Initialize angle of rotation values
Roll3(1) = 0;
Pitch3(1) = 0;
Yaw3(1) = 0;

% Calculate Euler angles
for i = 1:372
    % Transformation matrix
    T = [1 sin(Roll3(i)*rad)*tan(Pitch3(i)*rad)
cos(Roll3(i)*rad)*tan(Pitch3(i)*rad); 0 cos(Roll3(i)*rad) -
sin(Roll3(i)*rad); 0 sin(Roll3(i)*rad)/cos(Pitch3(i)*rad)
cos(Roll3(i)*rad)/cos(Pitch3(i)*rad)];

    % Euler rotation rates
    Euler_angle = T*[test3(i,4); test3(i,6); test3(i,5)];
    Ex = Euler_angle(1);
    Ey = Euler_angle(2);
    Ez = Euler_angle(3);

    % Integrate rotation rates to get angles
    Roll3(i+1) = Ex*tOh + Roll3(i);
    Pitch3(i+1) = Ey*tOh + Pitch3(i);
    Yaw3(i+1) = Ez*tOh + Yaw3(i);
end


figure(2)

% Plot angular position vs. time
subplot(2,1,1)
plot(t3,Roll3,'-')
hold on
plot(t3,Pitch3,'-.')
plot(t3,Yaw3,':')
title('Square Test: Angular Position vs. Time')
xlabel('Time (sec)')
ylabel('Pitch Angle (degrees)')
legend('Roll','Pitch','Yaw')

% Transfer acceleration vectors to constant reference frame
net_aX3 = aX3'.*cos(Yaw3*rad) + aY3'.*sin(Yaw3*rad);
net_aY3 = -aX3'.*sin(Yaw3*rad) + aY3'.*cos(Yaw3*rad);

% Initialize position and velocity values
pX3(1) = 0;
pY3(1) = 0;
vX3 = 0;
vY3 = 0;

% get position by double integration of acceleration
for i=2:373
    vX3 = vX3 + net_aX3(i-1)*tOs;
    vY3 = vY3 + net_aY3(i-1)*tOs;
```

```
    pX3(i) = pX3(i-1) + vX3*tOs;
    pY3(i) = pY3(i-1) + vY3*tOs;
end

% Plot track of IMU
subplot(2,1,2)
plot(pX3,pY3)
title('square Test: 2D XY Plot')
xlabel('X position (ft)')
ylabel('Y position (ft)')
grid



% ///////////////// Test 4 \\\\\\\\\\\\\\\\\\
% Staight Path Test
t4 = 0:.1:16.4;             % Length of test 4 in seconds

% Acceleration vectors
aX4 = test4(:,1)*g;
aY4 = test4(:,3)*g;
aZ4 = test4(:,2)*g;

% Initialize angle of rotation values
Roll4(1) = 0;
Pitch4(1) = 0;
Yaw4(1) = 0;

% Calculate Euler angles
for i = 1:164
    % Transformation matrix
    T = [1 sin(Roll4(i)*rad)*tan(Pitch4(i)*rad)
cos(Roll4(i)*rad)*tan(Pitch4(i)*rad); 0 cos(Roll4(i)*rad) -
sin(Roll4(i)*rad); 0 sin(Roll4(i)*rad)/cos(Pitch4(i)*rad)
cos(Roll4(i)*rad)/cos(Pitch4(i)*rad)];

    % Euler rotation rates
    Euler_angle = T*[test4(i,4); test4(i,6); test4(i,5)];
    Ex = Euler_angle(1);
    Ey = Euler_angle(2);
    Ez = Euler_angle(3);

    % Integrate rotation rates to get angles
    Roll4(i+1) = Ex*tOh + Roll4(i);
    Pitch4(i+1) = Ey*tOh + Pitch4(i);
    Yaw4(i+1) = Ez*tOh + Yaw4(i);
end


figure(3)

% Plot angular position vs. time
subplot(2,1,1)
plot(t4,Roll4,'-')
```

```
hold on
plot(t4,Pitch4,'-.')
plot(t4,Yaw4,':')
title('Straight Line Test: Angular Position vs. Time')
xlabel('Time (sec)')
ylabel('Pitch Angle (degrees)')
legend('Roll','Pitch','Yaw')

% Transfer acceleration vectors to constant reference frame
net_aX4 = aX4'.*cos(Yaw4*rad) + aY4'.*sin(Yaw4*rad);
net_aY4 = -aX4'.*sin(Yaw4*rad) + aY4'.*cos(Yaw4*rad);

% Initialize position and velocity values
pX4(1) = 0;
pY4(1) = 0;
vX4 = 0;
vY4 = 0;

% get position by double integration of acceleration
for i=2:165
    vX4 = vX4 + net_aX4(i-1)*tOs;
    vY4 = vY4 + net_aY4(i-1)*tOs;

    pX4(i) = pX4(i-1) + vX4*tOs;
    pY4(i) = pY4(i-1) + vY4*tOs;
end

% Plot track of IMU
subplot(2,1,2)
plot(pX4,pY4)
title('Straight Line Test: 2D XY Plot')
xlabel('X position (ft)')
ylabel('Y position (ft)')
grid


% ///////////////// Test 5 \\\\\\\\\\\\\\\\\\
% Drift Path Test
t5 = 0:.1:17.3;             % Length of test 5 in seconds

% Acceleration vectors
aX5 = test5(:,1)*g;
aY5 = test5(:,3)*g;
aZ5 = test5(:,2)*g;

% Initialize angle of rotation values
Roll5(1) = 0;
Pitch5(1) = 0;
Yaw5(1) = 0;

% Calculate Euler angles
for i = 1:173
    % Transformation matrix
    T = [1 sin(Roll5(i)*rad)*tan(Pitch5(i)*rad)
cos(Roll5(i)*rad)*tan(Pitch5(i)*rad); 0 cos(Roll5(i)*rad) -
```

```
sin(Roll5(i)*rad); 0 sin(Roll5(i)*rad)/cos(Pitch5(i)*rad)
cos(Roll5(i)*rad)/cos(Pitch5(i)*rad)];

    % Euler rotation rates
    Euler_angle = T*[test5(i,4); test5(i,6); test5(i,5)];
    Ex = Euler_angle(1);
    Ey = Euler_angle(2);
    Ez = Euler_angle(3);

    % Integrate rotation rates to get angles
    Roll5(i+1) = Ex*tOh + Roll5(i);
    Pitch5(i+1) = Ey*tOh + Pitch5(i);
    Yaw5(i+1) = Ez*tOh + Yaw5(i);
end


figure(4)

% Plot angular position vs. time
subplot(2,1,1)
plot(t5,Roll5,'-')
hold on
plot(t5,Pitch5,'-.')
plot(t5,Yaw5,':')
title('Drift Test: Angular Position vs. Time')
xlabel('Time (sec)')
ylabel('Pitch Angle (degrees)')
legend('Roll','Pitch','Yaw')

% Transfer acceleration vectors to constant reference frame
net_aX5 = aX5'.*cos(Yaw5*rad) + aY5'.*sin(Yaw5*rad);
net_aY5 = -aX5'.*sin(Yaw5*rad) + aY5'.*cos(Yaw5*rad);

% Initialize position and velocity values
pX5(1) = 0;
pY5(1) = 0;
vX5 = 0;
vY5 = 0;

% get position by double integration of acceleration
for i=2:174
    vX5 = vX5 + net_aX5(i-1)*tOs;
    vY5 = vY5 + net_aY5(i-1)*tOs;

    pX5(i) = pX5(i-1) + vX5*tOs;
    pY5(i) = pY5(i-1) + vY5*tOs;
end

% Plot track of IMU
subplot(2,1,2)
plot(pX5,pY5)
title('Drift Test: 2D XY Plot')
xlabel('X position (ft)')
ylabel('Y position (ft)')
grid
```

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Anthony Healey
   Naval Postgraduate School
   Monterey, California

4. Edward Thornton
   Naval Postgraduate School
   Monterey, California

5. Donald Brutzman
   Naval Postgraduate School
   Monterey, California