Attitude Estimation for Motion Stabilization in Sonar Systems

Johan Kleiberg Jensen

Master Thesis

Department of Physics
University of Oslo

June 2013

Abstract

This thesis describes a project in attitude compensation for sonar applications using Micro Electrical Mechanical Systems (MEMS) accelerometers and gyroscopes. An attitude determination algorithm based on a complimentary filter has been implemented to run on a Raspberry Pi single-board computer. Attitude performance tests have been conducted in controlled environment and in a typical field environment.

Contents

1 Introduction **10**
1.1 Preliminary information . . . 10
1.1.1 Abbreviations . . . 10
1.1.2 Marine Terminology . . . 10
1.1.3 Conventions . . . 11
1.2 Opening Words . . . 11
1.3 Problem Statement . . . 12
1.4 Motivation . . . 12
1.5 Commercial Units . . . 12
1.6 Attitude Estimation . . . 13
1.7 Thesis outline . . . 13
1.8 Time Line . . . 14

2 Theory **16**
2.1 Inertial Motion . . . 16
2.2 Rotational Formalism . . . 17
2.2.1 Vector mathematics . . . 17
2.2.2 Reference Frames . . . 18
2.2.3 Euler Angles . . . 18
2.2.4 Rotation Matrices . . . 19
2.2.5 Quaternions . . . 21
2.2.6 Complimentary Filter . . . 21
2.3 Hydro-acoustic range and angle determination . . . 22
2.4 Sensors . . . 22
2.4.1 Accelerometers . . . 22
2.4.2 Gyroscopes . . . 25
2.4.3 Magnetometer . . . 27
2.4.4 Barometer . . . 27
2.4.5 Other Sensors . . . 27
2.5 MEMS Technology . . . 28
2.5.1 MEMS Accelerometer . . . 29
2.5.2 MEMS Gyroscope . . . 29
2.6 PID controller . . . 30

3 Method **31**
3.1 Considerations . . . 31
3.1.1 Abbe Error . . . 31
3.1.2 Real-time systems . . . 31
3.1.3 Bus Technologies . . . 31
3.1.4 Level shifters . . . 32
3.1.5 Intellectual Property . . . 32
3.2 First approach to building an attitude system . . . 33
3.2.1 Sensors . . . 33
3.2.2 PCBs . . . 33
3.2.3 Testing . . . 37
3.2.4 Software to read sensors . . . 39
3.2.5 Results from testing . . . 39
3.3 Second approach to building an attitude system . . . 39

3.3.1 MPU9150 Sensor 39
3.3.2 RaspberryPi 41
3.3.3 Raspbian 42
3.3.4 Assembled Hardware 43
3.3.5 Price and availability 46
3.3.6 Current Consumption 46
3.3.7 Attitude estimation 47
3.3.8 Complimentary Filter 48
3.3.9 Implementing the Algorithm 51
3.3.10 Serial Data Output Format 52
3.3.11 Other Software 53
3.4 Control Software 53
3.4.1 Description 54
3.4.2 Commands 54

4 Testing **56**
4.1 Static Test 56
4.1.1 Static sensor testing 56
4.1.2 Temperature Testing 60
4.1.3 Filter parameters and tuning 62
4.1.4 Complimentary filter static test 64
4.1.5 Observations from the static test 65
4.2 Dynamic Test 65
4.2.1 Setup 65
4.2.2 Data 67
4.2.3 Shake test 72
4.2.4 Observations from the dynamic tests 73
4.3 Tachometer test 74
4.3.1 Setup 74
4.3.2 Data 75
4.3.3 Observations from tachometer test 79
4.4 Field test at Songsvann 79
4.4.1 Setup 79
4.4.2 Description of tests 81
4.4.3 Data 82
4.4.4 Observations from Field Test 89

5 Summary **90**
5.1 Improvements and further work 91

6 Conclusion **93**

7 Bibliography **94**

A Contents of DVD-ROM **95**

B NMEA PASHR String **98**

C RaspIMU Source Code **99**

List of Tables

1 Project Time-line 14
2 Planned Project Time-line 15
3 Performance of the MMA8451Q, LIS331DLH, and IMU3000 sensors quoted in the respective data-sheets. 33
4 Invensense MPU9150 quoted specifications. 40
5 Raspberry PI Hardware summary. 41
6 Cable and connector pin arrangement in external cable. 44
7 Internal connections between components. 46
8 Total component costs. 46
9 Quoted current draw of components and measured total current draw. 46
10 Baud rates versus byte throughput. 53
11 Custom data output string 53
12 Serial commands sent by the RaspIMU control program. 55
13 Standard Deviation and Average for Gyroscope and Accelerometer for the MPU9150 at 50Hz. 59
14 Standard deviation and average of raw data for gyroscope and accelerometer for the MPU9150 at 25Hz. 60
15 Standard deviation and average units for gyroscope and accelerometer for the MPU9150 at 25Hz. 60
16 MPU9150 accelerometer output versus temperature relationship . 62
17 Complimentary filter attitude output statistics in static test . . . 65
18 Servo PWM control signals for MSG995 servos. 66
19 Angle output correction based on the dynamic test. 70
20 Measurements of top and bottom angles in tachometer test. . . . 75
21 Tachometer tests performed. 75
22 Summarised tachometer test results. 79
23 Description of tests performed at Sognsvann 81
24 Statistics on vertical positions from test 2. 83
25 Statistics on vertical positions from test 2. 86
26 Statistics on vertical positions from test 8. 87
27 Test 8. Period of swing of the hydrophone. 87
28 Summary of test results from Sognsvann field test in degrees. . . 89

List of Figures

1 A model of a two dimensional navigation system. 17
2 Euler angle transformation in three dimensions. 19
3 Simple rotation of θ. in a two dimensional system. 20
4 Calculating horizontal position with two hydrophones. 22
5 Conceptual model of an Accelerometer. 23
6 Conceptual Model of a Gyroscope. 26
7 Side view of a typical MEMS device. 28
8 Close-up of the LIS331DLH MEMS accelerometer. 29
9 Close-up of the IDG300Q MEMS Gyroscope. 30
10 Bi-directional level shifter using MOS transistors. 32
11 Invensense IMU3000 sensor PCB made for this project. 34
12 STMicroelectronics LIS331DLH sensor PCB made for this project. 34
13 Freescale Semiconductors MMA8151Q sensor PCB made for this project. 35
14 Atmel ATMega8 micro-controller PCB made for this project. . . 35
15 Power and serial PCB made for this project. 36
16 Sensors, micro-controller, and power and serial board stacked together. 37
17 MMA8451Q accelerometer acceleration data recorded on the DSV Acergy Osprey. 38
18 LIS331DLH accelerometer pitch and roll angle calculations recorded on the DSV Acergy Osprey. 38
19 RawReader screen shot that decodes raw data from the sensors. . 39
20 The MPU9150 Engineering Evaluation Board. 40
21 The Raspberry Pi single-board computer. 41
22 Raspbian GUI accessed through a VNC connection. 43
23 Waterproof enclosure and connector used for the attitude system. 44
24 Exploded annotated view of the hardware components in the attitude system. 45
25 Diagramatic overview of the components in the attitude system. 45
26 Angle calculation from acceleration data. 47
27 Complimentary filter top level data-flow. 48
28 Top-level control flow of the attitude estimation system. 51
29 RaspIMU control program user interface. 54
30 RaspIMU control program when resized to a small window. . . . 54
31 Histogram from raw accelerometer readings for the MPU9150 at 50Hz. 57
32 Histogram from raw Gyroscope X-axis readings for the MPU9150 at 50Hz. 58
33 Histogram from raw Gyroscope Y-axis readings for the MPU9150 at 50Hz. 58
34 Histogram from raw Gyroscope Z-axis readings for the MPU9150 at 50Hz. 59
35 MPU9150 Accelerometer raw output versus temperature. 61
36 MPU9150 Gyroscope raw output versus temperature. 61
37 Complimentary filter PI-regulator tuning for different I factors. . 63
38 Complimentary filter PI-regulator tuning for different P factors. . 63

39 Typical settling time for the attitude estimation system from power-on. 64
40 Pan-tilt test rig used in the dynamic test. 66
41 Step pattern test using pan-tilt servo rig as measured by the attitude system. 67
42 Triangle pattern test using pan-tilt servo rig as measured by the attitude system. 68
43 Pitch Angle Output for Dynamic Test with accelerometer based angle calculations overlayed. 68
44 Roll Angle Output for Dynamic Test. 69
45 Pitch Angle Discrepancy for dynamic test. 69
46 Roll Angle Discrepancy for dynamic test. 70
47 Pitch Angle overshoot during sharp transition. 71
48 Roll Angle overshoot during sharp transition. 71
49 Angle calculated from accelerometer for shake test. 72
50 Algorithm attitude output for shake test. 73
51 Overview of the tachometer test setup. 74
52 Tachometer test 1. plotted tilt angle from sensor and tachometer. 76
53 Tachometer test 2. plotted tilt angle from sensor and tachometer. 76
54 Tachometer test 3. plotted tilt angle from sensor and tachometer. 77
55 Tachometer test 4. plotted tilt angle from sensor and tachometer. 77
56 Tachometer test 8. 78
57 Tachometer test 10. plotted tilt angle from sensor and tachometer. 78
58 Preparing the instruments at Sognsvann field test. 80
59 Copper acoustic target used in the field test. 80
60 Test 2. Vertical position before applying attitude corrections. . . 82
61 Test 2. Pitch angle measured by the sensor 82
62 Test 2. Roll angle measured by the sensor 82
63 Test 2. Vertical position after applying attitude corrections. . . . 83
64 Test 7. Vertical Position before applying attitude corrections. . . 84
65 Test 7. Pitch angle measured by the sensor. 84
66 Test 7. Vertical position after applying attitude corrections. . . . 85
67 Test 7. Uncompensated target strength from the sonar. 85
68 Test 8. Vertical position before applying attitude corrections. . . 86
69 Test 8. Vertical Position after applying attitude corrections. . . . 87
70 Test 10. Vertical position before applying attitude corrections. . 88
71 Test 10. Vertical position after applying attitude corrections. . . 88

Preface

I have been a part-time student on the master program since the autumn of 2009. In that time I have had two different full-time jobs, my first son was born, and I have moved to a new house in a new city. I have spent evenings, weekends and what spare time I have had writing this thesis. In these last four years my loving family, Lene Amalie Aadahl and Johs Kleiberg Jensen, have been supportive, patient, and a source of inspiration.

I would like to give a special thank my supervisor, Dr. Helge Balk. You have gone out of your way to lend a hand when needed.

I would also like to thank my grand father, Johan Kleiberg, for urging me on and for providing financial support.

I would also like to thank Bendik Sovegjerto and Joakim Myrland, for assisting during the Sognsvann field test. And Bendik Sovegjarto again for developing the tilt test rig together with Dr. Helge Balk. In addition to the three mentioned gentlemen scholars, I would like to thank Adam Olson, Morten Huseby and Kjetil Rensel for academic support and discussions.

I would also like to thank the following people and companies for their support: MEMSIC and Nils Olav Gjorvad at ACAL Norway for MEMSIC Parts, Freescale for providing MEMS parts, Chipsworks for allowing me to use their MEMS die photos, and UiO ELAB for assisting with PCB production.

Finally, I would like to thank colleagues my at Acergy, ECN, and Kongsberg Maritime for being flexible during my studies.

1 Introduction

1.1 Preliminary information

1.1.1 Abbreviations

ADC	Analogue to Digital Converter
AHRS	Attitude Heading Reference System
DoF	Degrees of Freedom
FOG	Fibre Optical Gyroscope
GCC	GNU Compiler Collection
GNU	GNU's Not Linux
I2C	Inter Integrated Circuit
IC	Integrated Circuit
IMU	Inertial Motion Unit
INS	Inertial Navigation System
MARG	Magnetic Accelerometer Rate Gyroscope
MEMS	Micro Electrical Mechanical Systems
MRU	Motion Reference Unit
PCB	Printer Circuit Board
PWM	Pulse Width Modulation
SPI	Serial Peripheral Interface Bus
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VRU	Vertical Reference Unit

1.1.2 Marine Terminology

Bow	The front of the vessel
Stern	The back of the vessel
Roll	Rotation along the vessel
Pitch	Rotation across the vessel
Yaw	Rotation around the vessel
Heave	up and down motion of the vessel
Sway	Left to right motion of the vessel
Surge	back to forth motion of the vessel
CoG	Centre of Gravity of the vessel
Centre line	Imaginary line precisely alongship

1.1.3 Conventions

Notation that we will be using to represent variables.

Acceleration vector	$\vec{a} = [a_x, a_y, a_z]$
Velocity vector	$\vec{v} = [v_x, v_y, v_z]$
Displacement vector	$\vec{s} = [s_x, s_y, s_z]$
Angular acceleration	$\vec{\alpha} = [\alpha_x, \alpha_y, \alpha_z]$
Angular velocity	$\vec{\omega} = [\omega_x, \omega_y, \omega_z]$
Angular displacement	$[\phi, \theta, \psi]$
Identity Matrix	$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation Matrix	$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$

1.2 Opening Words

This thesis describes the design, development, and analyses of an attitude estimating system for use in sonar applications, and particularly in horizontal sonar surveys.

Norway has committed to the Water Framework Directive[1], which states that all member states of the European Union should establish good qualitative and quantitative status of all water bodies by 2015. Part of this is to estimate fish population. In Norway, some of these water bodies may lie in hard to reach locations, sometimes only reachable by foot.

In marine environments sonar is a commonly used tool for estimating fish population. The sonar can mounted on a boat and directed downwards, so that the sonar can scan for fish as the vessel moves. Using the same method for estimating fish population in fresh water bodies poses a few challenges. Many fresh water bodies and rivers are shallow, and the fish tend to congregate near the surface. Because of this, a downward directed sonar would not be very useful. It is therefore common to direct the sonar along the horizon in shallow waters, rather than vertically. But this poses new problems; Firstly, small offsets in the vessel can cause the sonar to be misaligned so that its echograms hit either the surface or the lakebed. Secondly, varying motion in the vessel can cause the fish to appear to be dancing around, when the fish are in fact stationary or swimming in a straight line.

What is needed is a system that can sense the attitude of the vessel, specifically the pitch and the roll of the vessel. With pitch and roll data it is possible to stabilise the dancing appearance caused by vessel motion, and to exclude data that hit the surface or the lake bed. These types of systems are often referred to as Motion Reference Units or Attitude Reference Systems (MRU/ARS). The devices that are available on market are often bulky, expensive, and mainly designed for offshore environments. Such a system would be a valuable to the hydro-acoustic group at UiO.

[1]Directive 2000/60/EC of the European Parliament and of the Council of 23 October 2000 establishing a framework for Community action in the field of water policy

Inertial navigation systems, abbreviated INS, is a collective term for a system that can estimate a relative position and attitude based on acceleration and angular rates of change. This is possible because the second derivative of acceleration is position, and similarly the derivative of the angular rate of change is angular displacement. INS was successfully used first in submarines during World War 1.

Micro-mechanical Systems, or MEMS for short, is the technology of the very small, typically made up of components between 1 and 100 micrometers in size. Commonly made of silicon and manufactured using processes adopted from integrated circuit manufacturing. Because of economies of scale, MEMS devices are usually cheap compared to their mechanical counterparts. Both accelerometers and gyroscopes have been adopted as MEMS devices.

Since this project was started in 2009, there has been an explosion of devices and the applications using them. Can recent advances in MEMS sensor technology make it possible to construct a low-cost system for attitude compensation in sonar systems?

1.3 Problem Statement

The use of sonar in a mobile vessel or stationary in rivers can be problematic. In mobile vessels the motion of the vessel will move the sonar around. In rivers, the current can induce motion and oscillations. This paper will investigate sensors that can measure this type of motion, so that the misaligned sonar data can be corrected.

The sensors should be put together in such a way that the motion recorded by that system can be meaningfully used by a sonar system or stored on a computer for later processing.

There is a wide variety of sensors and sensor technology for capturing motion. The price of a sensor must be pitted against size, noise, robustness, accuracy and precision.

The system must be able to operate in the environment of its intended use. Further, mobility, simplicity of use and low cost should be prioritised. Lastly, the system should be able to function well together with existing sonar systems and auxiliary equipment.

1.4 Motivation

MEMS is a fast developing technology that is constantly finding new applications due to its small size, cost, low power, reliability and increasing performance. Is it with today's MEMS devices possible to construct a system that addresses the problem of an offset transducer and to measure vibration that is cheap, robust, low-power?

With the ability to determine where a transducer is pointing at during a sonar survey, it should be possible to improve the data collected by the sonar system.

1.5 Commercial Units

There are commercially available IMU/AHRS units, like the Kongsberg Maritime's Motion Reference Unit (MRU). This unit is however quite costly, costing up to several hundred thousand Norwegian Kroner, depending on type and options. It is also a little heavy at 2.5 Kg, and consumes up to 12 W of power[2].

Another interesting product is the x-IMU from X-IO Technologies[3]. This is an open-source and open-hardware project. The project is based on the same papers and community sources as our project, and has many similar sensor components. The x-IMU was made available in 2012, and costs only 249 (GBP) for the basic unit.

1.6 Attitude Estimation

The method of attitude estimation we use in this project is based on a complimentary filter described in papers by Robert Mahoney et al[4], and the community work behind diydrones[5] and gentlenav[6], in particular the work of William Premerlani and Paul Bizzrd[10].

This approach to calculating the attitude of a vehicle using mentioned method was developed with aircraft in mind, but there is no reason why it shouldn't work for our application. There are some aspects of the filter that we will not use, such as wind speed calculations and the assumption of the availability of a GPS.

Kalman filtering is another method that can be used to estimate attitude. It has been used in many guidance and navigation system, including the NASA space shuttle and the International Space Station.

By comparison, the complimentary filter method is much simpler than the Kalman filter method.

An internal paper by Sebastian O.H. Madgwick titled *An efficient orientation filter for inertial and inertial/magnetic sensor array*[7] concluded that a filter fusion algorithm[8] performed as well as a high quality commercial based Kalman-based system[9].

We ultimately chose the complimentary filter method because it is simpler and less computationally expensive.

1.7 Thesis outline

The Theory chapter will discuss inertial sensors with emphasis on the MEMS version of inertial sensors. Next, the chapter will introduce a formal mathematical framework for representing attitude. The paper will then go on to looking at how these sensors can be used in that framework.

The Method chapter will open up by looking at our first approach to building an attitude estimating system. We built a second attitude estimating system based on experiences gained from the first approach. We will give an description of the various components used in the final system. We will also describe the

[2]Quoting from data-sheet, included in appendix
[3]http://www.x-io.co.uk/products/x-imu/
[4][3][4][5][6]. Papers included in appendix A
[5]http://diydrones.com/
[6]http://code.google.com/p/gentlenav/
[7][11]. Paper included in appendix A.
[8]complimentary filter
[9]pg. 25

control program we wrote to go along the final attitude system. The implementation of the complimentary filter method for estimating attitude (roll and pitch) will also be described.

The Test chapter we look at how the system was tested, both in the lab and in the field.

The Summary chapter will sum up the results from testing. We will also suggested improvements and further work.

The Conclusion chapter will sum up the papers most important findings from the tests, and also what we learned.

Bibliography will list references used.

The appendix will also list the contents of the provided CD-ROM, which includes source code, circuit schematics and diagram, logged test data, data sheets, and other supporting information mentioned in this thesis.

1.8 Time Line

Table 1 shows the time-line of the project as it happened.

Year	Semester	Plan
2010	Spring	Problem definition.
2010	Autumn	Testing accelerometers and gyroscope sensors. Construction of test cards.
2011	Spring	Researching ways to implement AHRS system.
2011	Autumn	Attitude algorithm testing.
2012	Spring	Algorithm implementation on RaspberryPi.
2012	Autumn	Final hardware assembly and software programming. Static testing and data analyses.
2013	Spring	Field Test. Tachometer test. Dynamic test. System Completion.

Table 1: Project Time-line

Table 2 shows the original time allocation for the project, as it was planned in spring 2010.

Year	Semester	Part
2010	Spring	Research sensor and AHRS systems
2010	Autumn	Selecting suitable sensors
2011	Spring	Testing sensors and finding a suitable hardware platform
2011	Autumn	Planning
2012	Spring	Building prototype
2012	Autumn	Testing and field testing
2013	Spring	Analyses of results completion of system

Table 2: Planned Project Time-line

The project had many twists and turns, and towards the last semester things got a little hectic. All in all, the project went mostly as planned.

2 Theory

In this chapter we will introduce the theory that we will use in this project. The bulk of this chapter is dedicated inertial sensors and how to determine attitude from motion sensors.

2.1 Inertial Motion

Inertial navigation is the use of motion sensors (accelerometers) and rotation sensors (gyroscopes) to continuously calculate the position, orientation, and velocity without the need of external references. This is usually referred to as a Inertial Navigation System (INS). If the system doesn't calculate position, it is sometimes referred to as an Inertial Motion Unit (IMU), Attitude Heading Reference System (AHRS), Vertical Reference Unit (VRU) or Motion Reference Unit (MRU).

Some areas of application for Inertial Navigation Systems are aeroplanes, submarines, missiles, satellites, consumer electronics, and ships.

The first use of inertial technology for navigation utilised a stabilised platform, which is where the inertial sensors are isolated from the rotational motion of the vehicle through gimbals[10] A strap-down systems on the other hand, is the term given to a navigation system where the sensors are rigidly attached, or 'strapped down', to the body of the host vehicle. This reduces cost, size, and increases reliability compared to a platform system, all of which are important objectives for this project. The penalties are increased computing complexity and a demand on sensors to be capable of measuring much higher rates of turn[11].

With reference to figure 1[12], consider a one-dimensional system where an object can move along a two-dimensional train track.

[10][1] pg. 546
[11][1]. pg.3
[12]Example and figure from [1] pg. 9

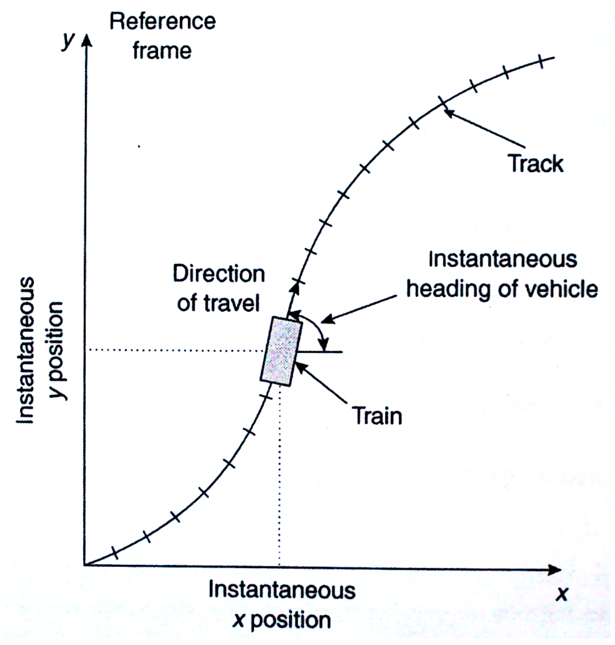


Figure 1: A model of a two dimensional navigation system.

Assume that there are no other forces but the accelerating force acting upon this rigid object. Further, assume that the object was initially stationary and we can absolutely measure the objects acceleration, a_x and a_y, and orientation θ. We would then be able to calculate the objects position, s_x and s_y, and velocity v_x and v_y, using Newtons laws of motion. This model can be extended to a three-dimensional system which is free to move and rotate in space.

The number of independent axis are commonly referred to as degrees of freedom (DoF). The above examples is a three-degree system. If we include rotational motion in the system, we add these to the number of degrees of freedom, i.e. three orthogonal linear axis and three orthogonal rotational axis is referred to as a six degrees of freedom system.

2.2 Rotational Formalism

This section will establish a formal system for describing attitude.

2.2.1 Vector mathematics

The following results are for three dimensional vectors.

Vector cross-product of matrices is defined as[13]:

$$\vec{b} \times \vec{c} = \begin{bmatrix} b_y c_z - b_z c_y \\ b_z c_x - b_x c_z \\ b_x c_y - b_y c_x \end{bmatrix}$$

[13][8] pg. 41

Vector dot-product is defined as[14]:

$$\vec{b} \cdot \vec{c} = b_x c_x + b_y c_y + b_z c_z$$

The angle between to vectors is given by[15]

$$\vec{b} \cdot \vec{c} = \|\vec{b}\| \|\vec{c}\| \cos\theta$$

A vector can be scaled in the following way:

$$a\vec{b} = \begin{bmatrix} ab_x \\ ab_y \\ ab_z \end{bmatrix}$$

A unit vector is a vector whose length is 1. A vector can be scaled to become a unit vector by scaling by its length[16]:

$$\hat{b} = \| \vec{b} \| = \frac{1}{\sqrt{{b_x}^2 + {b_y}^2 + {b_z}^2}} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

Lastly, a null vector is defined as a vector whose entries are zero.

2.2.2 Reference Frames

Reference frames are defined sets of orthogonal, right-handed, axis sets. We will use two reference frames in this project; the body frame and the navigation frame[17].

The body frame has its axis aligned with the roll, pitch and yaw axis of the sensor, and has its origin at the sensor. Our body frame is the sonar and our attitude sensor.

The navigation frame also has its origin at the sensor, but axis aligned with North, East and vertically down. This is our reference to the ground.

The angles between the body frame and the navigation frame are the angles we want to estimate to compensate echograms in sonar surveys.

Next, we describe how to transform between two reference frames.

2.2.3 Euler Angles

Euler angles is a system where the transformation between two reference frames is described by three successive rotations[18]. The transformation can be summarised as:

1. Rotate through angle ψ about reference z-axis
2. Rotate through angle θ about new y-axis
3. Rotate through angle ϕ about new x-axis

[14][8] pg. 375
[15][8] pg. 381
[16][8] pg. 429
[17][1] pg. 21-22
[18][1] pg 40-42

The angles ψ, θ, and ϕ are referred to as the Euler rotation angles. Figure 2 shows a general transformation[19].

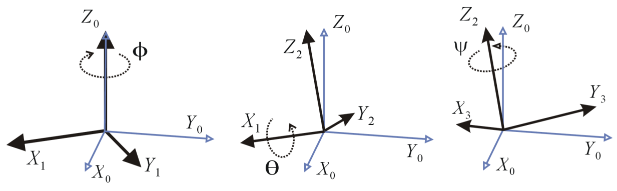


Figure 2: Euler angle transformation in three dimensions.

The Euler angle representation of rotation is popular because it corresponds to the angles at the pick-ups of a gimbal. A gimbal gyroscope will be described in the gyroscope section.

2.2.4 Rotation Matrices

Rotation Cosine Matrix, sometimes abbreviated as DCM (Direction Cosine Matrix) or simply rotation matrix, is a matrix representation of describing orientation. The DCM is a $n \times n$ matrix, for an n-dimensional representation, where the columns represents unit vectors in body axes projected along the reference axes. We use three dimensional vector space, so the matrix becomes a 3×3 matrix. This matrix is of the special orthogonal group $SO(3)$. The entries in a rotation matrix r_{ij} are real values between -1 and 1. The rows of the rotation matrix can be thought of as unit vectors along the axis of the rotated space.

For a two dimensional version of a rotation matrix, a rotation of θ from one reference frame to another can be written as:

$$\bar{R}(\theta) = \begin{Bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{Bmatrix}$$

The rotation can be illustrated as in figure 3[20]:

[19]Figure from wikipedia.org with permission http://en.wikipedia.org/wiki/File:EulerG.png
[20]Image from [http://en.wikipedia.org/wiki/File:Counterclockwise_rotation.png]

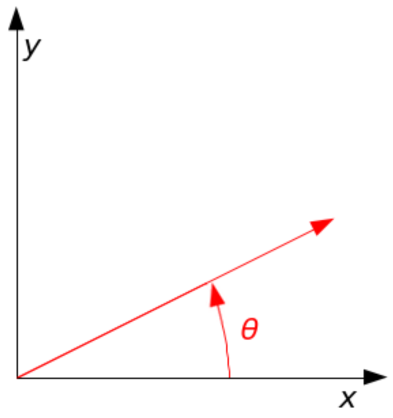


Figure 3: Simple rotation of θ. in a two dimensional system.

A three dimensional rotation matrix is represented as[21]:

$$\bar{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Rotation matrices can be added together. The resulting matrix will then represent the total rotation described by those two rotation matrices:

$$\bar{R_{total}} = \bar{R}_1 \bar{R}_2$$

The rotations are not commutative ($\bar{R}_1\bar{R}_2 \neq \bar{R}_2\bar{R}_1$). This is not so difficult to imagine: for example, pitching an aeroplane down and then rolling right will point a aeroplane in a different direction than if rolled right before pitching down.

The rotation matrix $\bar{R}$ is sometimes written as $\bar{C}_b^n$, where b and n denotes that this is the rotation matrix from the navigation frame (n) to the body frame (b). The entries in the rotation matrix r_{ij} are of real values, and the matrices are orthogonal with a determinant of one[22]:

$$\bar{R}^T = \bar{R}^{-1}, \det \bar{R} = 1$$

We can go from Euler angles to a rotation matrix representation using the result[23]:

$$\bar{R} = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\psi\sin\phi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}$$

[21][1] pg. 39-40

[22]This is the same as being in the special orthogonal group SO(n).

[23][1] pg. 41 Eq. (3.49)

The angles ψ, ϕ, and θ are the angles formed between two reference frames. Notice that if we insert zero angles (no rotation), we get an identity matrix.

To go from a rotation matrix to Euler angles we can use the following result[24].

$$\begin{aligned}\psi &= \arctan(\tfrac{r_{32}}{r_{33}})\\ \phi &= \arcsin(-r_{31})\\ \theta &= \arctan(\tfrac{r_{21}}{r_{11}})\end{aligned}$$

There are some situations were the above result becomes indeterminate, such as when r_{11} or r_{33} approaches zero. For these situations there are alternative transformations which will not be reproduced here.

Another result we will be using is the small angle rotation result[25], for when the sines of the Euler angles approximates the angles[26]:

$$\bar{R} \approx \begin{bmatrix} 1 & -\psi & \theta \\ \psi & 1 & -\phi \\ -\theta & \phi & 1 \end{bmatrix}$$

The above result will be used in the algorithm for when small angle updates of ψ, θ and ϕ are added to the rotation matrix that keeps track of current rotation.

2.2.5 Quaternions

We mention quaternions here briefly for completeness, because quaternions are another popular way to describe orientation of a rigid body. A quaternion is a four parameter representation of attitude[27]:

$$\bar{q} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = a + \vec{i}b + \vec{j}c + \vec{k}d$$

$\vec{i}$, $\vec{j}$ and $\vec{k}$ are complex numbers. We will not reproduce the relationship between quaternions, rotation matrices and Euler angles. The relationships can be found in [1] pg. 45.

2.2.6 Complimentary Filter

A complimentary filter is a method for combining accelerometer and gyroscope measurements, and optionally additional sensors. The idea is to take the strength of each sensor and fuse them together to get a better attitude estimate.

A full description of our implementation of a complimentary filter can be found in the *Method* chapter under *Complimentary Filter.*

[24] [1] pg.46 Eq. (3.66)
[25] [1] pg. 41 Eq. (3.50)
[26] $\sin\theta \to \theta, \sin\psi \to \psi, \sin\phi \to \phi$
[27] [1] pg. 42-43.

2.3 Hydro-acoustic range and angle determination

To determine target vertical position in hydro-acoustics we need to use two or more hydrophones that can pick up the returning echogram, as illustrated in figure 4.

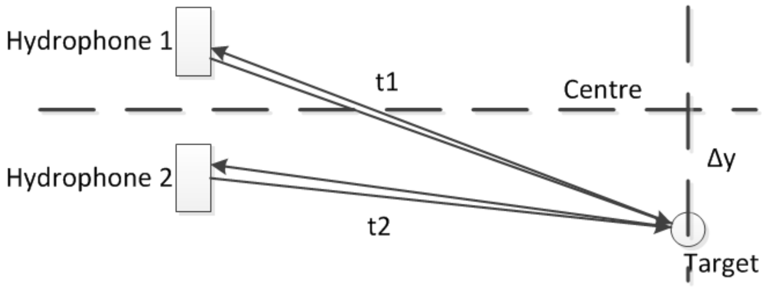


Figure 4: Calculating horizontal position with two hydrophones.

The time difference between receiving the echogram at the two hydrophones (t1 and t2) determine the vertical position. The time divided by two determine the range to the target. Trigonometry can be used to calculate the angle and the vertical displacement from the centre (Δy).

The Simrad EY500[28] hydrophone use the phase and not the amplitude to determine the time difference between the two hydrophones. The zero-crossing between the two detected signals are scaled by a factor to give degrees displacement from the centre.

2.4 Sensors

In this section we will present sensors that can be used to estimate motion and attitude.

A sensors range will be stated in terms of upper and lower bounds, for example $\pm 2g$ or $\pm 250°/S$. When outside of this range the sensor will become saturated. A saturated sensor will not report a higher measurement than its bounds. If we continue to subject the sensor to an increasing value, it will eventually break. This figure is higher than the range, often by several magnitudes. This data is usually stated as shock survivability.

Next we consider the sensitivity, which is the smallest change a sensor can detect. In digital accelerometers this figure is usually stated in $\frac{mg}{digit}$, or as $\frac{mV}{g}$ in analogue accelerometers.

Sampling rate is the rate at which a sensor takes measurements. For digital sensors such as digital accelerometers and digital gyroscopes, this is a user selectable setting.

2.4.1 Accelerometers

Accelerometers measure proper acceleration along their sensitive axis. This acceleration along straight lines is referred to as translational acceleration. By

[28]We used the EY500 for the field test at sognsvann

contrast, proper acceleration is a measure of *true* acceleration; acceleration with respect to its inertial space. The sensors inertial space is the same as its inertial frame.

Accelerometers do not measure acceleration directly, they measure forces. According to Newton's second law of motion, a force $\vec{F}$ acting upon a body of mass m will cause it to accelerate ($\vec{a}$) with respect to its inertial space:

$$\vec{F} = m\vec{a}$$

The force we measure, and by knowing the mass, we can estimate the acceleration experienced by that object. However, if we want to measure the acceleration of something through this relationship, it is not practical to measure the total force acting upon the body of interest. Instead, we measure the force acting upon a small mass that is coupled to the body of interest. This small mass is known as a proof mass or seismic mass.

When the two masses are coupled together, they must experience the same acceleration. This allows us to infer the acceleration of the system we are interested in, through the system we know something about. Conceptually this can be a proof-mass suspended between to springs, as illustrated in figure 5[29].

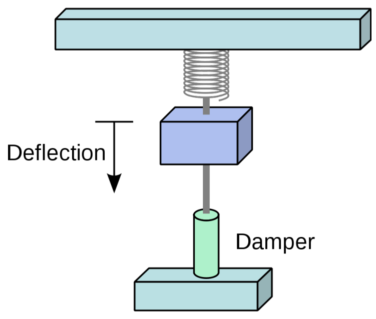


Figure 5: Conceptual model of an Accelerometer.

The figure shows a mass suspended by a spring one end, and a dampener in the other. The dampener represents certain characteristics of the acceleration, such as vibration suppression and oscillation suppression. Any acceleration along the accelerometers sensitive axis [30] will be resisted by the proof mass, due to the proof mass' own inertia. As a result, the proof mass is displaced with respect to the accelerometers body. The displacement will be balanced against the tension of the springs, and the net deflection of the springs is proportional to the force exerted on the proof mass. If we were to measure deflection, we could use Hooke's law to estimate the force. Through these two relationships it is possible to calculate the acceleration:

$$F = kX \quad [31]$$

[29]Illustration from Wikimedia Commons [http://commons.wikimedia.org/wiki/File:Pendular_accel.svg]

[30]In our simple concept accelerometer, the sensitive axis is parallel to the spring, mass and dampener.

[31][2] pg. 756

$$\vec{F} = m\vec{a} = m(\vec{f} + \vec{g})^{32}$$

The first equation is Hooke's law, which states that the force F is proportional displacement X multiplied by a spring constant k. The second equation is a decomposition of forces. $\vec{g}$ is the acceleration caused by gravity (if present), and $\vec{a}$ is the acceleration that is not. We have to include gravity as a force, since we are measuring forces. There is no way of separating the two forces. This is both a blessing and a curse, as this means that an accelerometer at rest[33] can be used to find a vertical reference. It is a curse because it is not possible to determine from an accelerometer alone that it is at rest.

By combining the two equations, we end up with a relationship for acceleration:

$$a = \frac{k}{m}X - g$$

The spring constant k and the inertial mass m are constant. X is the measured displacement.

We have removed the vector notation form since we are only dealing with one sensitive axis. If we had put three of these together with their sensitive axis perpendicular to each other, we would have a complete system for measuring acceleration in all directions.

The force of gravity g may or may not be present on the sensitive axis. If the sensitive axis was aligned parallel to the ground, the force of gravity measured by the accelerometer would be zero. Conversely, if the sensitive axis was perpendicular to the ground, the accelerometer would measure one unit of g_n[34]. One unit of g_n is defined as $g_n = 9.80665 m/s^2$. The force of gravity is not constant, but depends of geographical location and altitude. We are actually measuring the direction to the centre of gravity of our Earth. However, when no other forces are present, gravity is a force that is always downwards pointing. We will use this fact to estimate a vertical reference to the ground. Output from a typical accelerometer will be expressed in terms of units of g_n.

Mechanical accelerometers are a well established and mature technology, and can be as simple as the spring example presented above. Mechanical means that they are based on mechanical principles. Other types of accelerometers are vibrating quartz, vibrating fibre optic, silicon accelerometers, to mention just a few. In this project, we will use MEMS accelerometers.

Mechanical accelerometers are either constructed as open-loop or closed-loop systems. The example presented above is an example of an open-loop device. An open-loop type accelerometer measures or senses acceleration directly, like the tension in the spring.

A closed-loop accelerometer would try to counteract the deflection of the proof mass, and measure how much force is needed to keep the proof mass still. For example, if the proof mass was magnetic, an electromagnet would try to 'push' the proof mass back to its default position. In general, closed-loop accelerometers are often more stable and accurate than open-loop systems[35].

[32][1] pg. 154
[33]It is not accelerating in any direction.
[34]Not to be confused with the gravity vector
[35][1] pg. 161

Both closed- and open-loop accelerometers have errors in their output. Three important sources of error are fixed bias, scale-factor error, and cross-coupling errors[36].

Fixed bias errors, or simply bias, is a deviation in the output from zero when the accelerometer should be outputting zero. For example, if the accelerometer is at rest and the sensitive axis is aligned with the horizon. The accelerometer should output zero, and if it not, this is called fixed bias.

The scale factor error is a non-linearity in the output over the accelerometers range. When acceleration is applied accelerometer, the accelerometer should output a linear response. The deviation from the linear response is called scale factor error.

If an accelerometer picks up acceleration that is perpendicular to its sensitive axis this is called cross-coupling errors. This error is a result of manufacturing imperfections.

An accelerometer will have a range, usually called a full-scale-range. This is the maximum and minimum acceleration that the accelerometer can measure, and is usually expressed in terms of $\pm g_n$, for example $\pm 2g_n$ or $\pm 4g_n$. Go beyond these limits and the accelerometer will saturate, an increase beyond this will not be measured.

For a digital accelerometer, there is a lower limit to the resolution of the output. The least significant bit will represent the smallest difference between two measurements that the system can output. We use a digital accelerometer in this project.

The device specific specifications and parameters for the accelerometer we selected will be discussed in the method chapter.

2.4.2 Gyroscopes

Gyroscopes are devices that either measure or maintain rotation around its inertial space. There are several types of sensors which can be used to measure angular displacement or angular velocity, such as; ring laser gyroscopes (RLG) and fibre optic gyroscopes (FOG), electrostatic gyroscopes (ESG), and vibrating gyroscopes, to mention just a few technologies. Measurements can be either angular displacement or angular velocity. Strictly speaking, a sensor which measures angular velocity is <u>not</u> a gyroscope but an angular rate sensor. However, the term gyroscope has come to mean both an angular rate sensor and an angular displacement sensor. When mentioning gyroscopes, we will from now on mean either, but specify which we are talking about is there is any ambiguity.

We will continue this section by introducing a conceptual model of a simple gyroscope. Figure 6[37] shows a simple model of a mechanical spinning mass gyroscope. The rotor in the figure is spinning around the s-axis, the rotor is not mass-less. The rotor is free to rotate around axis t and p, through what is known as gimbals. The outer gimbal is then connected to an outer casing. The orientation of the rotor can be read of the angle pick-off.

[36][1] pg. 156

[37]Image from [1] pg. 61

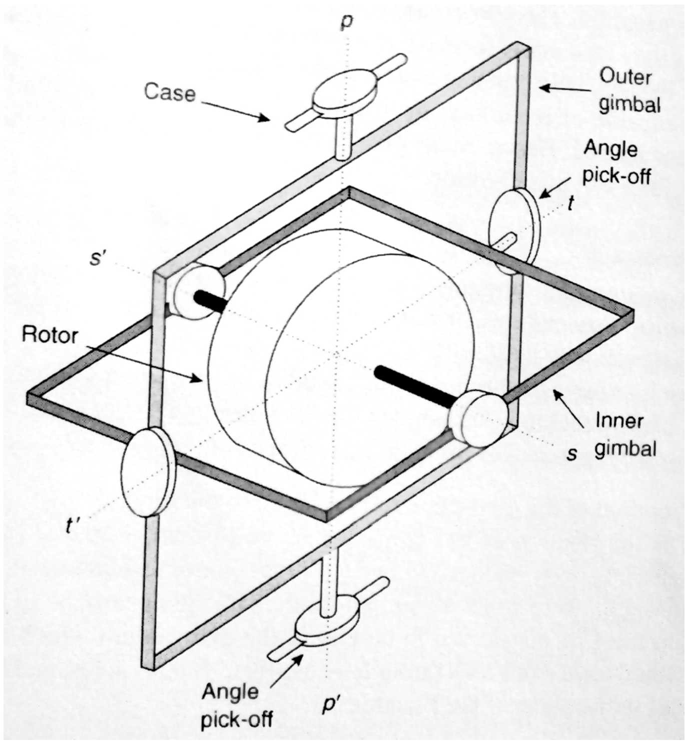


Figure 6: Conceptual Model of a Gyroscope.

Because of the inertia of the spinning rotor, when rotating the gyroscope's outer casing, the rotor will try to maintain its orientation. This is called gyroscopic inertia, and is because of the rotors angular momentum and precession[38].

First we present angular momentum. Angular momentum is the momentum of a spinning or rotating body given by H. I is its moment of inertia, and ω_S its angular velocity. The angular momentum gives the rotors resistance to stop spinning.

$$H = I\omega_S$$

Second we present precession, which is the rotation of the gimbal relative to inertial space. Mathematically, precession can be expressed as torque $\vec{T}$ through angular momentum vector $\vec{H}$ and angular velocity ω:

$$\vec{T} = \omega \times \vec{H}$$

Like an accelerometer would not react to rotation, a gyroscope would not react to linear acceleration. A gyroscope at rest[39] would not react to the force of gravity. These properties will be discussed further in a later section, and is important for inertial motion measurements.

[38]For a full mathematical explanation of angular momentum and precession, we recommend reading pg 60-67 in [1].

[39]not rotating

The range in angular rate gyroscopes is stated in degrees per second. Typical values are stated as $\pm 250°/s$ to $\pm 2000°/s$. If the gyroscope is rotated faster than this, the sensor will be become saturated.

For digital angular rate gyroscopes, sensitivity is expressed in bits per degree.

The gyroscope shares many of its sources of errors with the accelerometer; fixed bias, scale-factor, and cross-coupling errors. An additional important[40] sources of error specific to gyroscopes[41] is acceleration-dependent bias. This is the bias that is proportional to the applied linear acceleration.

Another concept in gyroscope technology is drift, which is not to be confused with bias. In this project we will use bias to mean a non-zero measurement when a gyroscope or a accelerometer is at rest. By drift, we will mean the divergence from the reference direction over time[42].

2.4.3 Magnetometer

Magnetometers measure the strength and the direction of magnetic fields. A three-axis magnetometer can be used to find the Earth's magnetic north, and since the Earth's magnetic pole is close enough to the Earth's true north, it can be used as a navigational aid. A common magnetometer technology is based on the Hall effect. Sensors based on this principle produce a voltage proportional to the applied magnetic field, and will also sense the fields polarity.

The Earth's magnetic field varies in strength, but for navigational purposes, it is the direction of the field that is of interest, and not its strength. Near the surface of the Earth, the magnetic field can be approximated by the field of a magnetic dipole, at a strength of roughly $1.3 \times 10^5 T$[43].

2.4.4 Barometer

Barometers are pressure sensors that are tuned to measure atmospheric air pressure. As you move vertically up, the pressure decreases. If you go down, pressure increases. With this you could estimate relative vertical displacement, called heave in marine terminology. The measured atmospheric pressure is not constant, so this would serve only as a relative measurement.

Such a sensor would need to be placed in air to work, and would not work in our setup were the sensor is under water.

2.4.5 Other Sensors

Global Positioning Systems (GPS), and recently the European Gallileo and the Russian GLONASS project, provide a global positioning system. From such a positioning system we can get absolute position, velocity and height above sea level. Velocity can be estimated between two consecutive positions.

Positioning systems are satellite radio based, and does not work under water, since the water will block these weak radio signals.

Another sensor that is worth mentioning in the context of this paper is the Doppler velocity logger (DVL) manufactured by Teledyne RDI[44]. This instru-

[40] To this project
[41] As listed in [1] pg. 72-73
[42] As used by [1]
[43] [2] pg. 572
[44] http://www.rdinstruments.com/navigator.aspx

ment uses acoustics to track the bottom, and calculates velocity based on this. The instrument is designed for under water operations, but needs to 'see' the bottom in order to work. This type of instrument is prohibitively expensive for this project.

2.5 MEMS Technology

MEMS is short for Micro Electro-mechanical Systems. These devices that are forged in processes adapted from technology used for making integrated circuits. Using materials like silicone, aluminium, and gold. By etching, depositing, lithography and layering, minute mechanical systems can be formed. These mechanical systems can be both complex, such as cogs and gears, and simple, consisting of little more that a silicone membrane. Any logic silicon[45] that is needed can be fused together with the MEMS part, as in figure 7[46].

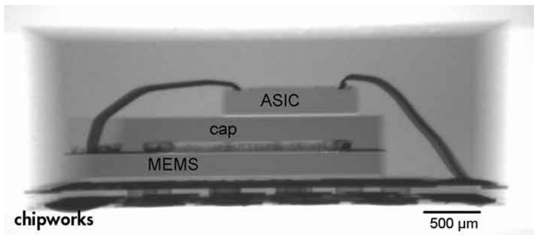


Figure 7: Side view of a typical MEMS device.

The advantages that MEMS holds over mechanical equivalent are[47]

- small size
- low weight
- lower cost (when mass-produced)
- lower complexity
- lower power
- better robustness
- high reliability

However, there are drawbacks. Because miniaturisation causes noise, non-MEMS sensors are less noisy and have a better bias. In time, it is expected that MEMS technology will approach perhaps even supersede their conventional counterparts[48]

[45]ASIC, Application Specific Integrated Circuit
[46]Used with permission from ChipWorks [http://www.chipworks.com/
[47][1] pg. 190
[48][1] pg. 191

2.5.1 MEMS Accelerometer

MEMS devices can be miniaturized versions of their mechanical counterparts. For example, a typical MEMS accelerometer is a miniaturised version of the proof-mass and spring. Figure 8 shows the LIS331DLH[49] accelerometer. Here we can see hinges, a proof-mass, and capacitive fingers. The capacitance fingers are an electronic part, and are used to sense displacement.

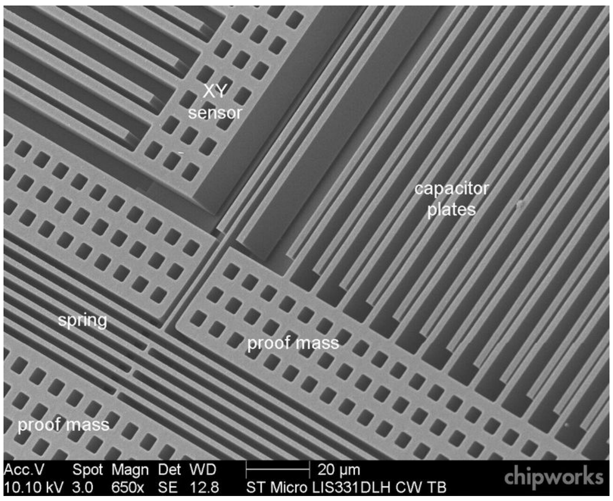

XY sensor
capacitor plates
spring
proof mass
proof mass
Acc.V Spot Magn Det WD 20 µm
10.10 kV 3.0 650x SE 12.8 ST Micro LIS331DLH CW TB
chipworks

Figure 8: Close-up of the LIS331DLH MEMS accelerometer.

2.5.2 MEMS Gyroscope

MEMS gyroscopes can also be miniaturized versions of a full-size gyroscope. Figure 9 shows a close-up of a MEMS gyroscope, the IDG300Q gyroscope from Invensense[50]. Again, the figure shows mechanical parts such as hinges and a proof-mass.

[49]Figure used with permission from ChipWorks [http://www.chipworks.com/

[50]Used with permission from ChipWorks [http://www.chipworks.com/]

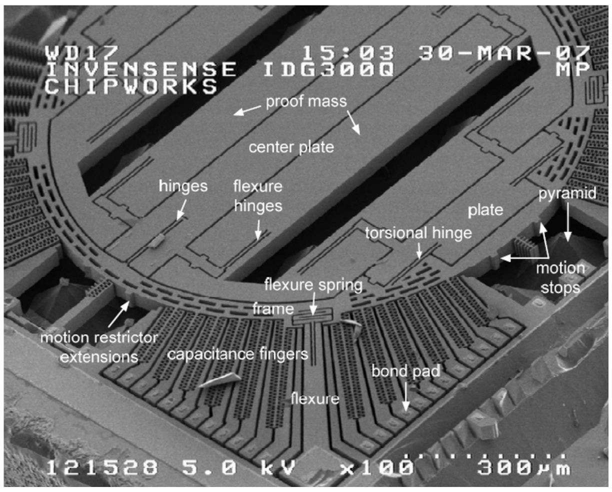


Figure 9: Close-up of the IDG300Q MEMS Gyroscope.

2.6 PID controller

PID controllers are process controllers. The P, I, D stands for Product, Integral, and Derivative respectivly. PID regulators have a measured process value and a desired[51] process value. The PID regulator will try to steer the process so that the measured value is equal to the desired process value. The difference between the measured and the desired process value is called the $e(t)$ error, and this error is used to steer the process.

$$P = K_P e(t)$$
$$I = K_I \int e(t)dt$$
$$D = K_D \frac{d}{dt} e(t)$$

The P term is simply the error multiplied by a factor K_P. The P term can be thought of as the current error. The I term is the integral of the error $e(t)$ multiplied by a factor K_I, and can be thought of as the historic error. The D term is the derivative of the error $e(t)$ multiplied by a factor K_D, and can be thought of as the projected future error. The sum of P, I, and D is the value that will be used to steer the process towards the desired process value. By adjusting the factors K_P, K_I, and K_D, the way the process is steered changes.

A general equation for a PID regulator can be stated as follows[52]:

$$e_control(t) = K_P \cdot e(t) + K_I \int_0^t e(t)dt + K_D \frac{d}{dt} e(t)$$

[51]called a set-point
[52][9] pg. 210

3 Method

Our goal is to build a system that can determine the attitude of a hydrophone, and by attitude we mean the orientation of the hydrophone. We want to use the attitude to compensate for the hydrophones movement to eliminate data outside of the hydrophone beam and to stabilise data the data that is inside the beam. This would increase accuracy in fish population estimates and other biomass estimations, especially in horizontal hydro-acoustic surveys. To do this, we need to determine the angles that the hydrophone is pointing in. We have built two units that can do that, and a control software that can be used with the second unit.

3.1 Considerations

3.1.1 Abbe Error

Abbe Error, also called sine error, is the magnification of angular error over distance. A $1°$ discrepancy in any one axis will at 1 meter result in a 1.745cm, and 17.45cm at 10 metres. When shooting horizontal sonar surveys, Abbe error can be particularly detrimental.

3.1.2 Real-time systems

A real-time system can be said to be a system that is *required to react to stimuli from the environment (including passage of time) within time intervals dictated the environment*[53].

Our project has to be aware of passage of time. Time is needed in calculating motion, and time is needed as a reference when outputting a new set of calculated attitude data. The system would not fail if it takes longer time to collect a new sample. By knowing the time at which a new sample of raw motion data was taken, we have made the system time-aware.

This project uses Linux as its operating system. In general, Linux is not a Real-time operating system (RTOS). There are possibilities of creating custom Linux kernels that are real-time. However, the Linux distribution available for our hardware is not real-time. In practise, it was not been a problem for us that the operating system was not real-time.

3.1.3 Bus Technologies

I2C is a serial bus technology developed and licensed by Phillips. The bus topology is multiple master and multiple slaves, which means that many *master* device can access many *slave* devices on a single bus. I2C uses two wires for bi-directional communication, one clock line, and one data line. The lines are open-drain. The name I2C is an acronym for Inter Integrated Circuit, which reflects its applications. It is mainly used for communication between devices over short distances, such as between devices on the same PCB. We use I2C to communicate with the sensors.

[53][7] pg. 2.

SPI short for Serial Peripheral Interface Bus is another popular serial bus technology seen in digital sensor devices. The application of SPI are mostly the same as for I2C, low speed data transmission over short distances. The digital sensor devices investigated in this project can communicate using I2C or SPI, or both.

UART short for Universal Asynchronous Receiver Transmitter, is another serial communication device. Although it is almost completely disappeared from personal computers, it can still be found in industrial applications. Two popular electrical standards of UART are RS-232 and RS-422. The RS-422 standard defines a maximum transmission distance of 1500 metres. RS-232 has no defined maximum transmission distance. In practise, RS-232 should be able to function over several tens of metres. In this project we chose to use RS-232 for data transmission over medium ranges. For very long ranges of hundred metres and longer, RS-422 would be an option.

3.1.4 Level shifters

Because there may be situations in which one device operates at 5 volts and another at 3.3 volts, we may need to convert between the two voltage levels. This is the situation for the Raspberry Pi single-board computer which we will use in this project. The Raspberry Pi is a 3.3 volt device, and it is not 5 volt tolerant. If a pin is fed a 5 volt signal, it would be permanently damaged.

An application note from Phillips[54] has an elegant solution to this problem using MOS transistors. Such a device is also bi-directional, so it is perfect for our use. Figure 10 is a schematic of the level shifter taken from the application note.

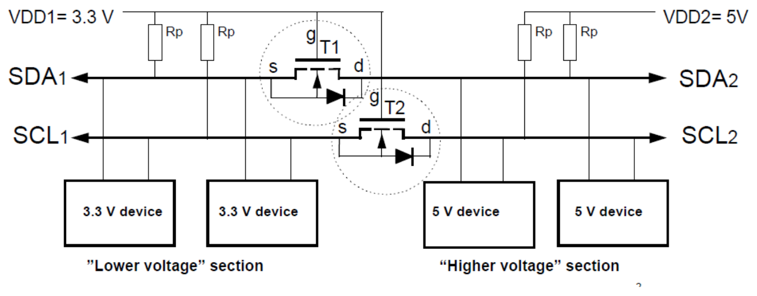


Figure 10: Bi-directional level shifter using MOS transistors.

3.1.5 Intellectual Property

The makers of the Raspberry Pi single-board computer, the RaspberryPI foundation, has pledged to make the RaspberryPI an open hardware platform. On the 20th of April 2012 they released the schematics for the first production version of the B model. The schematic is included in the appendix.

The operating system this project runs on is Raspbian, which is based on Debian. Debian is a free-software, and so are the tools and utilities used in this

[54] AN97055 Bi-directonal level shifter for I2C-bus and other systems.

project. The software wrote that runs on the Linux platform, was compiled using the GCC[55] compiler.

The software we wrote for the Microsoft Windows platform is compiled in Lazarus (IDE[56]), and is licensed under GPL[57].

All software written or used in this project is listed the appendix and included on the CD-ROM.

3.2 First approach to building an attitude system

As a pilot project we built a system based on an 8-bit micro-controller and an accelerometer sensor. Later we added a gyroscope sensor when we saw that an accelerometer alone could not reliably determine attitude. The system was modular and stackable, so that we could try out different sensors. We wrote a MS Windows program that read the raw data from the sensors via the micro-controller UART serial port.

3.2.1 Sensors

The first sensors that were investigated were the MMA8451Q accelerometer from Freescale Semiconductors, the LIS331DLH accelerometer from STMicroelectronics, and the IMU3000 gyroscope from Invensense. Common for these devices where that they were all digital, so they did not need an external analogue to digital converter (ADC). Further, the accelerometers and the gyroscopes could measure within the range we wanted. Table 3 is a summary of some of the parameters of these sensors. The parameters listed are the best achievable values with respect to low noise and high sensitivity.

Sensor	Parameter	Value
MMA8451Q	Range	$\pm 2g, \pm 4g, or \pm 8g$
	Sensitivity	4096 counts/g at $\pm 2g$
	Output Noise	$99\mu/\sqrt{Hz}$
LIS331DLH	Range	$\pm 2g, \pm 4g, or \pm 8g$
	Sensitivity	3.9 mg/digit at $\pm 2g$
	Output Noise	$218\mu/\sqrt{Hz}$
IMU3000	Range	$\pm 250°/s, \pm 500°/s, \pm 1000°/s, or \pm 2000°/s$
	Sensitivity	131 LSB/(°/s) at $\pm 250°/s$
	Output Noise	$0.1°/s(RMS)$

Table 3: Performance of the MMA8451Q, LIS331DLH, and IMU3000 sensors quoted in the respective data-sheets.

3.2.2 PCBs

Printed circuit boards (PCB) were created in StarCAD, and manufactured by El-lab at Blindern. All boards are 5cm by 5cm, and have matching mounting holes. Because of the fine pitched IC package of the sensors, the production was

[55]GNU C Compiler
[56]Integrated Development Environment
[57]GNU General Public License

done by an external PCB manufacturer. All sensor ICs used a Land-Grid Array (LGA) in either 16 or 20 pin packages, which is designed to be re-flow soldered.

The components were placed onto the boards by hand, and re-flow soldered with El-labs re-flow oven.

Figures 11, 12, and 13 show the sensor boards that were made.

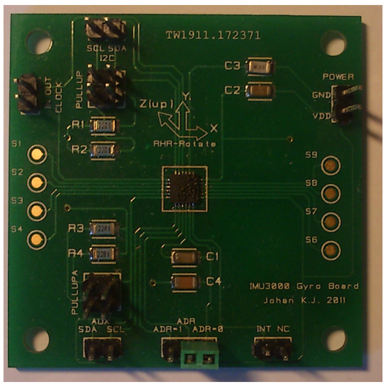


Figure 11: Invensense IMU3000 sensor PCB made for this project.

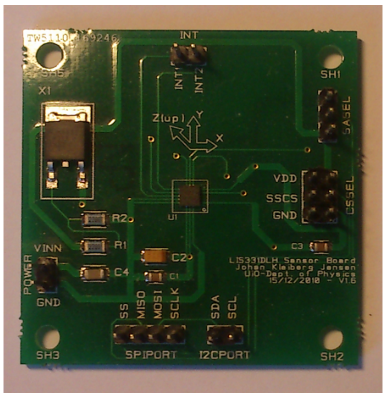


Figure 12: STMicroelectronics LIS331DLH sensor PCB made for this project.

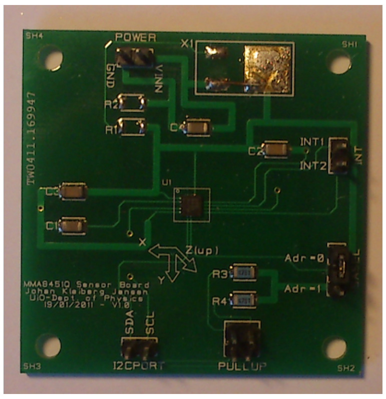


Figure 13: Freescale Semiconductors MMA8151Q sensor PCB made for this project.

The boards were made so that all functions of the sensors could be investigated and taken advantage of. The boards have configurable pull-up for the I2C bus, configurable address trough jumpers, external interrupt lines, and voltage regulator (except IMU3000).

Figure 14 and 15 shows the two boards that were made to interface with the sensors; an Atmel ATmega8 micro-controller board, and a combined serial and power board.

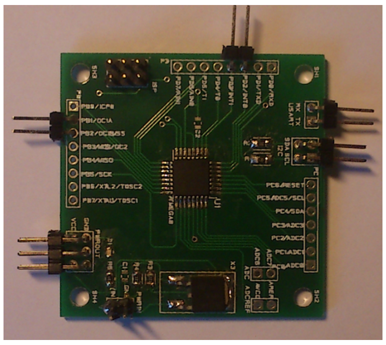


Figure 14: Atmel ATMega8 micro-controller PCB made for this project.

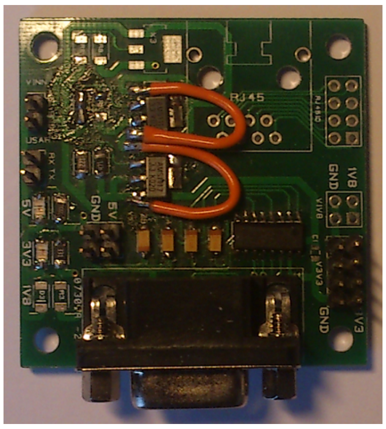


Figure 15: Power and serial PCB made for this project.

The ATMega8 PCB was of a simple design, and had only a voltage regulator and jumper selectable pull-up resistors for the I2C bus.

The Power and USART board regulated voltage from source to 5V, 3V3, and 1V8. This is because of the different voltage requirements of the sensors. The board also has a MAX3232[58] chip from Maxim, which is a TTL-to-USART chip. This is the serial line converter for the micro-controller.

The cards where designed to be stackable, as can be seen in figure 16. There were two reasons for this. Firstly to minimize axis non-alignment between the gyroscope and the accelerometer. Secondly, to have tidy electronics in which to build and enclosure around.

Schematics, PCB, and Gerber files are included on the DVD-ROM as listed in appendix A.

[58]Data-sheet is included in appendix.

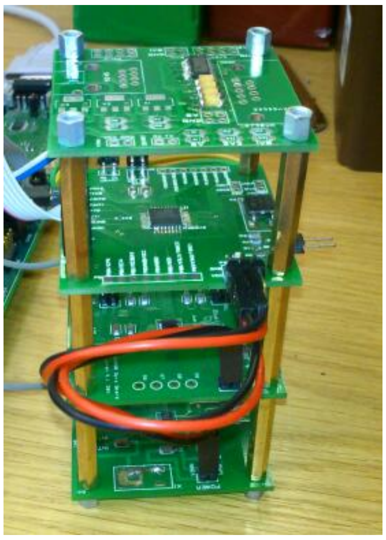


Figure 16: Sensors, micro-controller, and power and serial board stacked together.

Serial data that came up on the serial line from the ATmega8 micro-controller was raw data. The micro-controller would first set up the I2C and USART communications, and then wrote settings to the sensors via the I2C bus. These settings adjusted scale, sample-rate, filters and interrupts. The sensors triggered an external interrupt when data was ready to be read. The micro-controller would then read and convert the sensor readings from two eight-bit twos-compliment to numeric ASCII characters. These characters, together with comma separation and line feed, where written to the USART port. The source code for the micro-controller software is included in the appendix.

3.2.3 Testing

In 2011 we were fortunate enough to be able to test the accelerometers aboard the dive support vessel DSV Acergy Osprey. We did several recordings of data from the sensors during normal seas. We realised that an accelerometer alone could not reliably provide attitude data. The accelerometers were very noisy, and sway and surge got in the way of angle calculations. Below is a test we did on the dive support vessel DSV Acergy Osprey. Figure 17 shows the acceleration measured using the MMA8451Q accelerometer. Figure 18 shows the pitch and roll angles calculated from the acceleration measured by the LIS331DLH accelerometer.

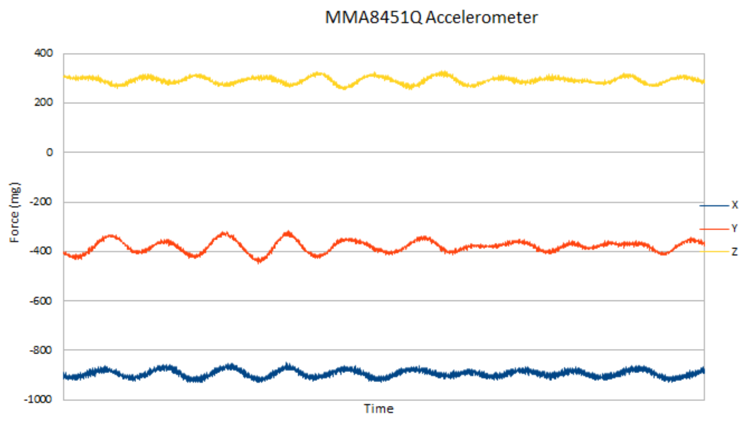


Figure 17: MMA8451Q accelerometer acceleration data recorded on the DSV Acergy Osprey.

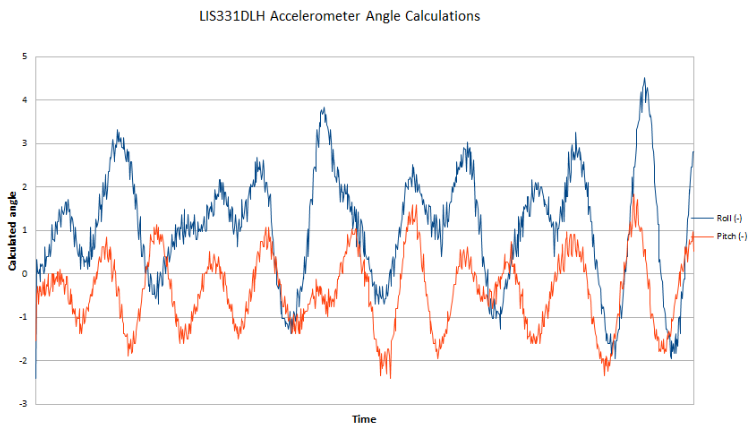
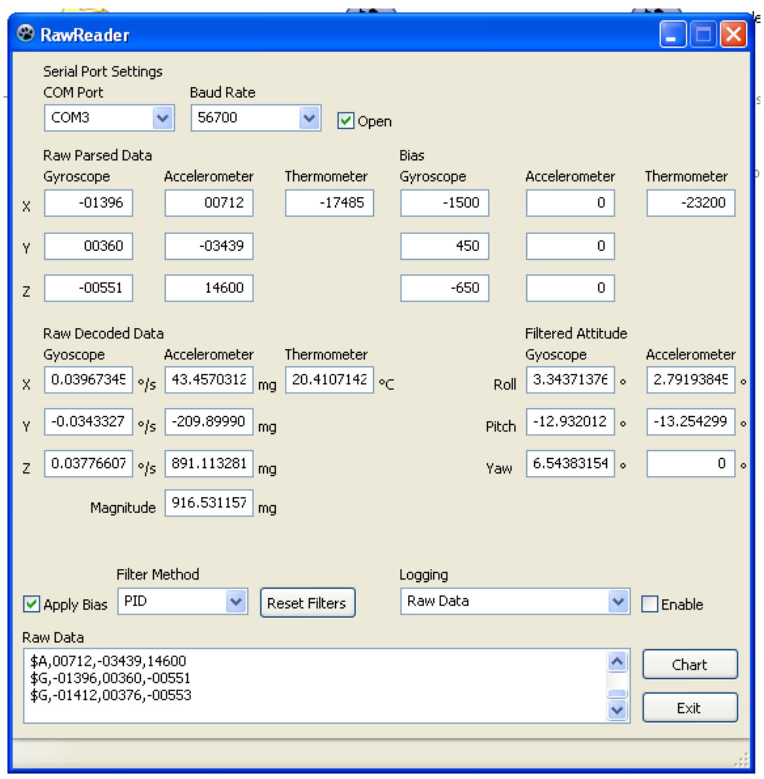


Figure 18: LIS331DLH accelerometer pitch and roll angle calculations recorded on the DSV Acergy Osprey.

After testing the accelerometers, we concluded that we needed a gyroscope to combine with an accelerometer to calculate reliable pitch and roll angles. We acquired a gyroscope, the IMU3000 from Invensense, and began testing this sensor.

We found one method for combining accelerometer and gyroscope data called a complimentary filter. To gain insight into the complimentary filter algorithm, a program was written that read the serial data from the COM-port and processed it in real-time. The program for was called RawReader, and was written in Lazarus Free Pascal. Figure 19 shows a screen-shot of this program as it decodes the data it is receiving. The source code and compiled software is included in the appendix.

3.2.4 Software to read sensors


RawReader

Serial Port Settings
COM Port: COM3
Baud Rate: 56700
[✓] Open

Raw Parsed Data

	Gyroscope	Accelerometer	Thermometer
X	-01396	00712	-17485
Y	00360	-03439	
Z	-00551	14600	

Bias

	Gyroscope	Accelerometer	Thermometer
X	-1500	0	-23200
Y	450	0	
Z	-650	0	

Raw Decoded Data

	Gyroscope	Accelerometer	Thermometer
X	0.03967345 °/s	43.4570312 mg	20.4107142 °C
Y	-0.0343327 °/s	-209.89990 mg	
Z	0.03776607 °/s	891.113281 mg	
Magnitude		916.531157 mg	

Filtered Attitude

	Gyoscope	Accelerometer
Roll	3.34371376 °	2.79193845 °
Pitch	-12.932012 °	-13.254299 °
Yaw	6.54383154 °	0 °

[✓] Apply Bias
Filter Method: PID
Reset Filters
Logging: Raw Data
[ ] Enable

Raw Data
```
$A,00712,-03439,14600
$G,-01396,00360,-00551
$G,-01412,00376,-00553
```

Figure 19: RawReader screen shot that decodes raw data from the sensors.

3.2.5 Results from testing

We learned that an accelerometer alone could not be used to determine attitude of a mobile platform. We needed a gyroscope sensor as well.

Secondly, we realised that the ATMEGA8 micro-controller was not powerful enough to run the complimentary filter algorithm. We still wanted the final sensor unit to run as a stand-alone system, so we began searching for another hardware system to run the algorithm on.

3.3 Second approach to building an attitude system

The second generation system consists of a powerful single-board computer together with a sensor that combines an accelerometer and a gyroscope in one IC package.

3.3.1 MPU9150 Sensor

In 2012 Invensense released a new sensor, the MPU9150. This sensor has an accelerometer, a gyroscope and a magnetometer all built into one chip package. This eliminates the problem of the axis of the different sensors coming out of alignment, secondly, this eases the interfacing to the sensor. Lower component count, lower power and lower complexity. When the MPU9150 was released it was not available as a single chip. An engineering evaluation board was available, and this was purchased. Figure 20 shows the evaluation board[59].

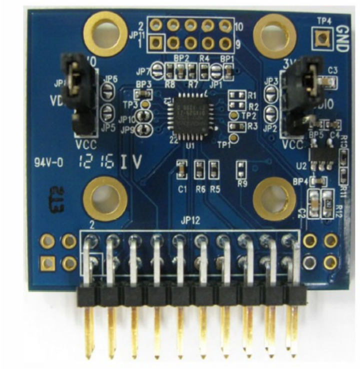


Figure 20: The MPU9150 Engineering Evaluation Board.

The board has all the required connections broken out on a pin header, which made it easy to interface. In addition the evaluation board had four mounting holes. With these holes, the board could be securely mounted to the enclosure.

An extract of some of the parameters of the MPU9150 sensor are listed in table 4. The values are taken from the data-sheet, which is available on the DVD-ROM.

MPU9150	
Accelerometer	Tri-axial
Gyroscope	Tri-axial
Temperature sensor	On-board
Magnetometer	Tri-axial AKM AK8975
Interface	I2C fast-mode (400kHz)
Operating temperature	-40C - 105C
Power	9.9mA at 3V3 (Full power)
Accelerometer Range	$\pm 2g, \pm 4g, \pm 8g, or \pm 16g$
Accelerometer Sensitivity	16'384 LSB/g at $\pm 2g$
Accelerometer Output Noise	$4mg$ (RMS)
Gyroscope Range	$\pm 250°/s, \pm 500°/s, \pm 1000°/s, or \pm 2000°/s$
Gyroscope Sensitivity	131 LSB/(°/s) at $\pm 250°/s$
Gyroscope Output Noise	$0.06°/s$ (RMS)

Table 4: Invensense MPU9150 quoted specifications.

In addition to the accelerometer and the gyroscope, the MPU9150 has an on-board temperature sensor and an on-chip magnetometer. The temperature

[59]The data-sheet and schematics for the evaluation board is included in the appendix.

sensor we used to investigate how the MPU9150s accelerometer and gyroscope reacted to changes in temperature. The temperature sensor measures the temperature on the IC die of the MPU9150. The on-chip magnetometer can be used as a compass to determine magnetic North.

3.3.2 RaspberryPi

In 2012 the Raspberry Pi Foundation released its first Raspberry Pi singe-board computer. The Foundations goal was to provide a cheap and versatile, yet powerful single-board computer. The board can be seen in figure 21[60].

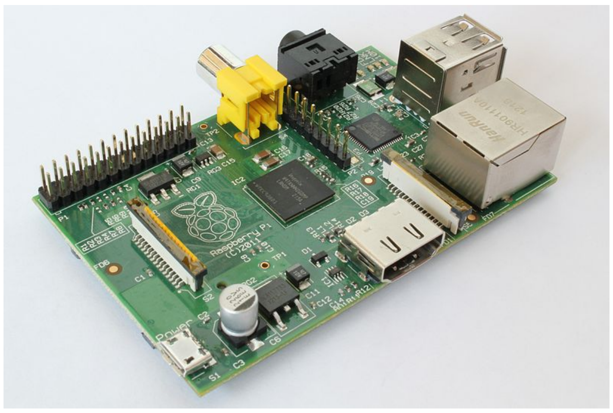


Figure 21: The Raspberry Pi single-board computer.

The heart and the brains of the Raspberry Pi board is the Broadcom BCM2835 System-on-Chip (SoC). This chip has an ARM core with floating point support running at 700Mhz. Table 5 lists an extract of some of the specifications of the Raspberry Pi.

	RaspberryPi B-Model
CPU	700 MHz ARM1176JZF-S on SoC
Storage	Secure Digital (SD) card
Power	5V 700mA through Micro USB
LAN	10/100 wired Ethernet RJ45
Peripherals	UART (2-pin), I2C, SPI and GPIO (all 3V3)
Size	85.60mm x 53.98mm

Table 5: Raspberry PI Hardware summary.

We thought that Raspberry Pi would be ideal for this project. It is powerful, cheap, has the required interfaces, small, and does not use allot of power.

[60]Picture from Wikipedia.org. [http://en.wikipedia.org/wiki/File:RaspberryPi.jpg]

3.3.3 Raspbian

The Raspberry Pi has its operating system and software stored on a SD[61] card, unlike a micro-controller which has its softwares stored in non-volatile memory. At present, there are a few available operating systems for the Raspberry Pi. We chose to run Rasbian as the operating system. Raspbian is a variant of the Linux OS Debian. It is compiled specifically to run on the Raspberry PI, and enables, among other things, support for hardware floating point mathematics. However, we needed to make some modifications to the operating system to suit our needs.

First we needed to disable the serial console to free the serial port. We need to free this device so that we can use this serial port to transmit data out of the system. We need to edit the file /boot/cmdline.txt, changing the line:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 kgdboc=ttyAMA0,115200
console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline
rootwait
```

to:

```
dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfs=ext4
elevator=deadline rootwait
```

And then delete the line in /etc/inittab:

```
T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

As we were early adopters of the RasberryPi, the default operating system, Raspbian, did not come with support for I2C. Luckily, someone wrote kernel drivers for this. We had to compile a new kernel to get the I2C working. In the latest release of Raspbian, I2C support is included in the kernel by default. We now use a kernel with I2C support in the kernel. There is a similar problem with GPIO[62] support, were there isn't hardware interrupt routines at the time of writing this. This is something that the Raspbian community is working on, and will at some point be included in the kernel. A project called WiringPi[63] has made good progress in making the GPIO-port accessible hardware.

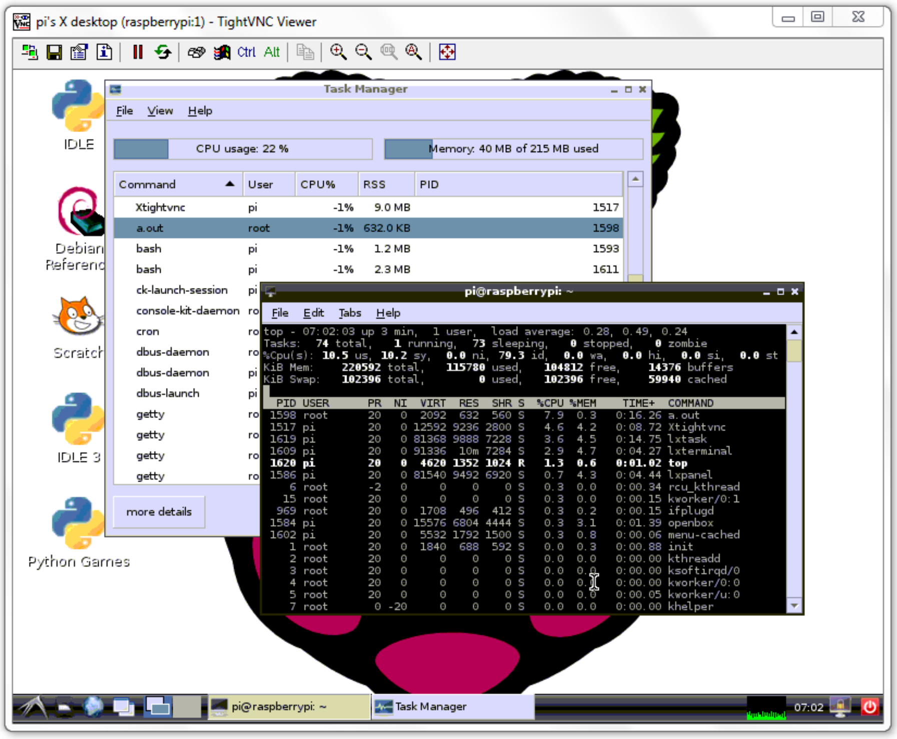
The attitude data is output on the serial port, but general access to the Raspberry Pi is done via Ethernet.

To access the Raspbian Pi we can open a remote shell using SSH[64]. To open a remote shell in MS Windows we used the free software Putty[65]. We have set up the Raspberry Pi with a fixed IP address of 192.168.1.109. The user login name is *pi*, and the password is *Johankj2012*. SSH comes installed with Raspbian by default.

We have also set up the Raspberry Pi to start up a GUI[66] session automatically. In MS Windows this can be accessed using ThightVNC[67], another free software. Access is on address 192.168.1.109:1, and the user name and password

[61]Secure Digital
[62]General Purpose Input Output, digital input and output from the RaspberryPi
[63]https://projects.drogon.net/raspberry-pi/wiringpi/
[64]Secure Shell
[65]Included in the appendix
[66]Graphical User Interface
[67]Included on the DVD-ROM

is the same as for the SSH login. Figure 22 shows the Raspbian GUI running on the Raspberry Pi accessed through a VNC connection.



Command	User	CPU%	RSS	PID
Xtightvnc	pi	-1%	9.0 MB	1517
a.out	root	-1%	632.0 KB	1598
bash	pi	-1%	1.2 MB	1593
bash	pi	-1%	2.3 MB	1611
ck-launch-session	pi			
console-kit-daemon	ro			
cron	ro			
dbus-daemon	ro			
dbus-daemon	pi			
dbus-launch	pi			
getty	ro			
getty	ro			
getty	ro			
getty	ro			

```
top - 07:02:03 up 3 min,  1 user,  load average: 0.28, 0.49, 0.24
Tasks:  74 total,   1 running,  73 sleeping,   0 stopped,   0 zombie
%Cpu(s): 10.5 us, 10.2 sy,  0.0 ni, 79.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:    220592 total,   115780 used,   104812 free,    14376 buffers
KiB Swap:   102396 total,        0 used,   102396 free,    59940 cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
 1598 root      20   0  2092  632  560 S   7.9  0.3   0:16.26 a.out
 1517 pi        20   0 12592 9236 2800 S   4.6  4.2   0:08.72 Xtightvnc
 1619 pi        20   0 81368 9888 7228 S   3.6  4.5   0:14.75 lxtask
 1609 pi        20   0 91336  10m 7284 S   2.9  4.7   0:04.27 lxterminal
 1620 pi        20   0  4620 1352 1024 R   1.3  0.6   0:01.02 top
 1586 pi        20   0 81540 9492 6920 S   0.7  4.3   0:04.44 lxpanel
    6 root      -2   0     0    0    0 S   0.3  0.0   0:00.34 rcu_kthread
   15 root      20   0     0    0    0 S   0.3  0.0   0:00.15 kworker/0:1
  969 root      20   0  1708  496  412 S   0.3  0.2   0:00.15 ifplugd
 1584 pi        20   0 15576 6804 4444 S   0.3  3.1   0:01.39 openbox
 1602 pi        20   0  5532 1792 1500 S   0.3  0.8   0:00.06 menu-cached
    1 root      20   0  1840  688  592 S   0.0  0.3   0:00.88 init
    2 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S   0.0  0.0   0:00.00 ksoftirqd/0
    4 root      20   0     0    0    0 S   0.0  0.0   0:00.00 kworker/0:0
    5 root      20   0     0    0    0 S   0.0  0.0   0:00.05 kworker/u:0
    7 root       0 -20     0    0    0 S   0.0  0.0   0:00.00 khelper
```

Figure 22: Raspbian GUI accessed through a VNC connection.

The GUI session also serves another important purpose, and that is to automatically start any software when power is applied. When the Raspberry Pi starts up, it will automatically run the complimentary filter attitude program that we wrote for this project. To start the motion software we wrote a bash script that launches together with the GUI session. The script is located in:

```
/.config/lxsession/LXDE/autostart
```

3.3.4 Assembled Hardware

The whole system needed to be enclosed in a watertight box so that the electronics would be kept dry during normal use. We use a waterproof plastic container with a rubber seal and a waterproof connector. Figure 23 shows the container, the connector, and the cable.

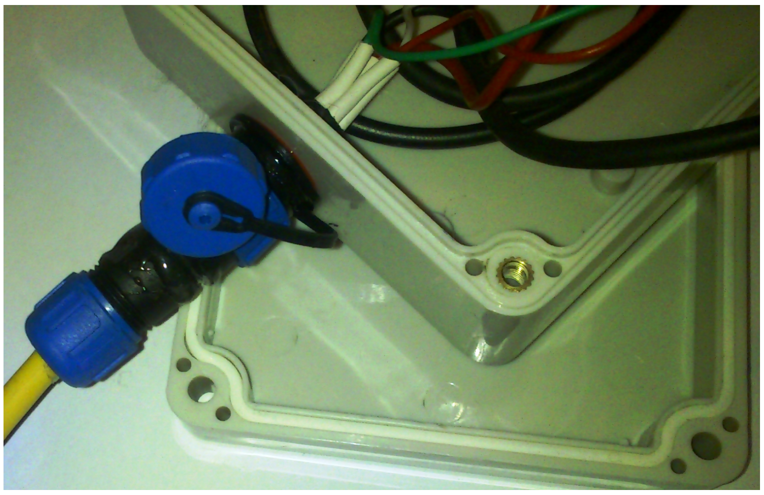


Figure 23: Waterproof enclosure and connector used for the attitude system.

The cable is a regular multi-stranded twister pair Ethernet cable, were two of the pairs are connected together for power. The connector has five pins, and are arranged according to table 6:

Pin	Function	Wire code
1	12 VDC	Brown/Brown White
2	Ground	Orange/Orange White
3	Signal ground	Green
4	Transmit	Green White
5	Receive	Blue White

Table 6: Cable and connector pin arrangement in external cable.

Figure 24 shows an exploded view of the system. The MPU9150 sensor board is screwed onto stand-offs through the mounting holes on the PCB. The stand-offs are glued to the enclosure with epoxy.

Other than the Raspberry Pi and the MPU9150 sensor board, we have a DC-to-DC converter that converts 12 volts down to 5 volts. We did not want to use a low drop-out regulator[68], because of the power loss in such devices.

We also have a USART board which converts the serial signal from the Raspberry Pi to true RS-232 signals. We also have a bi-directional level converter that converts the 3.3 volt signal from the Rasperry Pi to a 5V signal. The USART board is based on the Maxim MAX3232 IC[69] which can be used with 3.3 volts, but the level converter also acts as a buffer that protects the pins on the Raspberry Pi. Running the USART board at 5 volts also means increased voltage differential on the USART line, which in turn makes the line more robust.

[68]LDO

[69]Data-sheet is included in the appendix

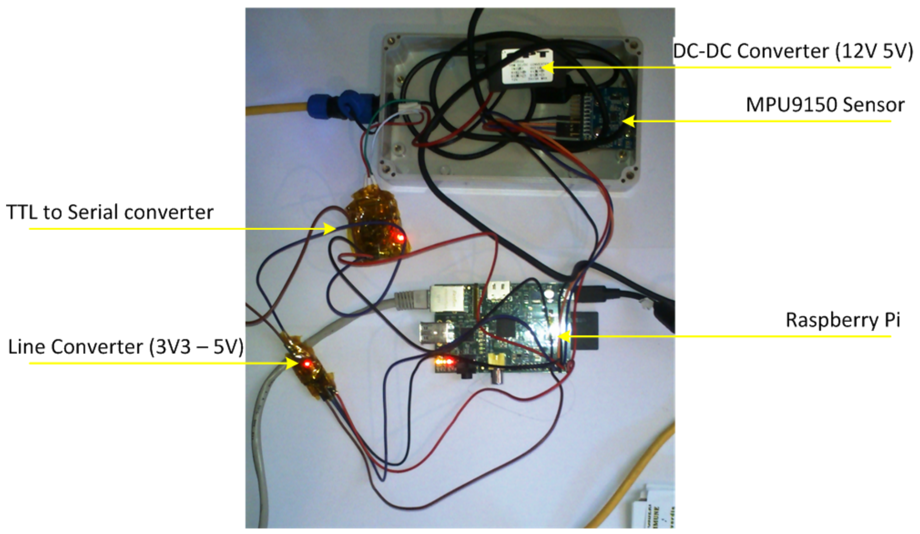


Figure 24: Exploded annotated view of the hardware components in the attitude system.

All the different parts were wrapped in Capton tape, a non-conductive thermal resistant tape, to prevent short-circuiting between components.

Figure 25 a diagrammatic representation of the hardware, with all the lines named. Table 7 describes the connections to the different headers on the devices.

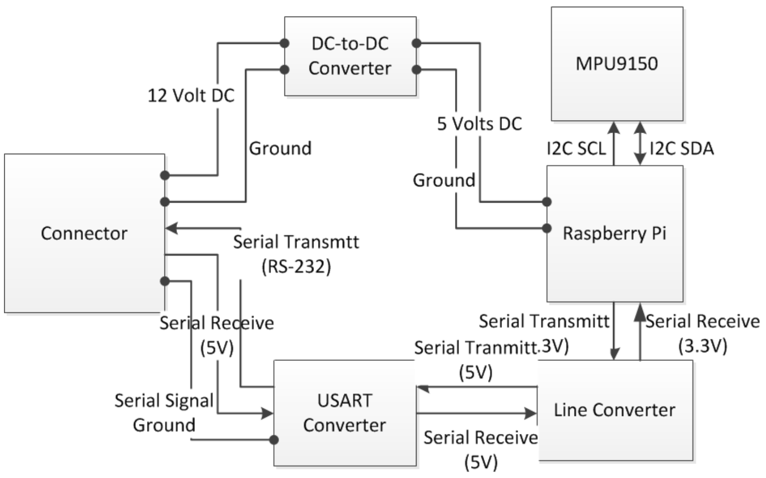


Figure 25: Diagramatic overview of the components in the attitude system.

Include table of internal connections here, from RPI to MP9150EVB and USART. Including power and ground.

Pin in (RPI)	Pin out (Device)	Function
P1/3	JP12/22	I2C SDA
P1/5	JP12/20	I2C SCL
P1/14	-	UART TX
P1/15	-	UART RX

Table 7: Internal connections between components.

3.3.5 Price and availability

All the components used are off the shelf and easily obtainable. Our goal was also to assemble a low cost system. Table 8 is a summary of the cost of all the components used. The cost is in USD, since most of the components were paid for in that currency. We have not included shipping costs.

Part	Price (USD)
Raspberry Pi B-model	35 USD
MPU9150 Engineering board	79.95 USD
DC-to-DC converter	5.50 USD
USART Converter	3.28 USD
Line Converter	2.46 USD
Water-proof enclosure	10.93 USD
Water-proof connector	8 USD
Cable	free
Total	142.12 USD

Table 8: Total component costs.

The simpler A model of the Raspberry Pi costs 25 USD. It has no Ethernet port, and only a single USB port. The MPU9150 sensor on its own costs 17 USD to purchase.

3.3.6 Current Consumption

Another important consideration of the system is how much power it draws, since the system is powered of a battery in the field. Table 9 lists the maximum current consumption taken from the data-sheets where known.

Device	Current
Raspberry Pi	700 mA
MPU9150	9.9 mA
MAX3232	1.0 mA
Line converter	Unknown
Other parts	Unknown
Total	710.0 mA
Measured	182.5 mA

Table 9: Quoted current draw of components and measured total current draw.

The stated efficiency of the DC-to-DC converter is up to 96 percent. During the field test at Sognsvann we powered the system from a 12 volt 7Ah sealed lead acid battery for several hours.

We measured the current draw from the system to be 182.5 mA using a multimeter.

3.3.7 Attitude estimation

An accelerometer by itself can be used to estimate pitch and roll angles, since an accelerometer can be used to measure the force of gravity. The angles can be estimates by taking the cosines of the forces as measured on the axis, as in the figure 26:

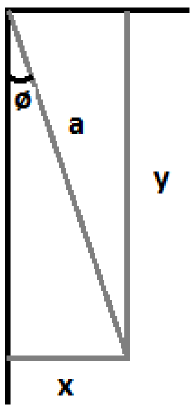


Figure 26: Angle calculation from acceleration data.

The angle can be calculated by taking the sine:

$$\phi = \sin^{-1}(\tfrac{x}{y})$$

However, an accelerometer can not differentiate between the force of gravity and acceleration. If acceleration is present, the roll and pitch estimate will not be correct.

Likewise, an angular rate gyroscope can be used to estimate roll and pitch angles. An angular rate gyroscope will measure angular velocity, so to get angular displacement we need only to integrate angular velocity once with respect to time. The time step, dt, is simply the time step between two samples:

$$\theta = \int \omega_\theta dt$$

If we sum the angular displacements, we can get the roll, pitch angles. The data from the angular rate gyroscope would not be susceptible to acceleration as with the accelerometer. The first problem with this approach is that we need to provide a starting point for the sum, a sort of *tarring*. The second problem is that bias and errors in the angular velocity data will cause the sum of angular displacements to drift.

If we could combine the data from the accelerometer and the gyroscope, we could have the cake and eat it too; an absolute reference provided by the accelerometer and angles provided by the angular rate gyroscope.

Our choice of algorithm for estimating roll and pitch angles is based on a so called complimentary filter, which will be presented in the next section.

3.3.8 Complimentary Filter

A top level data flow of our implementation of the complimentary filter can be seen in figure 27:

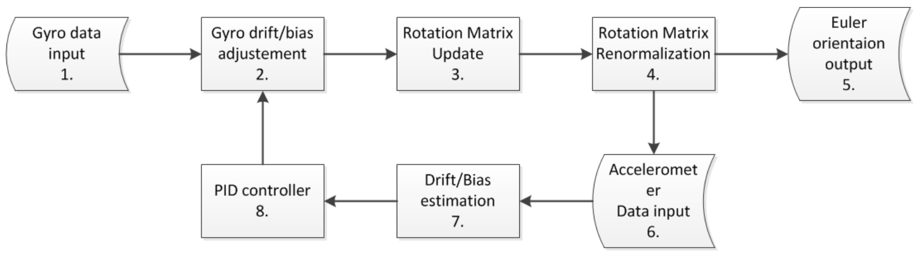

Gyro data input 1.
Gyro drift/bias adjustement 2.
Rotation Matrix Update 3.
Rotation Matrix Renormalization 4.
Euler orientaion output 5.
Acceleromet er Data input 6.
Drift/Bias estimation 7.
PID controller 8.

Figure 27: Complimentary filter top level data-flow.

We have enumerated the different processes so that they can be discussed.

1. Gyro data input. Here we convert the input from the angular rate gyroscope and accelerometer into a form we can use in the algorithm. In this project the sensors were all digital, and the sensor data were readily available to collect from the sensor. If the sensors were analogue, we would have to convert the reading using an ADC[70].

Next we convert the raw sensor data into real units, i.e. acceleration in (m/s^2) and angles in degrees (deg). This step will depend on the sensor model, and any options that is set with regards to scale and sampling rate.

To convert the raw acceleration data to real acceleration data we simply be scale linearly. The gyroscope data will need to be scaled and then multiplied by the time step, which is the same as the sampling rate.

2. Gyro drift/bias adjustement. In this step we apply corrections calculated in the previous cycle. The calculation of the corrections will be discussed in step 8. The corrections are added to the gyroscope data.

3. Rotation matrix update. Next we add the new gyroscope data to the rotation matrix that keeps track of the attitude. We do this by constructing what we have called an update rotation matrix $\bar{R}_{update}$, which based on the small angle theorem we introduced earlier.

$$\bar{R}_{update} = \begin{bmatrix} 1 & -\psi_n & \theta_n \\ \psi_n & 1 & -\phi_n \\ -\theta_n & \phi_n & 1 \end{bmatrix}$$

[70]Analogue to Digital Converter.

Where ϕ_n, θ_n and ψ_n are the gyroscope data from the previous step.

We can now multiply the two matrices together to get the updated rotation matrix:

$$\bar{R}_n = \bar{R}_{n-1}\bar{R}_{update}$$

4. Rotation matrix renormalization. Because repeated applications of small angle rotation matrices will skew the rotation matrix, we need to take steps to keep the rotation matrix orthogonal. We do this by exploiting that the rows in the rotation matrix should be orthogonal[71]. If the rows are orthogonal, the dot product between the should be zero. A non-zero value is the error scalar:

$$error = \begin{bmatrix} r_{11} r_{12} r_{13} \end{bmatrix} \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}$$

We then apply the error scalar to the two rows in opposite direction by cross coupling:

$$\begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix}_{orthogonal} = \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix} - \frac{error}{2} \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}$$

$$\begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}_{orthogonal} = \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix} - \frac{error}{2} \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}$$

With the two now orthogonal rows we calculate the third row by taking their cross product:

$$\begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \end{bmatrix}_{orthogonal} = \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix}_{orthogonal} \times \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}_{orthogonal}$$

Lastly, we scale the orthogonal matrices to ensure that their magnitudes are equal to one. We do this by dividing the sum of the squares of each row:

$$\begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix}_{normalized} = \frac{1}{\sqrt{r_{11}^2 + r_{12}^2 + r_{13}^2}} \begin{bmatrix} r_{11} \\ r_{12} \\ r_{13} \end{bmatrix}_{orthogonal}$$

$$\begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}_{normalized} = \frac{1}{\sqrt{r_{21}^2 + r_{22}^2 + r_{23}^2}} \begin{bmatrix} r_{21} \\ r_{22} \\ r_{23} \end{bmatrix}_{orthogonal}$$

$$\begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \end{bmatrix}_{normalized} = \frac{1}{\sqrt{r_{31}^2 + r_{32}^2 + r_{33}^2}} \begin{bmatrix} r_{31} \\ r_{32} \\ r_{33} \end{bmatrix}_{orthogonal}$$

These three normalised matrices now form the normalized rotation matrix $\bar{R}$.

[71]This method was developed by William Premerlani and Paul Bizard of the Gentlenav community

5. Euler Orientation output. Now we calculate the Euler angles based on the current rotation matrix. These are the pitch and roll angles which is the attitude estimate output of the algorithm.

$$\psi = \arctan(\tfrac{r_{32}}{r_{33}})$$
$$\phi = \arcsin(-r_{31})$$
$$\theta = \arctan(\tfrac{r_{21}}{r_{11}})$$

Where ψ, ϕ and θ is the roll, pitch and yaw angle respectively. Since the inverse tangens function has singularities, we could optionally use one of the other forms. But since this is only a problem at the poles, we can ignore this; a boat that is upside down will have larger problems to deal with.

6. Accelerometer data input. Here we simply take the accelerometer data and scale it to a unit vector. The result will be used as our horizontal reference in the next step.

$$\hat{g} = \parallel \vec{g} \parallel$$

7. Drift/Bias estimation. In this step we want to estimate the drift of the system, so we can counteract the drift in the next cycle. We start off by estimating the drift *error* by comparing the current rotation matrix $\bar{R}$ with the accelerometer vector $\hat{g}$. The assumption here is that on the average, the accelerometer will give us gravity as a vector. I.e. the system will not accelerate in any one direction indefinitely, so accelerations will cancel out eventually. The exception to this is if the system is constantly turning in the same direction, for example going around in a circle.

We calculate the cross product between the Z-row in the rotation matrix with the unit vector from the accelerometer[72], and get a third vector that represents the drift error:

$$\vec{e}_{drift} = \hat{g} \times \vec{R}_z$$

If the gravity unit vector and the z-row of the $\bar{R}$-matrix are aligned, the resulting error vector $\vec{e}_{drift}$ will be a null vector. The larger the deviation between the two, the larger the error vector will be.

8. PID Controller. In the final step we want to use the error vector from step 7. to create a set of corrections to apply in the next iteration.

We then sum and multiply the estimated error from drift into two new vectors we call $\vec{C}_I$ and $\vec{C}_P$:

$$\vec{C}_P = K_P \cdot \vec{e}_{drift}$$

$$\vec{C}_I = \vec{C}_I + K_I \cdot \vec{e}_{drift}$$

This is a PID regulator where we use the PI parts, the proportional and integral terms, only. To use common process control terminology; The accelerometer output is our set-point, the drift error is the error, and the rotation matrix is the process variable.

The algorithm now has made a full pass, and will start back at step 1.

[72]assumed to be the direction of gravity

3.3.9 Implementing the Algorithm

We chose to implement the algorithm in C language using the GCC compiler. We chose this language because it is hardware near and it is known to us.

Figure 28 shows a top-level data flow diagram for the program we wrote to implement the algorithm.

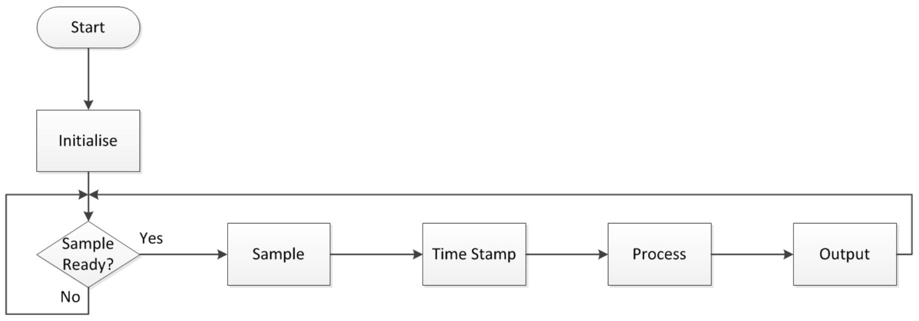


Figure 28: Top-level control flow of the attitude estimation system.

Initialize marks the start-up of the program. This is were we initialise the variables we are going to use. It is also were we set ut the devices we are going to use. This is the USART port, the I2C-device, and the log file. The log file is mostly for writing debugging information, and can be disabled by setting the option

```
DEBUG_MODE=false
```

in the header of the source code. This is a compile option. The USART port which is used to output data from the motion algorithm is set up with 57600 baud in a 8-N-1[73] configuration. The USART settings are coded into the source. The last device we set up is the I2C-device, namely the MPU9150 MEMS motion sensor.

After setting up the I2C-link, we write a number of options into registers[74] that determine the behaviour of the MPU9150. The register SMPLRTDIV is set to 49. The formula from which the output sample rate is calculated is

$$samplerate = (internalgyroscopeoutputrate)/(1 + SMPLRTDIV)$$

This which gives a sample rate of 20 Hz. Next we set the register DLPFCFG to 1. This sets the digital low pass filter for the accelerometer and the gyroscope to option 1[75]. This sets the internal gyroscope output rate to 1kHz. The accelerometer and gyroscope bandwidths are set to 184 Hz and 188 Hz respectively. This also introduces a delay for the accelerometer and gyroscope data of 2.0 mS and 1.9 mS respectively. The MPU9150 is set to a gyroscope range of $\pm 250°$ and $\pm 2g$ for the accelerometer at power-on, so we do not have to write to those registers.

[73] 8 bits, no parity, one stop-bit.

[74] Registers are named as in the device data-sheet, which is included in the appendix.

[75] See the MPU9150 register map pg. 13 for other modes. The register map is included in the appendix.

Sample is were we check if the MPU9150 has a new sample ready, and read that accelerometer and gyroscope data. The MPU9150 is capable of generating an external signal when a new sample is ready, but the Raspberry Pi did not have this feature implemented in the time of writing this. We read the DATARDYINT register bit instead of using the external interrupt. This register bit is set to 1 when a new sample is ready. We read the accelerometer, gyroscope and temperature at the same time.

Time stamp is exactly that. We record the Raspberry Pi system time at this point. This time will be output together with the attitude and temperature data.

Process is the implementation of the complimentary filter as described in the theory section.

Output is where the sample time, temperature and calculated attitude is output on the USART.

3.3.10 Serial Data Output Format

The serial data will be output at a predefined baud rate. By default the baudrate is set to 57600 *baud*, in an 8-1-N configuration (8 data-bits, one stop-bit, and no parity). A new data string is sent as soon as updated data has be calculated. Data can be sent both ways. On an electrical level the data will be output using the RS232 standard, but in a three wire configuration (No handshaking). In this way, the data can be carried for a reasonable length (tens of metres).

The USART serial line is a relatively slow bus, with a limited transmitting range. The higher the bit-rate, the shorter distance the serial line can transmit error free. This is because of the unbalanced nature of the RS232 bus. While we want to keep the bit-rate low, we don't want to saturate the bus[76].

Serial data for marine electronics commonly use the NMEA 0183 protocol, or its successor NMEA 2000. This is a proprietary and licensed protocol, but we used it as a basis for the first serial data output format. The NMEA 0183 protocol defines an attitude sensor ASCII string $PASHR. As with all NMEA strings they follow a specific format. The NMEA string has a maximum length of 71 bytes, including sign bits and carried return/line feed. The number of bytes we have available per sentence must be more than 71 bytes. The available data output-rates for common standard baudrates is listed in table 10. We chose a baudrate of 57'600 as a good trade-off between data-rate and transmitting distance. At at data-rate of 50Hz, we can transmit 144 bytes per sentence, and are therefore left with spare capacity. The final system outputs at a data rate of 25Hz.

[76]Trying to push data to the bus faster than the bus can transmit it

Baud Rate	100 Hz	50 Hz	25 Hz	10 Hz
9600	12	24	48	120
19200	24	48	96	240
38400	48	96	192	480
57600	72	144	288	720
115200	144	288	576	1 440

Table 10: Baud rates versus byte throughput.

The data fields in the PASHR NMEA string are listed in appendix B.

After the field test at Sognsvann, we decided to use a custom data format instead of the NMEA string. The way we used the NMEA string meant we had allot of empty fields, so we took away those, and the string identifier.

The revised and custom data-format is much shorter. Table 11 lists the new output data-format implemented:

No.	Name	Description	Format	Example
1	Time	Time-stamp of data	hhmmss.mm	150822.12
2	Temperature	Temperature at the sensor	[±]ccc.c	14.2
3	Roll	Roll in degrees	[±]ppp.pp	3.66
4	Pitch	Pitch in degrees	[±]rrr.rr	-3.95

Table 11: Custom data output string

The data in the custom string is comma-separated and line-terminated, just as the NMEA string. The maximum number of characters that will be sent in this format is 37 characters. We decided to keep the baud-rate of the serial port at 57'600 baud, since this worked well at the Sognsvann field test.

3.3.11 Other Software

To write the initial Raspbian OS onto an SD card we use the free software Win32DiskImages. Both this software, and the official Raspbian OS image is included on the DVD-ROM.

To copy an existing OS from an SD card we use the free software HDDRaw-Copy. This software, and the latest disk image is included in on the DVD-ROM.

A free serial emulator called RealTerm is also included on the DVD-ROM. This software can read and log data coming from the serial port of the Raspberry Pi. This is what we used before we wrote the control software.

3.4 Control Software

We wrote a program that can be used together with the attitude system. This control program can log the data coming from the attitude system to file. The control program can send settings to the attitude system. The control program will also show current readings from the attitude system, both numerically and graphically.

The control program is written in Lazarus IDE. The source code and a compiled program is included on the DVD-ROM.

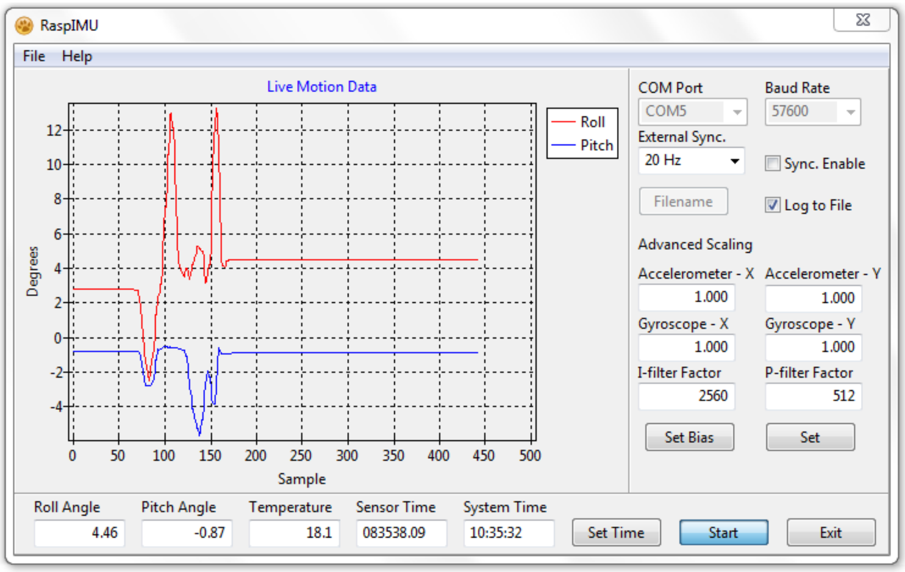
3.4.1 Description

The program can record the motion data from the system to a file, as well as displaying the current sensor time, pitch and roll, and sensor temperature.

The control software can send current system time down to the attitude system, for time synchronisation.

It is also possible to tune filter parameters and scaling factors in the attitude system. The bias can also be *nulled* by the control software.

In addition, it is possible to select an output rate and whether or not to send out an external syncing pulse. Figure 29 shows the user interface of the control program.



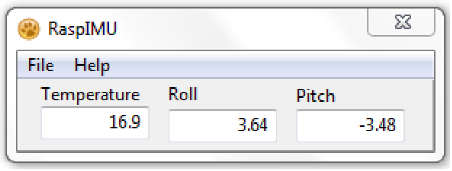
COM Port: COM5
Baud Rate: 57600
External Sync.: 20 Hz
☐ Sync. Enable
☑ Log to File

Accelerometer - X	Accelerometer - Y
1.000	1.000
Gyroscope - X	**Gyroscope - Y**
1.000	1.000
I-filter Factor	**P-filter Factor**
2560	512

Roll Angle	Pitch Angle	Temperature	Sensor Time	System Time
4.46	-0.87	18.1	083538.09	10:35:32

Figure 29: RaspIMU control program user interface.

The control program window can be resized to only show the current pitch and roll, so that it is unobtrusive. Figure 30 shows a resized window.



Temperature	Roll	Pitch
16.9	3.64	-3.48

Figure 30: RaspIMU control program when resized to a small window.

3.4.2 Commands

RaspIMU can send commands to the Raspberry Pi on the UART port. The program that runs on the Raspberry Pi can then interpret these commands.

The different commands are listed in the table 12.

Data output	Function
TInn:nn:nn	Sends current system time
SY0 / SY1	Turns External Sync. signal off/on
SRn	Sets output rate of external sync. signal and data rate
GXn.nnnn	Sets gyroscope scaling x-axis
GYn.nnnn	Sets gyroscope scaling y-axis
AXn.nnnn	Sets accelerometer scaling x-axis
AYn.nnnn	Sets accelerometer scaling y-axis
KInnnn	Sets integral filter parameter
KPnnnn	Sets product filter parameter
SB	Sets gyroscope bias to current reading

Table 12: Serial commands sent by the RaspIMU control program.

4 Testing

We tested the system in four separate tests; a static test, two dynamic test, and a field test. The static test was performed to investigate the raw output from the sensors. The dynamic test was a dry-run to the real-world test, to see how well our system worked and to uncover any problems in a controlled environment. The real-world test was to see how well the system performed in a typical field test.

The tests in this section are all performed using the MPU9150 sensor on the Raspberry Pi.

4.1 Static Test

The static sensor tests were done to investigate the characteristics of the sensors when they are motionless. In this state, it is easier to investigate the characteristics of the sensors, such as accelerometer output noise and angular rate gyroscope bias. If we know something about the characteristics of the output from the sensors we can perhaps take steps to improve measurements, or at least get an indication of why we get the results we get. We expected measurements to be normally distributed. The angular rate gyroscope will have bias, and with a motionless system we can determine this bias. Both the angular rate gyroscope and the accelerometer will display some output noise, and again, since the sensor is motionless we can statistically determine these values.

With the sensor in a motionless state, we can also investigate characteristics of the algorithm we use to estimate the attitude angles.

It was observed that the sensor is sensitive enough to pick up footsteps, opening and closing of doors, and similar disturbances. We took care not to do these things while recording data from the sensor.

4.1.1 Static sensor testing

We waited a minute for the sensor to stabilise before we recorded the data. We recorded a few thousand samples of acceleration, angular rate, and temperature.

Figures 31, 32, 33, and 34 show the histogram for the accelerometer and the gyroscope with sampling at 50Hz. The raw data can be found on the DVD-ROM.

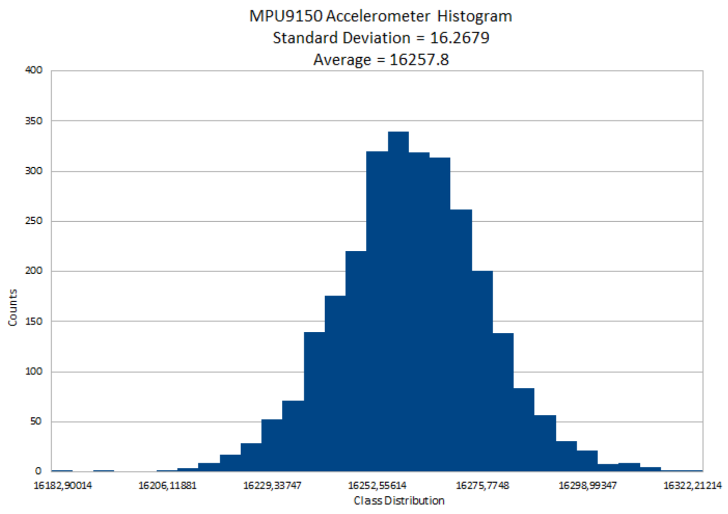


Figure 31: Histogram from raw accelerometer readings for the MPU9150 at 50Hz.

Figure 31 shows the histogram for the accelerometer. Here we use the length of the vector, and not independent axis, as for the gyroscope below.

We calculated that the average for the accelerometer is 16'257.8 raw units, and the standard deviation is 16.2679 raw units. To get the results in terms of the constant, g, we divide by the number of bits to the g[77]. We end up with 0.992297g average, and a standard deviation of 0.992914mg. We would expect the average to be close to one g, which it is. We are mostly interested in the direction of gravity, and not its magnitude. This is because we use the direction to estimate a horizontal reference.

The standard deviation is close to 1mg. This is of more importance to us, as this will say something about the noise present in the accelerometer data. We also note that the histogram looks normally distributed, i.e. it is not skewed. This is good, and expected for an accelerometer sensor.

[77]From MPU9150 data-sheet, which depends on selected range. 16'384 for $\pm 2g$

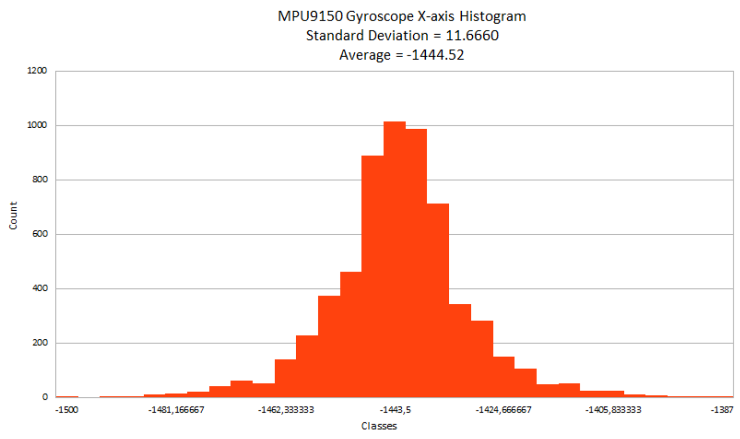


Figure 32: Histogram from raw Gyroscope X-axis readings for the MPU9150 at 50Hz.

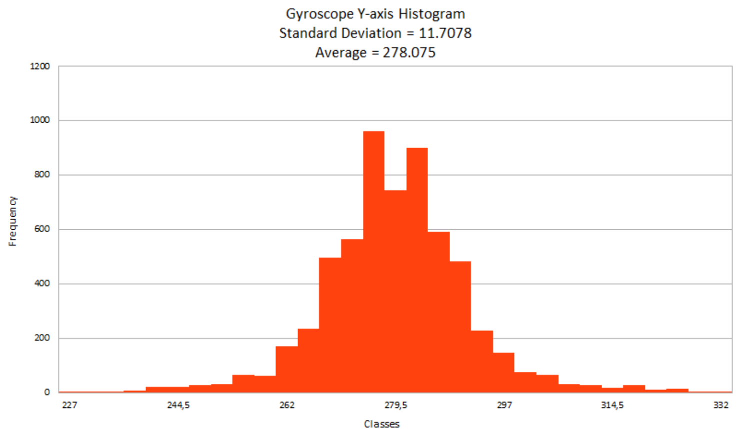


Figure 33: Histogram from raw Gyroscope Y-axis readings for the MPU9150 at 50Hz.

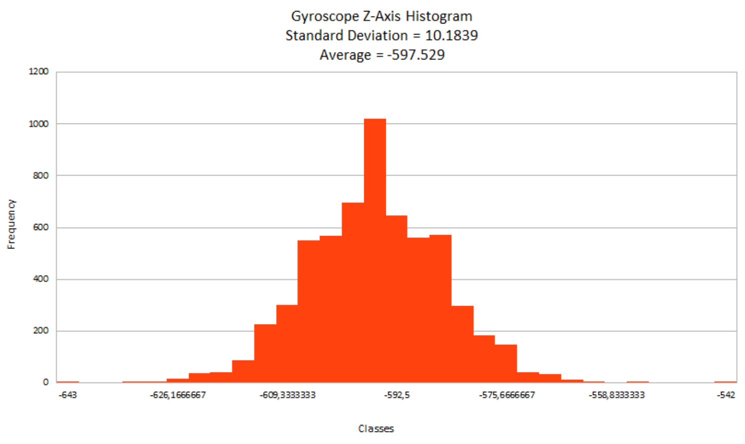


Figure 34: Histogram from raw Gyroscope Z-axis readings for the MPU9150 at 50Hz.

Figure 32, 33, and 34 show histograms from of each axis of the gyroscope. On each histogram the average and standard deviation is noted. We see that the gyroscope does not measure an average of zero, but take on a non-zero value. This is the bias of the gyroscope. The standard deviation, as for the accelerometer, say something about the noise in the gyroscope. We divide the raw data by the number of bits to $°/s$ [78]. Table 13 is a summary of the results stated in $°/s$.

Axis	Average	Standard Deviation
X	$-11.0269°/s$	$0.089053°/s$
Y	$2.12271°/s$	$0.089373°/s$
Z	$-4.56129°/s$	$0.077740°/s$
Accelerometer	$0.992297g$	$0.992914mg$

Table 13: Standard Deviation and Average for Gyroscope and Accelerometer for the MPU9150 at 50Hz.

We repeated the test, but now we lowered the sampling rate from 50Hz to 25Hz. We did this to see the effects the sampling rate had to the data. Table 14 lists the results.

[78] From MPU9150 data-sheet, depends on selected range. 131 for $\pm 250°/s$

	Minimum	Maximum	Average	Standard Deviation
Temperature	16.33	16.47	16.41	0.029
Accelerometer X	-1264	-926	-1090.67	43.65
Accelerometer Y	-298	28	-134.08	44.03
Accelerometer Z	14626	15116	14868.45	63.41
Gyroscope X	-103	-58	-73.37	4.07
Gyroscope Y	79	140	92.27	3.62
Gyroscope Z	22	58	39.23	4.86

Table 14: Standard deviation and average of raw data for gyroscope and accelerometer for the MPU9150 at 25Hz.

	Average	Standard Deviation
Accelerometer X	$-66.57mg$	$2.66mg$
Accelerometer Y	$-8.18mg$	$2.69mg$
Accelerometer Z	$907.50mg$	$3.87mg$
Gyroscope X	$-0.56°/s$	$0.031°/s$
Gyroscope Y	$0.70°/s$	$0.028°/s$
Gyroscope Z	$0.30°/s$	$0.037°/s$

Table 15: Standard deviation and average units for gyroscope and accelerometer for the MPU9150 at 25Hz.

We notice that the standard deviation is smaller when the sampling rate is lowered, as is expected. We decided to keep the sampling rate at 25Hz.

4.1.2 Temperature Testing

In this test we wanted to see how the MPU9150 sensor reacts to changes in temperature. We cooled the sensor down to about 8 degrees centigrade. We then moved the sensor to a warm room, and let the sensor warm up gradually. When the temperature reached ambient room temperature, we heated the sensor further with a hot air gun. There are fewer data points between 16 and 22 degrees centigrade. This is because the hot air gun heated the sensor a little faster than anticipated. We moved the hot air gun further away for the remainder of the temperature test. Figure 35 shows the accelerometer during the temperature test, and figure 36 shows the gyroscope.

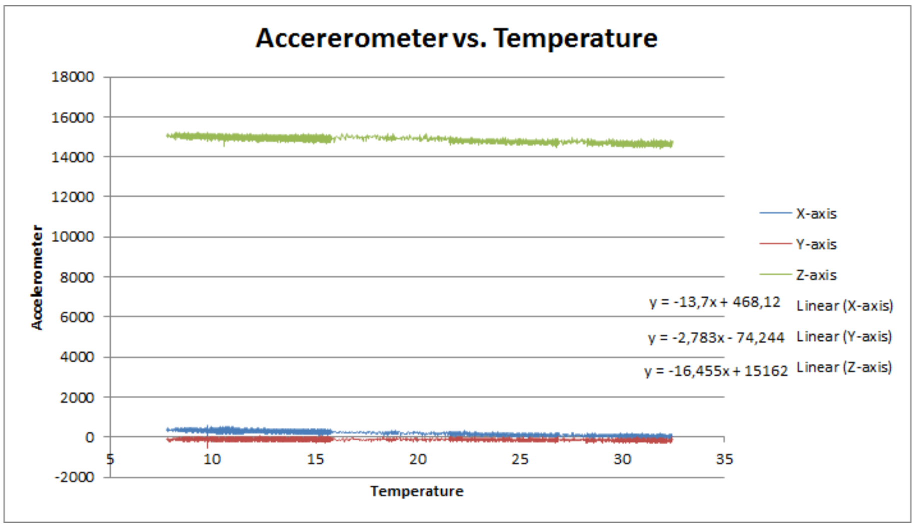


Figure 35: MPU9150 Accelerometer raw output versus temperature.

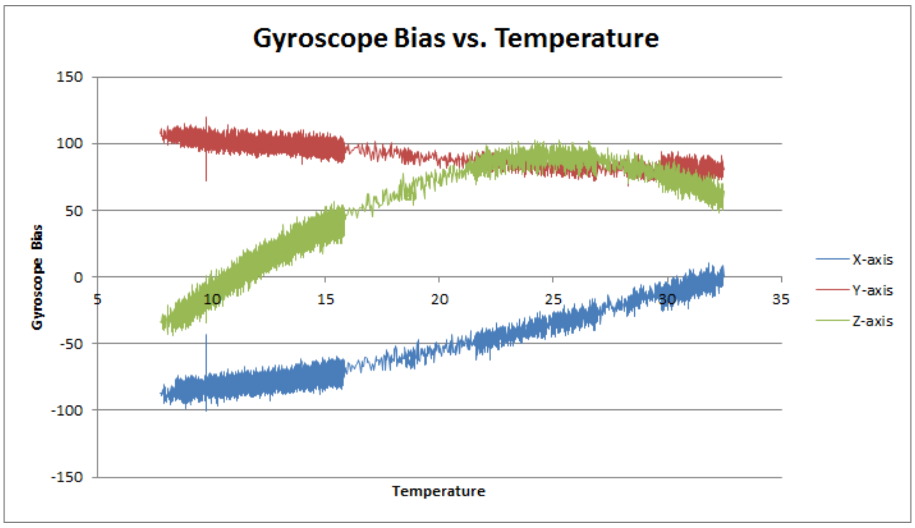


Figure 36: MPU9150 Gyroscope raw output versus temperature.

The accelerometer data is quite flat for the whole range of temperatures, however there is a small change. The changes we measured are listed in the table. We assume the trend is a linear, so we used a linear regression in a spread sheet to calculate the relationship. Table 16 summarised the relationship we found.

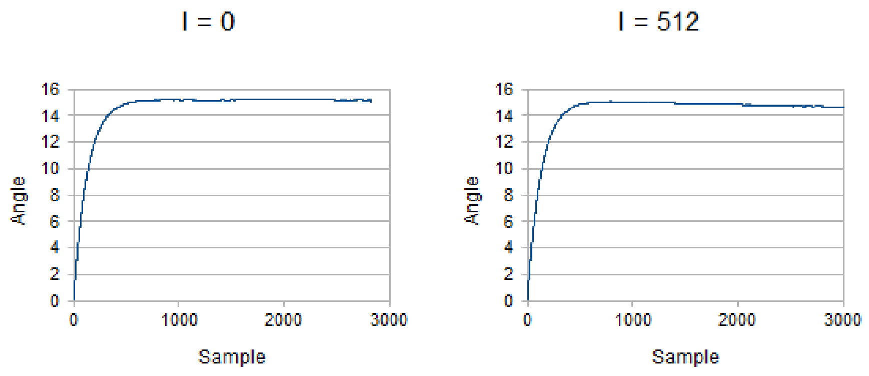
Axis	Raw units	Real units
X	$-13.7\frac{unit}{c^\circ}$	$-0.8361\frac{mg}{c^\circ}$
Y	$-2.783\frac{unit}{c^\circ}$	$-0.1698\frac{mg}{c^\circ}$
Z	$-16.455\frac{unit}{c^\circ}$	$-1.004\frac{mg}{c^\circ}$

Table 16: MPU9150 accelerometer output versus temperature relationship

The MPU9150 gyroscope shows a more complex reaction to temperature, perhaps a quadratic regression would be a good fit. We can remove this bias at one temperature point from the gyroscope data in our algorithm through the *Set Bias* command. The idea is that any slow change in bias after the *set bias* command will be compensated for in the complimentary filter.

4.1.3 Filter parameters and tuning

The complimentary filter has a PI-regulator with parameters that can be tuned. With the MPU9150 slanted at an angle, we changed the P and I parameter to see the effects on the attitude output. For each different setting og P and I[79] we restarted the algorithm. The data is included on the DVD-ROM. Figure 37 shows the attitude output for different setting of I, whilst holding P constant. Figure 38 shows the attitude output for different setting of P, whilst I is constant. For this test we did not zero the gyroscope bias, to see how well the filter would handle a non-zero bias. The factor is scaled for readability, both here and in the software. The real PI factors are divided by 65'535.



[79]Product term and Integral term.

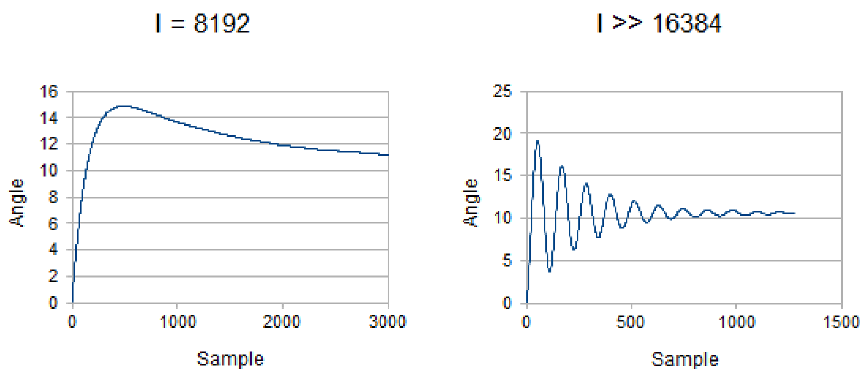


Figure 37: Complimentary filter PI-regulator tuning for different I factors.

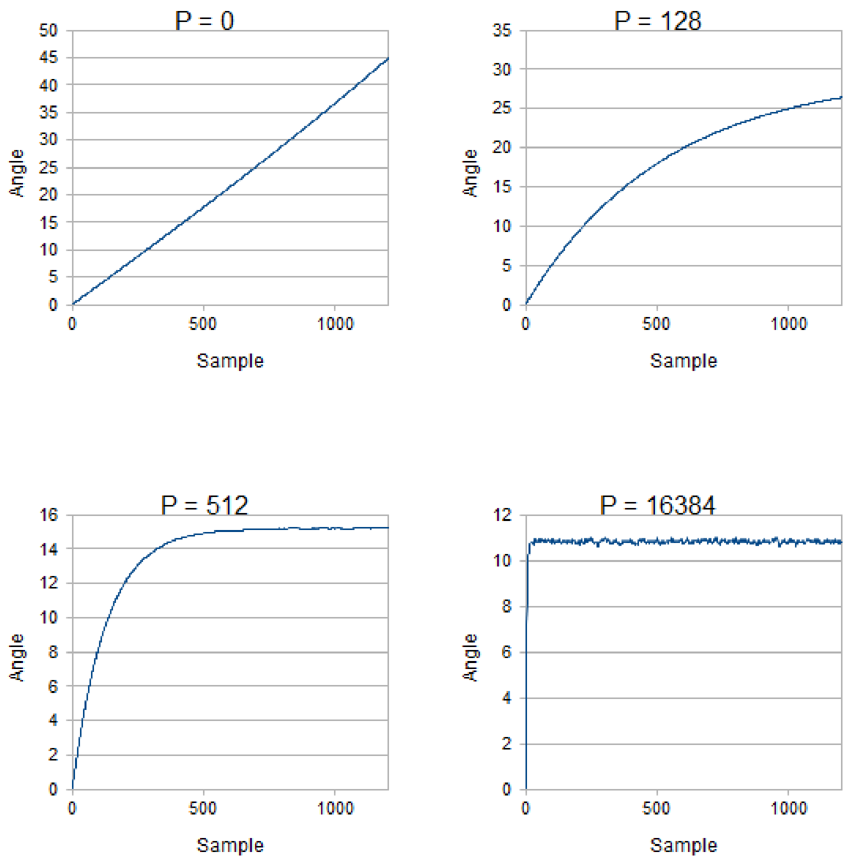


Figure 38: Complimentary filter PI-regulator tuning for different P factors.

We see that it is only when I approaches 8192 that the I factor takes effect. If the I factor is pushed even further, we get oscillations.

For the P factor, we see that when P is 0, the filter doesn't settle. When P is set to 128 and above the filter settles. If we keep increasing the we see that the filter settles very quickly, but that allot of the noise from the accelerometer comes through.

A P and I factor somewhere in between these extremes is what we use. We have set the default values for P and I are 512 and 512 respectively. These values for P and I worked well for us.

While we looked at the filter parameters, we also considered the settling time. The settling time is the time it takes for a system to reach a fraction of the final output value. This is interesting to us primarily to see how long it takes for the algorithm to go from its initial state, to a state we consider operational. Figure 39 of the output from the algorithm from its initial state up to 1000 samples.

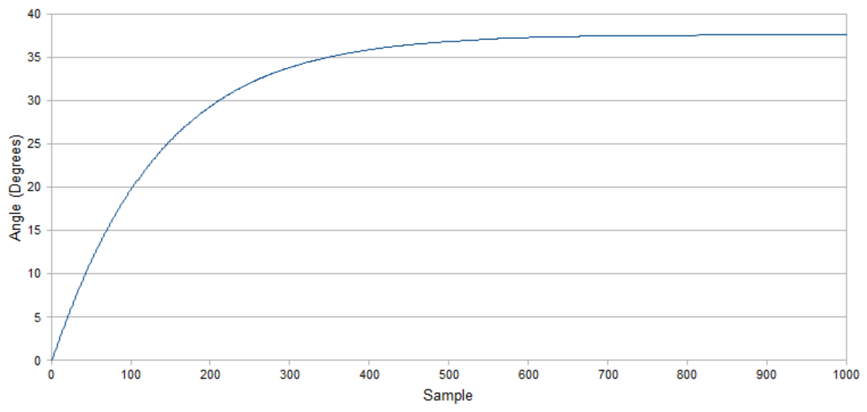


Figure 39: Typical settling time for the attitude estimation system from power-on.

The output reaches 99% of its final value after the 582. How long this takes in real time depends on the sampling rate and filter parameters, everything else being equal. For a sample rate of 10Hz the settling time would be 58.2 seconds, and 29.1 seconds with a sample rate of 25Hz.

4.1.4 Complimentary filter static test

Following the same setup as the static sensor test, we recorded the attitude output from the attitude system. Table 17 summarises statistics calculated from this test. We waited for the complimentary filter to settle before we started recording. We collected 9555 samples.

	Temperature	Pitch	Roll
Average	17.59 C	2.68°	−0.11°
Standard deviation	0.07 C	0.016°	0.032°

Table 17: Complimentary filter attitude output statistics in static test

Compared to the standard deviation in the raw output from the gyroscope in table 13 from the static sensor testing, we see that the deviation is of the same magnitude.

4.1.5 Observations from the static test

We have seen that the MPU9150 accelerometer and gyroscope data is normally distributed. We also saw that both the accelerometer and the gyroscope is sensitive to changes in temperature.

We manually tuned the PI-regulator in the complimentary filter, and found a set of good P and I filter parameters to use. For large P filter parameters that the noise from the accelerometer became evident. We have selected a P filter parameter that removes that noise.

When we recorded the attitude output from the complimentary filter, we saw that the standard deviation of the attitude was of the same magnitude as the standard deviation of the raw gyroscope output.

The standard deviation of the attitude from the complimentary filter will be our measure of precision, but only when the system is motionless. In the next tests we investigate the systems precision when it is in motion.

4.2 Dynamic Test

For the dynamic test we want the system to experience motion, and we want be able to compare the estimated attitude against an external reference. For our dynamic test, we used a laser pointer aimed at a flat wall. We used simple geometry to make an angular scale based on the distance between the wall and the sensor. The sensor itself was mounted on a hardware platform that was able to move in two independent axis.

The point of the dynamic test is to investigate how well the system is able to estimate attitude when in motion.

4.2.1 Setup

We assembled a hardware system that could move a platform in two independent axis, we call this the pan and tilt platform. The pan and tilt platform is moved to a position by two digital servos, which are able to move the platform in full 180 ° by 180 ° half-sphere. An Atmel ATMega8 micro-controller sends out a Pulse Width Modulated (PWM) signal to the servos to set the angle of the pan and tilt platform. Figure 40 shows the pan and tilt platform with servos.

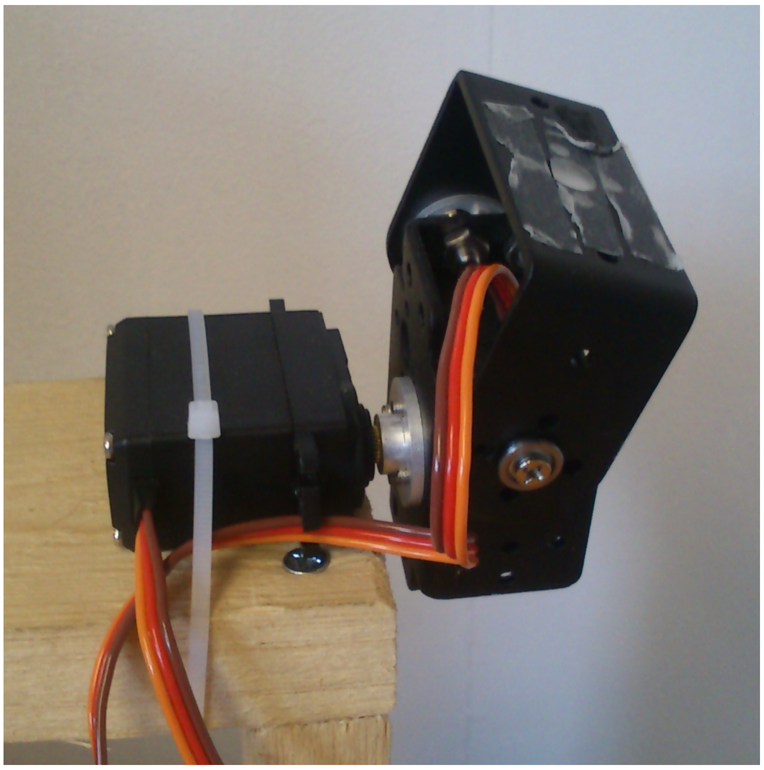


Figure 40: Pan-tilt test rig used in the dynamic test.

A servo will react to a PWM-pulse by trying to set the servo shaft to an angle based on the length of the pulse. Commonly servos expect a 20mS repetition rate, with a pulse length between 1mS and 2mS where 1.5mS is the neutral position. The two servos used were standard hobbyist servos (MSG995) and their position to pulse relationship can be found in table 18.

No.	Servo type	Min (0 °)	Neutral (90 °)	Max (180 °)	Pulse Rate
-	Common	1mS / $-90\circ$	1.5mS / 0°	2mS / +90°	20mS

Table 18: Servo PWM control signals for MSG995 servos.

A program was written for an ATMEGA8 micro-controller that generates PWM-pulses. The source code can be found in the appendix. The ATMEGA8 micro-controller was chosen because it can generate the PWM-pulse in hardware and it is relatively easy to do so. The program will the move the sensor platform in a known pattern, so that measurements and observations can be made. Since measurements are now done while the sensor is in motion, that tests are dynamic.

The sensor was mounted on the platform relatively close to the point of rotation, only offset by a few centimetres. Because of this we assume that the rotation is at the sensor, so that we don't have to take into account the centripetal forces during rotation. The centripetal forces will be larger the farther away the sensor is placed from the point of rotation.

The idea was to move the pan-tilt in different patterns, such as a sine or a sawtooth motion, or even simulating a boat. We would know the position of the platform through the PWM signal, which we could then compare against the measured attitude.

4.2.2 Data

The problem with this method was that the servos were no where near accurate enough for this purpose. The servos would *hunt* for the position that was sent from the micro-controller. The servos could not reliably reproduce the motion patterns we fed them either. The servos would miss the target angle by several degrees, so we had no repeatability. Figure 41 shows the attitude data for a *stepping* pattern. Typical servo *hunting* can seen in step number two and four.

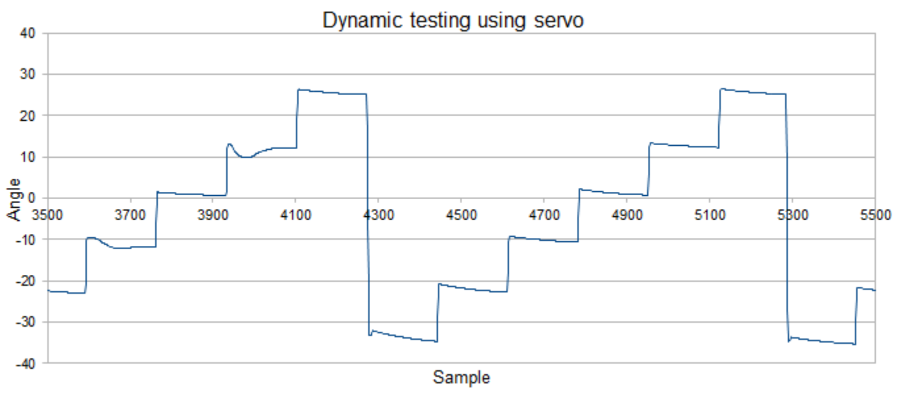


Figure 41: Step pattern test using pan-tilt servo rig as measured by the attitude system.

Figure 42 shows the attitude data for a triangle pattern. We could hear and see the servos hunting for position. We could pick up this jerky motion with the attitude sensor.

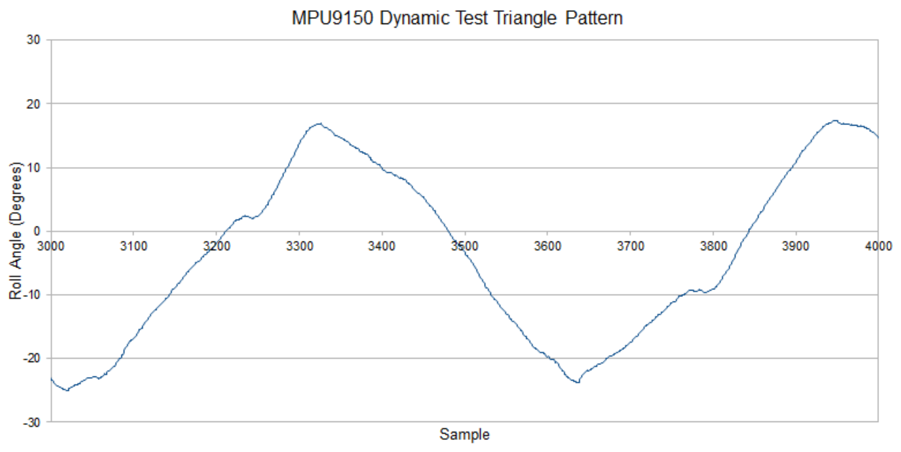


Figure 42: Triangle pattern test using pan-tilt servo rig as measured by the attitude system.

What we did in the end was to manually rotate the platform in small steps, and then recorded the projected angles. Figures 43 and 44 shows a plot for the pitch axis and the roll axis from one of these test. The pitch angle plot is overlayed with the pitch angle calculated from the raw accelerometer data.

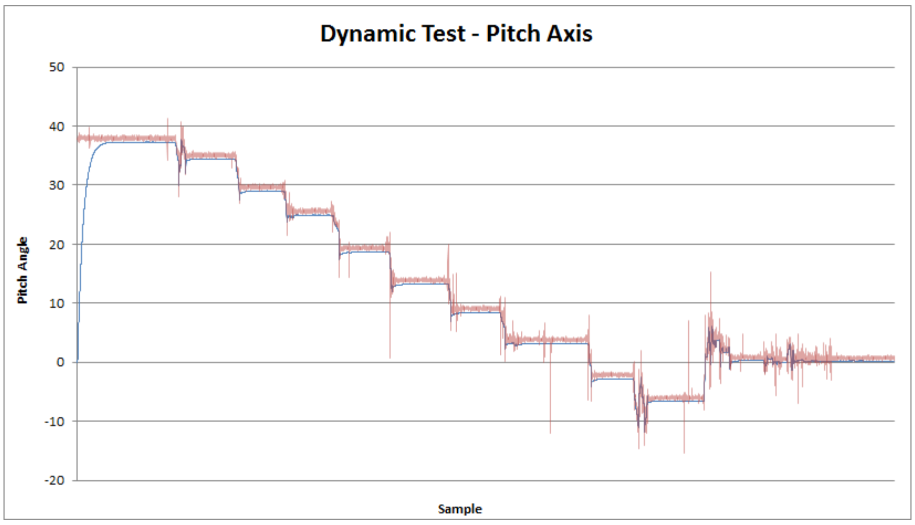


Figure 43: Pitch Angle Output for Dynamic Test with accelerometer based angle calculations overlayed.

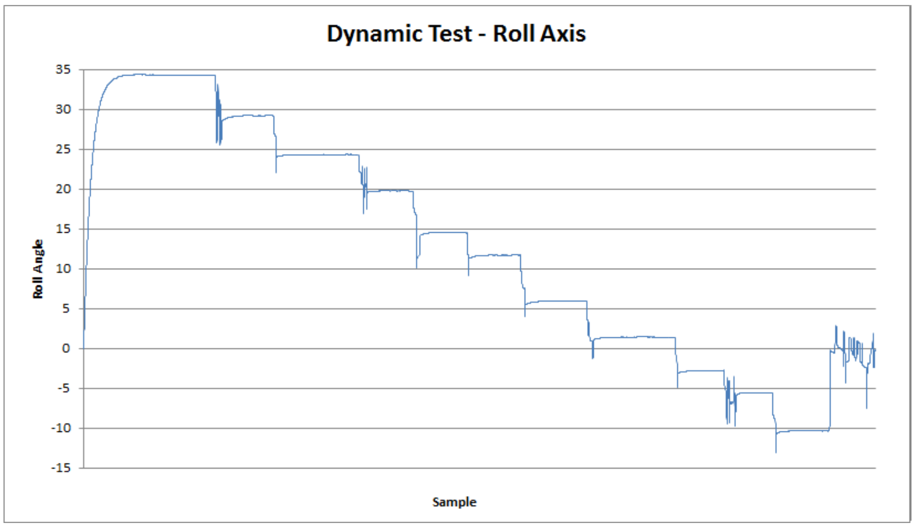


Figure 44: Roll Angle Output for Dynamic Test.

We compared the measured attitude data and the measured projected angles. We found some discrepancy between the two, the result is shown the two plots in figure 45 and 46.

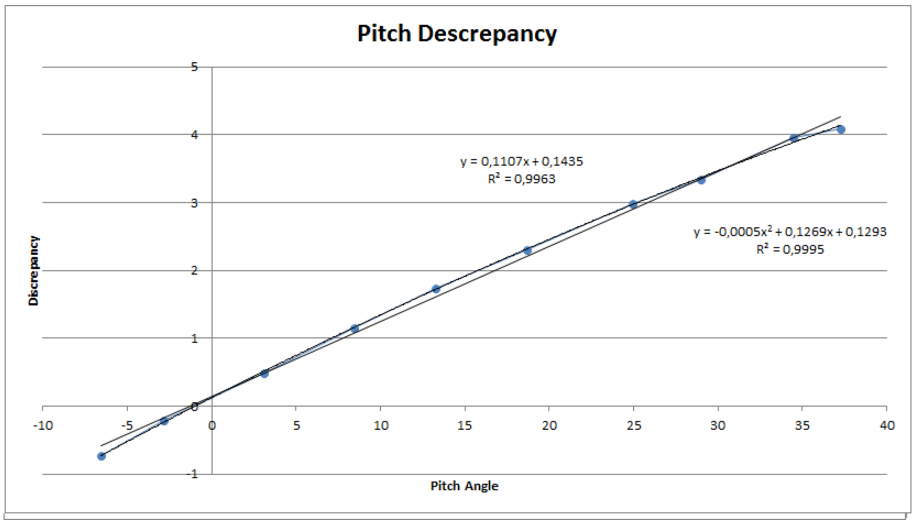


Figure 45: Pitch Angle Discrepancy for dynamic test.

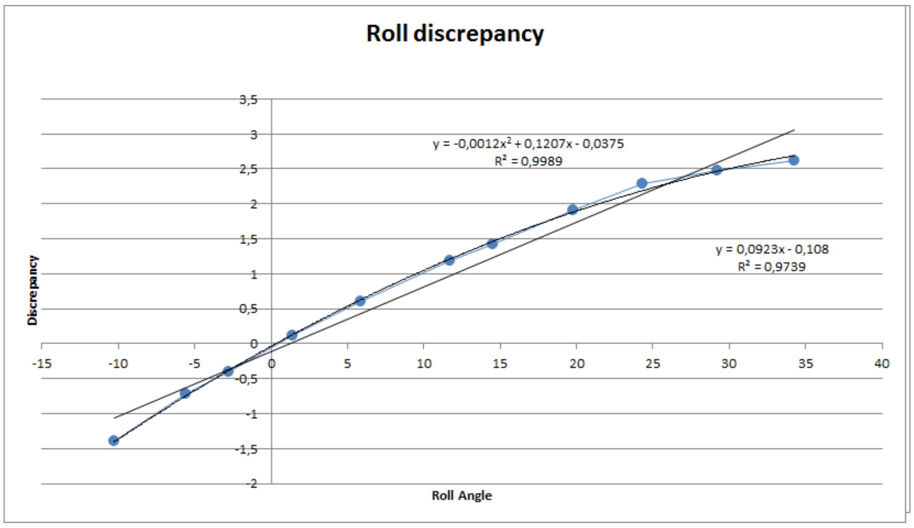


Figure 46: Roll Angle Discrepancy for dynamic test.

Table 19 summarises the linear and quadratic equations we calculated on the discrepancy. The regressions were calculated in a spread-sheet. We have decided to take out the zero-crossing parameter from the equations, because the angles should be zero when at level.

Axis	Linear regression	Quadratic regression
Roll	$y = 0.0923x$	$y = -0.0012x^2 + 0.1207x$
Pitch	$y = 0.1107x$	$y = -0.0005x^2 + 0.1269x$

Table 19: Angle output correction based on the dynamic test.

We observe from the figures that the quadratic equation is a better fit. We also see that the quadratic parameter is one magnitude smaller than the linear parameter. The linear equation would be a good approximation of the angle discrepancy.

Another effect that we saw was that the attitude estimate over-shoots during the sharp transitions. Figure 47 and 48 shows a transition for each roll and pitch.

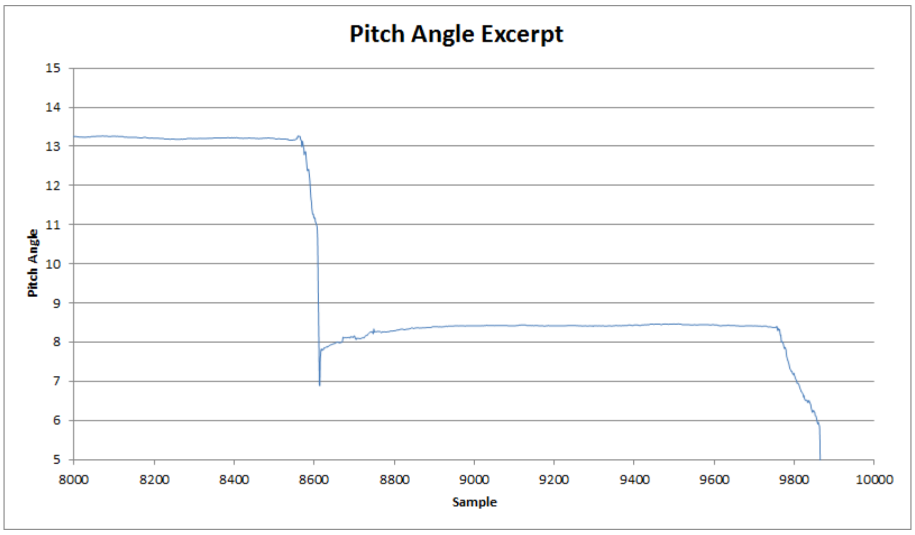


Figure 47: Pitch Angle overshoot during sharp transition.

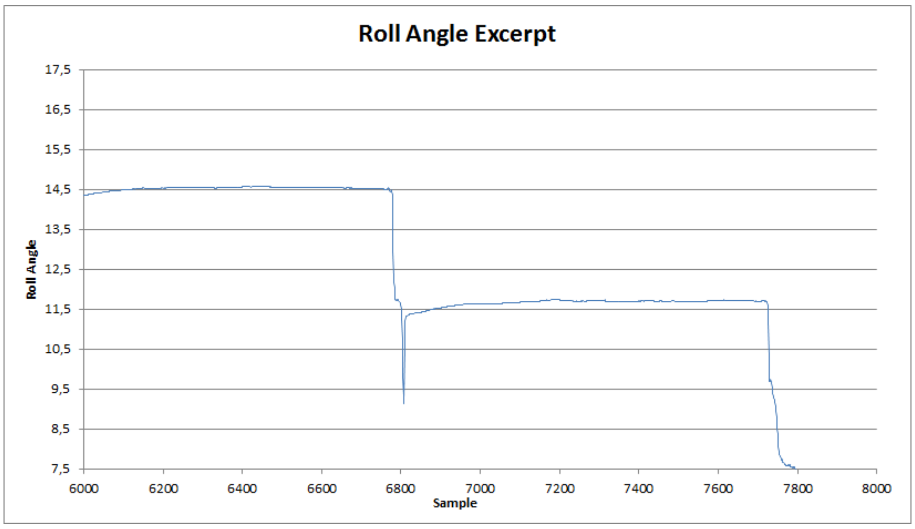


Figure 48: Roll Angle overshoot during sharp transition.

We see that there is a short spike during sharp transitions for both axis. This is most probably due to some *give* in the gearing of the test platform; when the platform is forced to a new angle, the platform returns a little when we let go. In other words, it is not perfectly rigid.

We decided to let the user input parameters for linear corrections for the accelerometer and the gyroscope as part of the user interface.

4.2.3 Shake test

One additional test we did was a shake test. In this test we gave the pan-tilt rig a violent shake. The purpose was to see how well the algorithm suppresses linear acceleration. We let the system settle first, and that shook the rig back and forth, keeping the rig flat against a surface. We did this first for the roll-axis, then for the pitch-axis, and one second time for the roll-axis. The accelerometer should pick up acceleration, but the gyroscope should pick up very little rotation. Next we plotted output from the algorithm and the angle for the corresponding axis calculated solely from the accelerometer data. Figures 49 and 50 show the plots for the roll axis.

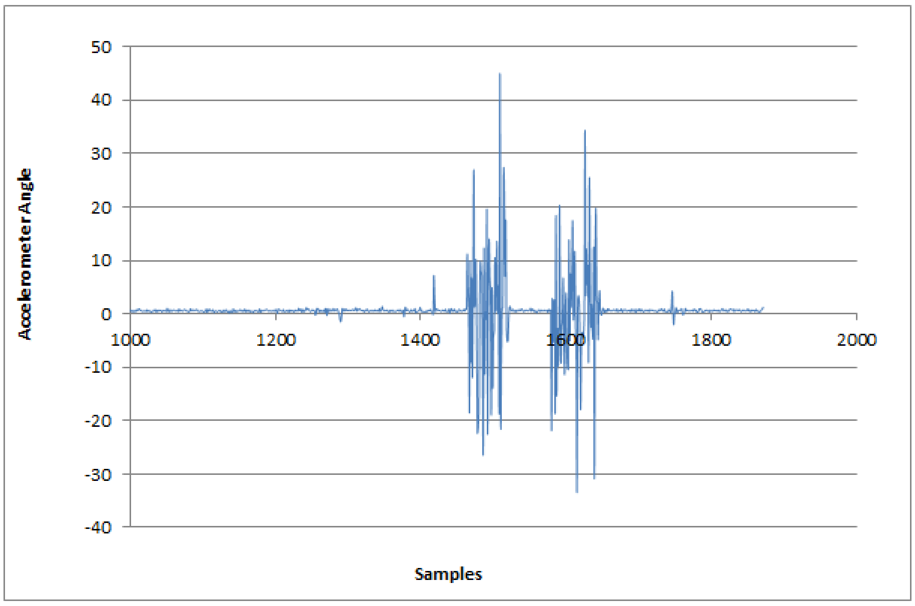


Figure 49: Angle calculated from accelerometer for shake test.

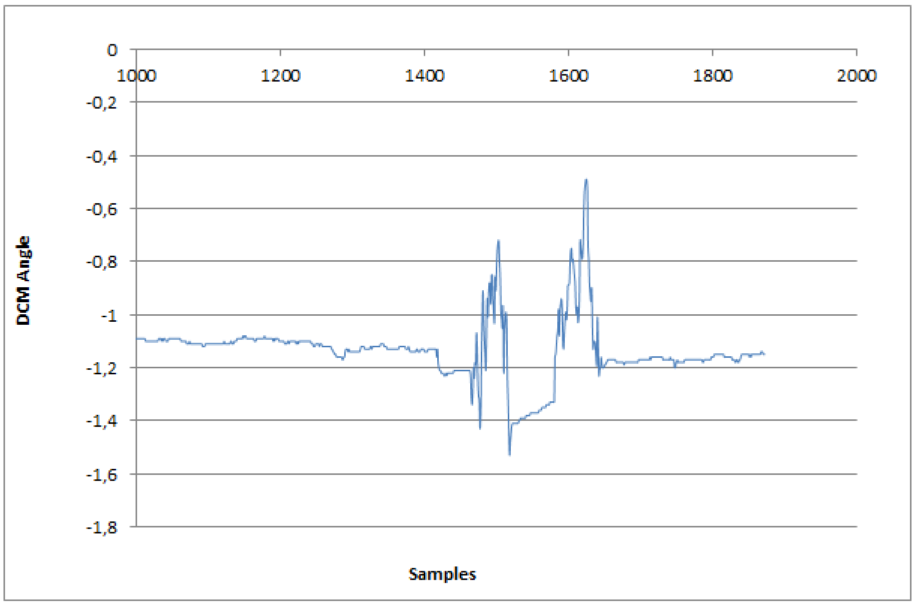


Figure 50: Algorithm attitude output for shake test.

We notice that the angle calculated from the accelerometer is quite large, almost 60 degrees difference peak-to-peak. When compared to the output from the algorithm, we see two peaks corresponding to the two shakes. There is a change in angle, but less than a degree. This could well be a little rattling in the pan-tilt rig, as the sensor was at this point still mounted on the pan-tilt rig. The attitude algorithm suppresses linear acceleration well.

4.2.4 Observations from the dynamic tests

We saw that general purpose hobby servos are not well suited for precise measurements. The main issues with the test method was that the servos were not very precise. When the servos were commanded to go to a position, we saw that the servos would often miss by several degrees. When a repeated command were sent the servos would not to go to the same position. Secondly, sometimes the servo would hunt for its position. When this happened, the servo would shake. This hunting would introduce noise into our attitude measurements.

It is possible that there are more precise servos that would be better suited for our test, but we did not trying to find one. A better method would perhaps to use a stepper motor, which have precisely defined steppes, and to gear that motor as low as possible to produce smooth movements.

We observed that there was a discrepancy between the attitude sensor angle and the measured angle. We also saw an overshot during sharp transitions.

The shake test showed that the attitude sensor suppressed linear disturbances very well.

4.3 Tachometer test

The aim of this test is to investigate the performance of the system when in motion.

4.3.1 Setup

For this test we attached our sensor to a rod coupled to a wheel. The wheel could spin. The spinning wheel will translate to a sinusoidal tilt angle as seen from the rod. A micro-controller was set up to count the length between pulses from an IR reflector/detector. This would give us a basic tachometer. See the figure 51 for the test setup.

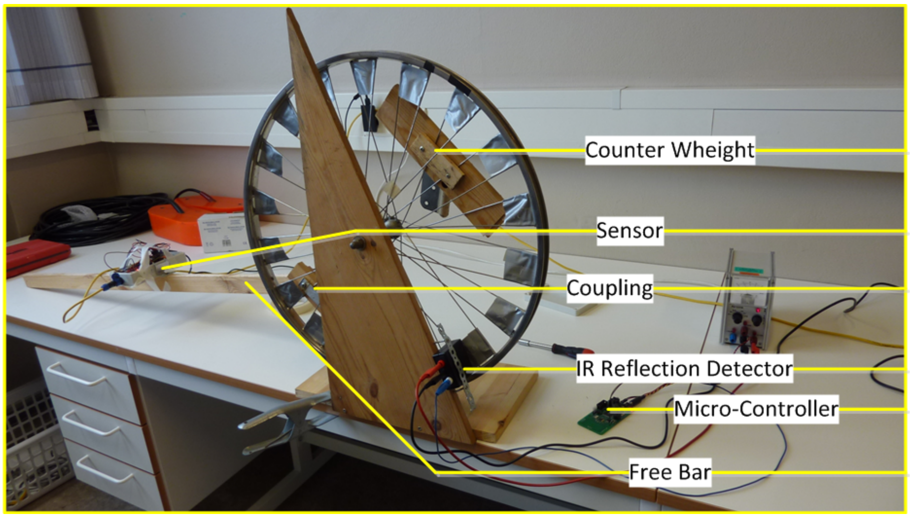


Figure 51: Overview of the tachometer test setup.

Every pair of spokes was taped with reflecting silver tape, 18 pairs altogether. The difference between top angle and bottom angle was 26 degrees. This means we have a 1.45 degree resolution per *tick*. The IR detector will give out a pulse when a reflection of the tape is detected. The IR detector is based on the Hamamatsu light modulation photo IC[80]. An Atmel XMEGA measures the time between two pulses, and outputs that time on the USART port[81].

Measurements where taken of the geometry the setup. A laser pointer was used to measure the top and bottom angles against a flat wall. A tape measurer was used to measure distances. We assume that the wall we measure against is right angled. The angles are listed in table 20.

[80]Data-sheet is included in the appendix
[81]Source code is included in the appendix

	Adjacent	Opposite	Calculated	Sensor
Top	251.3 cm	513.3 cm	26.09°	27.64°
Bottom	41.3 cm	524.7 cm	4.50°	3.75°

Table 20: Measurements of top and bottom angles in tachometer test.

A total of ten tests where performed. Table 21 list the tests performed.

No.	Summary
1	One slow controlled revolution.
2	Gentle spin.
3	Three successive gentle spins.
4	Moderate spin.
5	Hard spin. Failed when equipment detached.
6	Hard spin. Failed when rod lifted off the table.
7	Hard spin with extra weight. Rod lifted off table again.
8	Hard spin with extra weight.
9	Three successive hard spins and stops. Frame moved, failed.
10	Three successive hard spins and stops.

Table 21: Tachometer tests performed.

Of the ten tests, tests 5, 6, 7, 8 and 9 are not used due to equipment malfunction.

4.3.2 Data

We manually fitted the sensor and the tachometer data together, the result of the plots are in figures 52, 53, 54, 55, 56 and 57. We adjusted the start time offset between the tachometer and the attitude sensor. The tachometer was always started from the same sector of the wheel, we adjusted the angle offset to match the two data sets. Finally we adjusted the half range angle of the tachometer data set to fit the attitude sensor data.

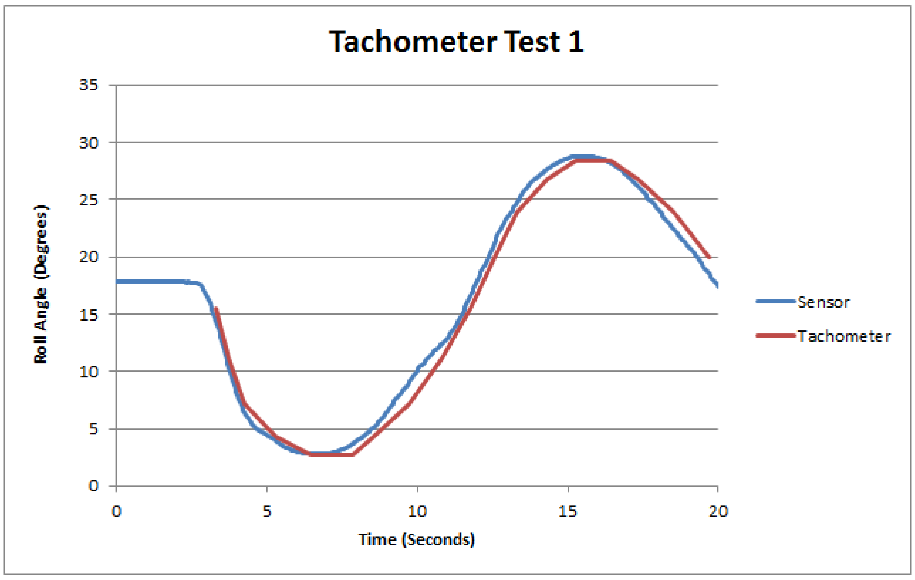


Figure 52: Tachometer test 1. plotted tilt angle from sensor and tachometer.

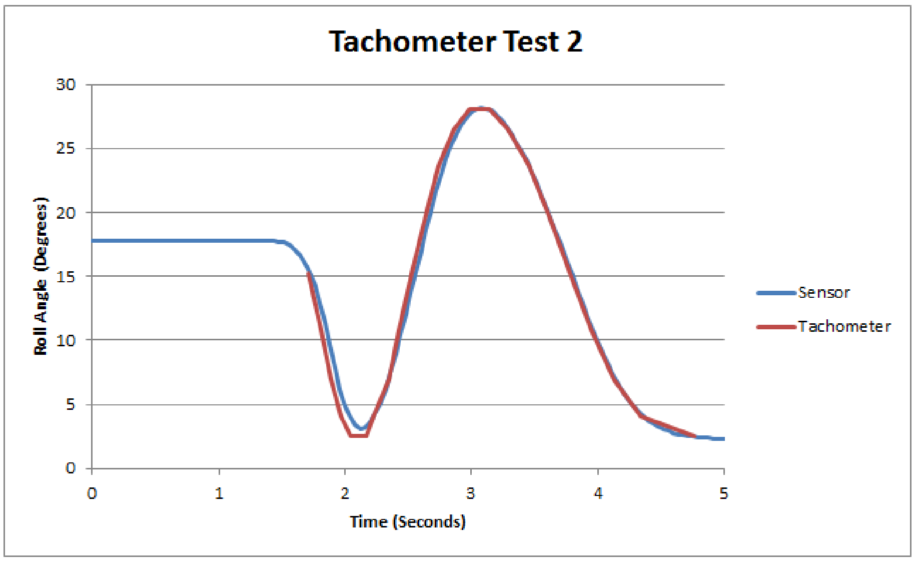


Figure 53: Tachometer test 2. plotted tilt angle from sensor and tachometer.

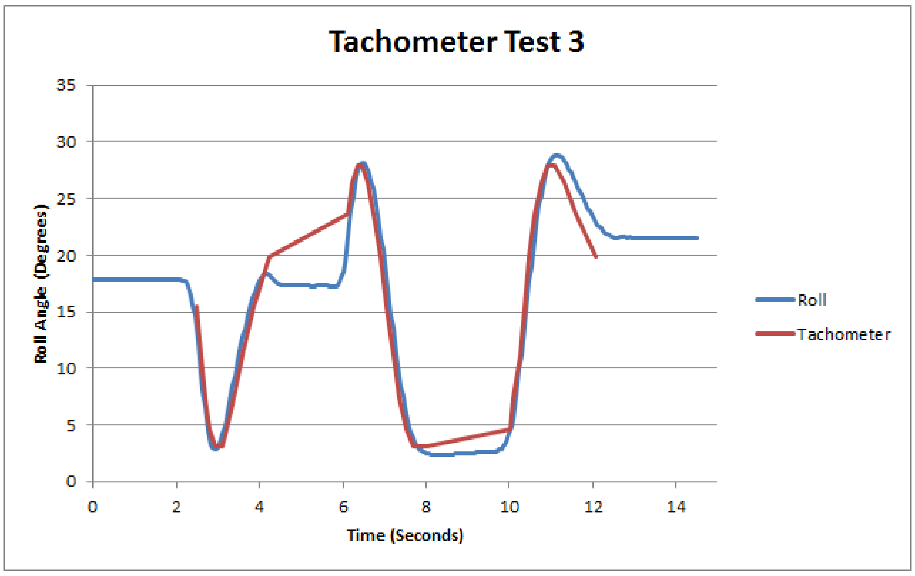


Figure 54: Tachometer test 3. plotted tilt angle from sensor and tachometer.

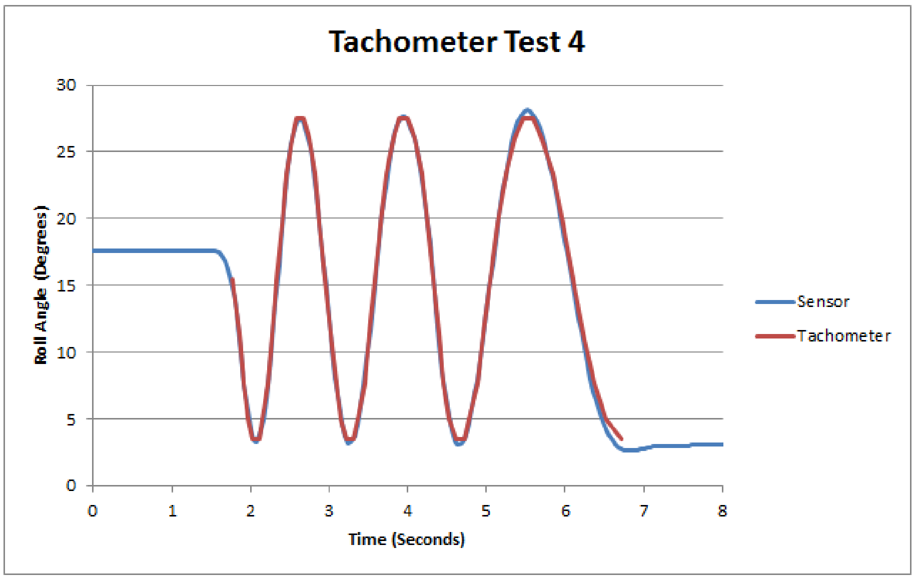


Figure 55: Tachometer test 4. plotted tilt angle from sensor and tachometer.

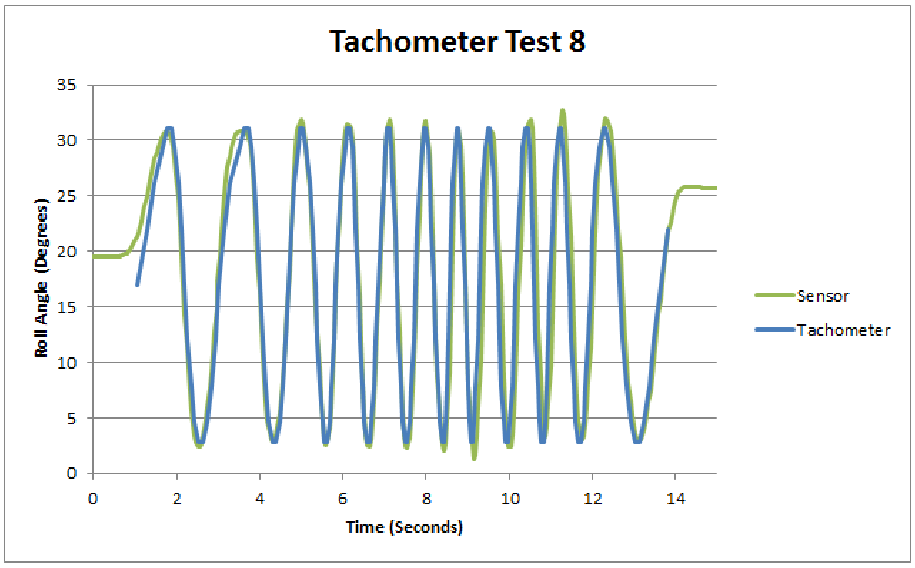


Figure 56: Tachometer test 8.

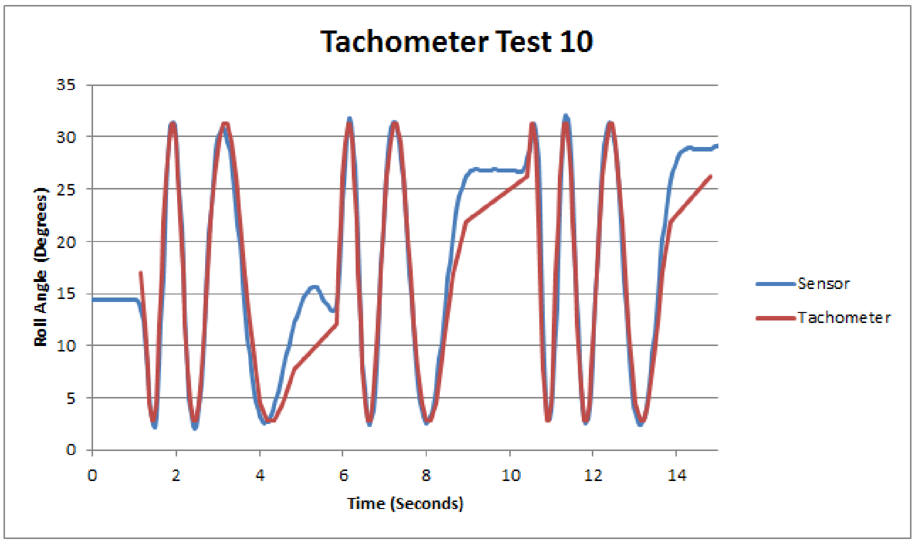


Figure 57: Tachometer test 10. plotted tilt angle from sensor and tachometer.

We then took the took the difference between the tachometer angle data and the sensor angle data to get the difference between the two. We then took the standard deviation of the difference. The results for the test are summarised in table 22:

No.	Standard Dev.	Average	Time offset	Angle offset	Angle Half Range
1	0.6886°	−0.005°	-0.395 s	15.53°	13°
2	0.5692°	0.008°	-3.525 s	15.35°	13°
3	1.583°	0.008°	-2.39 s	15.675°	12.575°
4	0.954°	−0.003°	-0.35 s	15.25°	12.16°
8	2.668°	0.003°	-1.6 s	16.9°	14.35°
10	1.883°	−0.002°	-3.95 s	16.82°	14.45°

Table 22: Summarised tachometer test results.

4.3.3 Observations from tachometer test

In the tachometer test we saw that there was a good match between the attitude sensor and the tachometer rig. Visually, we can see that the two curves fit together. When we looked at the difference between the attitude sensor and the tachometer we get figures for standard deviation as low as 0.5692°.

We observe that there is a change in the manual adjustments we had to make for test 8 and 10. In test 8 and 10 we had fitted the rod with a weight to hold down the rod while we were spinning the wheel. This altered the initial setup, and this is probably the reason for the changes.

The attitude sensor performed well when measured against the tachometer rig, but we saw lower accuracy when the speed of the wheel increased.

4.4 Field test at Songsvann

Sognsvann is a small lake close to the University of Oslo, Blindern campus. The lake is representable for the type of environment our system is designed for. The purpose of the field test was twofold: To test the system in a typical environment, and to test its performance against the sonar equipment.

4.4.1 Setup

The test was carried out on the 20th of February. Prior to departing for Sognsvann, all the equipment was checked and packed at Blindern. Along with the attitude system we brought a sonar system, and equipment needed at the site (such as ice bores, batteries, an axe, a table, and so on). The equipment was then packed into a car, and we headed for Sognsvann.

At Sognsvann the weather was very good, with clear blue skies and no wind, as can be seen in figure 58 from the test site. We carried the equipment the short distance from the car park down to the lake. We then scouted out a promising spot to carry out the test. The lake was frozen over, and the ice was deemed safe. We wanted to find a spot on the ice were the water was deep enough for our test, and to have an unobstructed path between the sonar and the target.

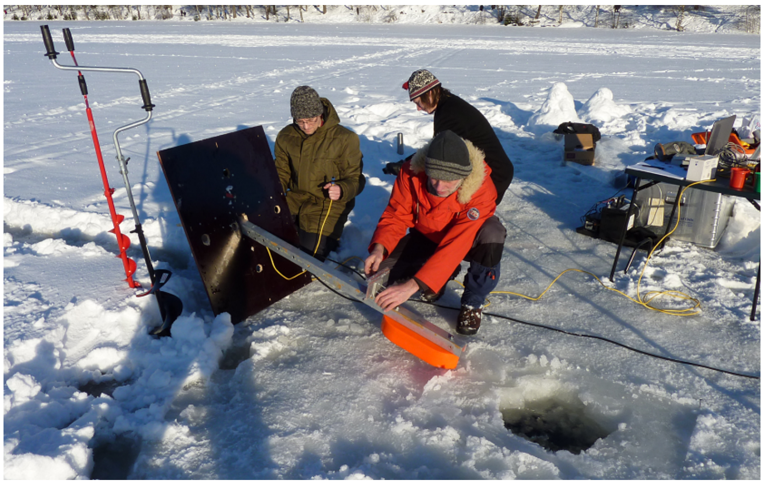


Figure 58: Preparing the instruments at Sognsvann field test.

We removed the snow from the top of the ice. There was about 10 cm of water on top of the ice. The water made it a little difficult to drill holes for our equipment.

We drilled an exploratory hole. We measured the depth to be 5 meters, which would be sufficient for our test. The hole was then enlarged to be big enough to pass the sonar boom through. A second hole was drilled roughly ten meters away from the first hole. This hole was measured to be 5 meters deep. The second hole would be used to lower the acoustic target into. The target was a small copper sphere, and its diameter is such that resonates with the sonar frequency. This creates a strong return echo. The target sphere is 1 cm in diameter, and can be seen in figure 59.

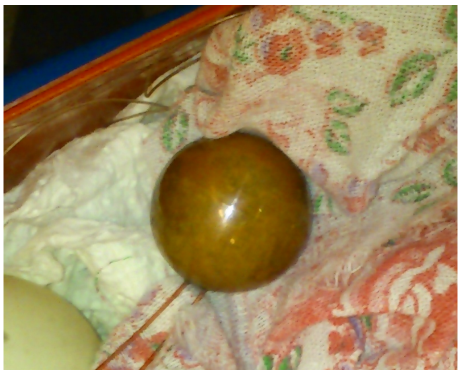


Figure 59: Copper acoustic target used in the field test.

After we had prepared the two holes, we lowered the sonar and the acoustic target. We then tested that the sonar system could locate the target. The sonar could see the target, so we retracted the sonar again. We then mounted the sensor system onto boom and lowered the boom into the water. The sensor was mounted so that the roll axis of the sensor was aligned to the sonars beam direction.

4.4.2 Description of tests

A total of nine tests were carried out. Data from the sonar and the sensor system was recorded simultaneously. The sonar was attached to a boom, and the motion sensing system was placed close to the sonar. The boom assembly was attached to a wooden cradle assembly that allows the boom to swing in one direction. A centre position and a grade scale marks the top of the boom cradle. The description of the tests are listed in table 23, in chronological order.

No.	File	Description
1.	caption1.txt	Equipment test
2.	caption2.txt	No Movement - Stationary
3.	caption3.txt	Sonar moved in steps
4.	caption4.txt	Slow swinging
5.	caption5.txt	Shaking of mounting plate and boom
6.	caption6.txt	No movement - Stationary
7.	caption7.txt	Sonar moved in steps
8.	caption8.txt	Slow, then fast swinging
9.	caption9.txt	Shaking of the mounting plate and boom
10.	caption10.txt	Shaking of the mounting plate and boom

Table 23: Description of tests performed at Sognsvann

Tests 1 through 5 were performed with the target at roughly 11.5 meters distance from the sonar. The sonar data was a little noisy, so we decided to drill a third hole closer to the sonar and repeat the tests. Tests 6 through 9 were a repeat of the tests, but approximately 1 meters closer to the target. Test number 9 was repeated in test number 10 as the acoustic target went too far outside the sonar beam.

Test number 1 was an equipment test, just to see that everything worked before we proceeded.

Test number 2 and 6 were stationary tests. With the acoustic target centred in the sonar beam, a short length of attitude data was collected. This serves as a baseline, so that we can extract angle offsets between the two systems and calculate a baseline performance.

In tests 3 and 7 we moved the transducer in steps. In this test it should be relatively easy to pick out the angles in the two systems in this test, to do a stepwise comparison.

In tests 4 and 8 the boom was swung back and forth, taking care in keeping the acoustic target inside the view of the sonar system. These tests were to emulate the rocking of a boat boat.

Tests 5, 9, and 10, were designed to try to throw of the motion sensing system by violently and rapidly shaking the boom cradle and the boom itself.

4.4.3 Data

The data from the sonar and the sensor was processed in Sonar5. Our acoustic target was stationary, so when the boom was tilted up and down, we could see the target moving up and down inside the beam. After merging the attitude data with the sonar data, we would want to see the acoustic target stationary, as if the hydrophone was not moving at all.

The first set of data we present is the stationary test from test 2. Here we looked at the sonar target when there is no movement in the hydrophone. We want to see a stationary flat line plus noise. Figure 60 shows the vertical position calculated from the sonar data using Sonar5.

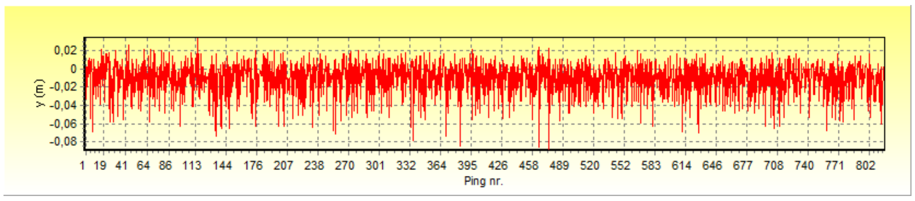


Figure 60: Test 2. Vertical position before applying attitude corrections.

The pitch and the roll should not vary much either, since we did not move the rig with the sonar. Figures 61 and 62 show the pitch and the roll data recorded by the sensor system.

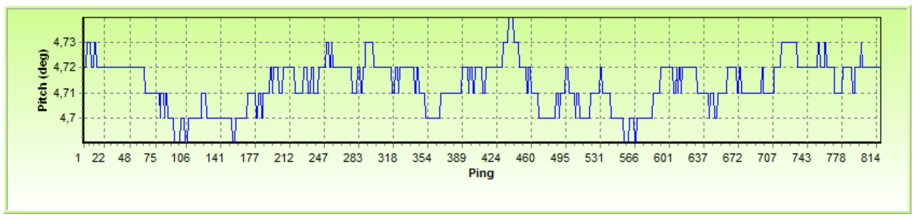


Figure 61: Test 2. Pitch angle measured by the sensor

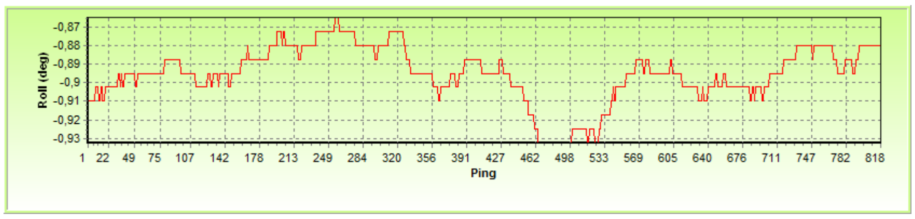


Figure 62: Test 2. Roll angle measured by the sensor

We can see that there is some very slight changes in the pitch and roll angles. The attitude[82] data was then merged with the sonar data in Sonar5.

[82]Pitch and roll

The compensated vertical position can be seen in figure 63.

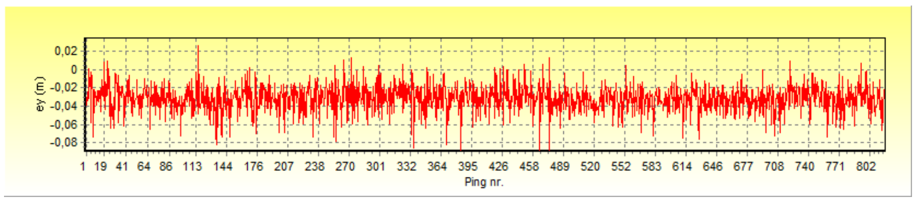


Figure 63: Test 2. Vertical position after applying attitude corrections.

Visually there is not much to distinguish the two curves. We calculated the standard deviation from test 2 for the vertical position and the corrected vertical position, summarised in the table 24.

Average	Standard Deviation	Data
-1.110 cm	1.934 cm	Sonar data only
-3.150 cm	1.420 cm	Sonar merged with attitude data

Table 24: Statistics on vertical positions from test 2.

The standard deviation is a little lower for the corrected vertical position. If the recorded attitude was just noise, that noise should be additive. The standard deviation of the merged data would be higher that the sonar data alone.

We have now established a baseline for when the acoustic target is in the centre of the beam and the hydrophone is not moving. This is the optimal situation with respect to performance of the sonar and the motion sensing system. Next we present the data for the other three tests.

The next plots is data from test 7. This is one of the test where we move the sonar boom. Figure 64 shows the vertical position of the target, as seen by the sonar system. Figure 65 shows the tilt angle simultaneously measured by the sensor.

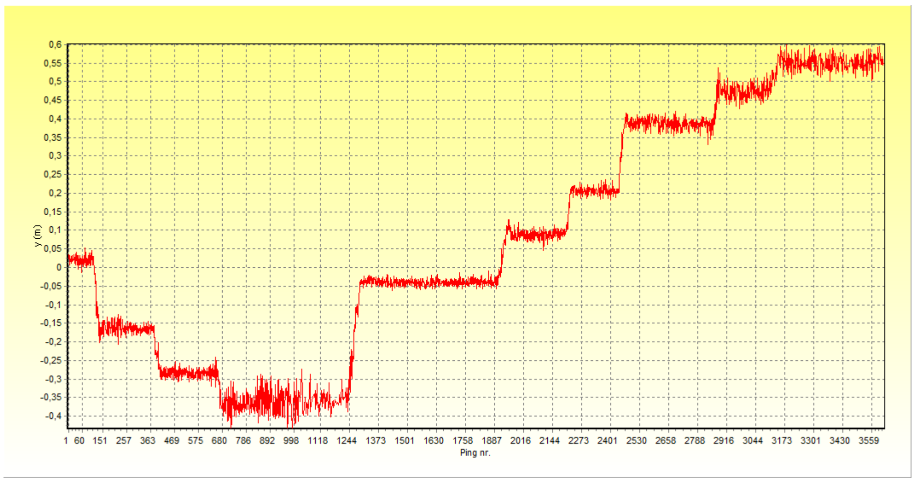


Figure 64: Test 7. Vertical Position before applying attitude corrections.

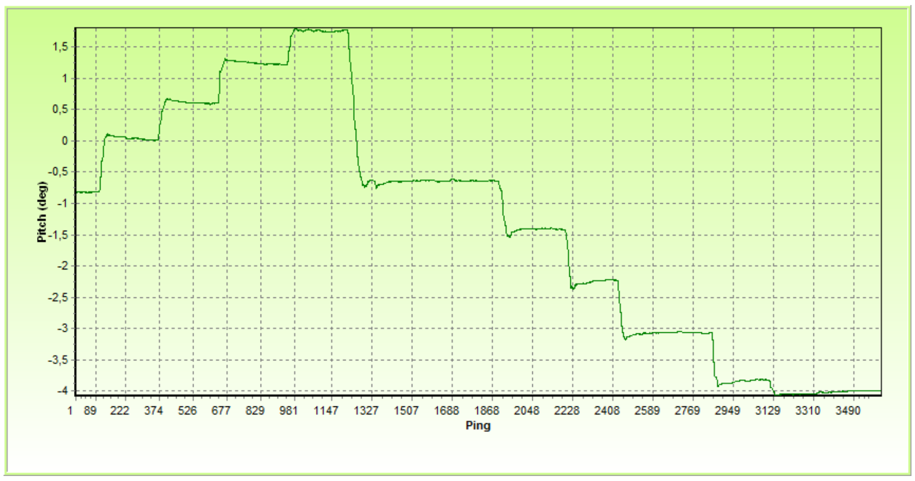


Figure 65: Test 7. Pitch angle measured by the sensor.

As can be seen from figures 64 and 65, the data from the sonar and the sensor are similar but opposite. The oppositeness depends on how you define your axis. Figure 66 shows the vertical position of the target after we merge the attitude with the sonar data.

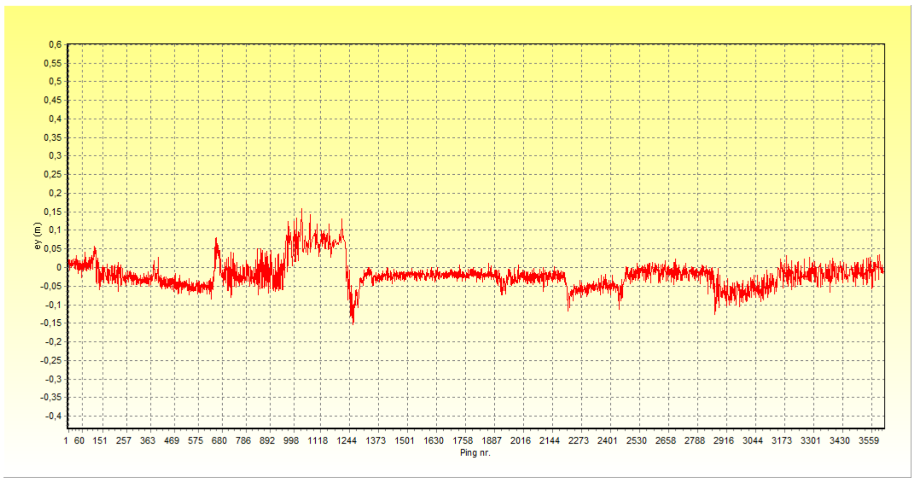


Figure 66: Test 7. Vertical position after applying attitude corrections.

As we can see in the figure 66, the jumps are now gone. There are some parts of the plot that are straighter than other. If we look at the return strength[83] of the echo in figure 67, we can see that return is strong when the target is in the centre of the beam, and weakens when the target is nearer the edge of the beam. With a lower return strength it becomes more difficult for the tracking algorithm to estimate vertical position. The determination of the position becomes drastically unreliable outside the half-power[84] of the sonar beam[85].

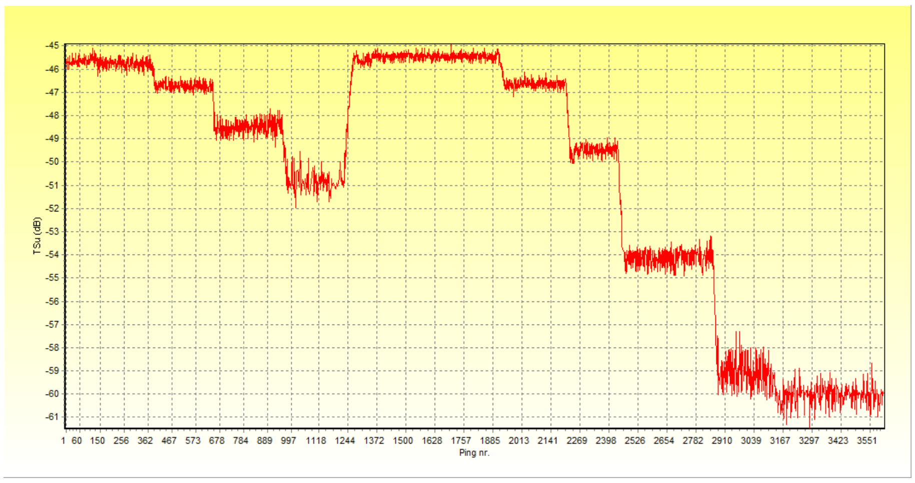


Figure 67: Test 7. Uncompensated target strength from the sonar.

We observe that low return strength causes more noise in the vertical posi-

[83]Uncompensated target strength
[84]-3dB
[85]Unpublished data by Helge Balk

tion.

We also see that between ping 680 and 1244, the motion sensor picks up two distinct positions, but the sonar only picks up one.

Table 25 summarises the standard deviation in the vertical position from test 7. If we where to exclude noisy sections, we would get a lower deviation.

Average	Standard Deviation	Data
4.71 cm	29.66 cm	Sonar data only
-2.917 cm	3.188 cm	Sonar merged with attitude data
-2.349 cm	1.766 cm	Sonar merged with attitude data (ping 1372-2398)

Table 25: Statistics on vertical positions from test 2.

If we take only an extract of the data, from when the target was near centre of the beam and the return strength was strong, we get an improved figure of standard deviation. We calculate the standard deviation from the corrected vertical position in ping 1372 through 2398. This deviation is closer to the deviation we calculated from test 2.

In test 8 we swung the boom back and forth in a smooth motion, first slowly and the quickly. We tried to keep the acoustic target inside the beam at all times. Test eight test was designed to see how well the motion sensing system copes with a constantly moving target. Figures 68 and 69 shows the sonar alone and merged vertical position of the acoustic target.

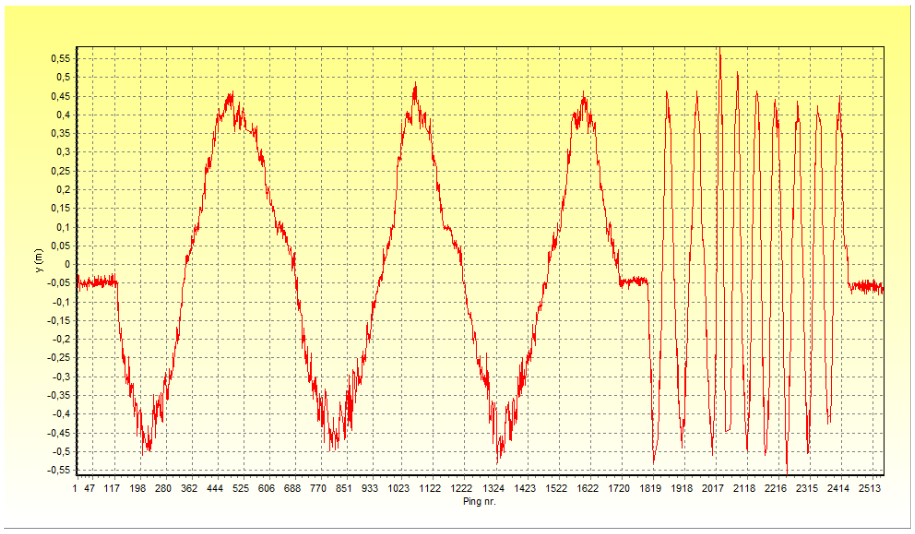


Figure 68: Test 8. Vertical position before applying attitude corrections.

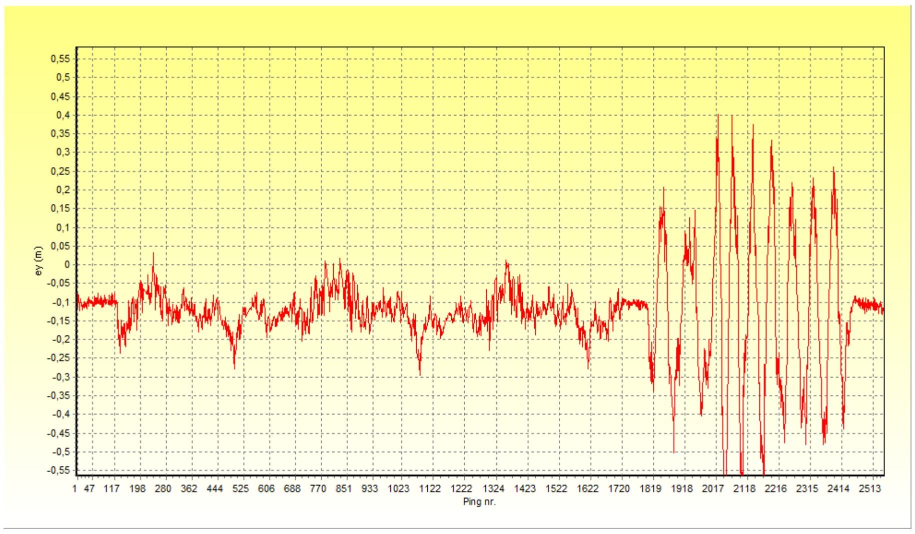


Figure 69: Test 8. Vertical Position after applying attitude corrections.

We see that the we are able to smooth out the data when the hydrophone moves at a period of 17.68 seconds, but not when the period increases to 1.229 seconds[86]. We summarise the data in table 26. The data up until ping 1800 is of a slowly swinging boom.

Average	Standard Deviation	Data
-1.982 cm	26.42 cm	Sonar data only
-12.62 cm	12.29 cm	Sonar merged with attitude data
-12.61 cm	4.070 cm	Sonar merged with attitude data (ping 1-1800)

Table 26: Statistics on vertical positions from test 8.

The period of the swinging motion is listed in table 27, showing the shortest and the longest swing period.

	Period	Frequency	Motion Samples	Pings
Slow	17.68 Sec	0.0566 Hz	884	586
Fast	1.229 Sec	0.5882 Hz	90	67

Table 27: Test 8. Period of swing of the hydrophone.

In test 9 and 10 we tried to stress the sensor system as much as possible. We picked up the boom rig by one corner and shook the whole platform. We then shook the rig by the opposite corned. Finally, we shook the sonar boom violently. Figure 70 show the vertical position calculated using the sonar data. Figure 71 shows the vertical position when merged with the attitude data.

[86]See table 27 for periods

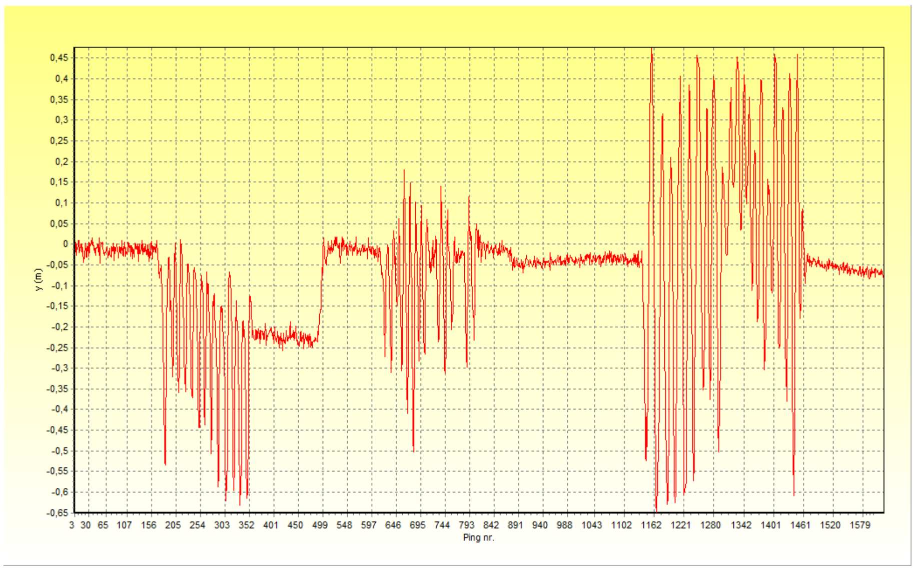


Figure 70: Test 10. Vertical position before applying attitude corrections.

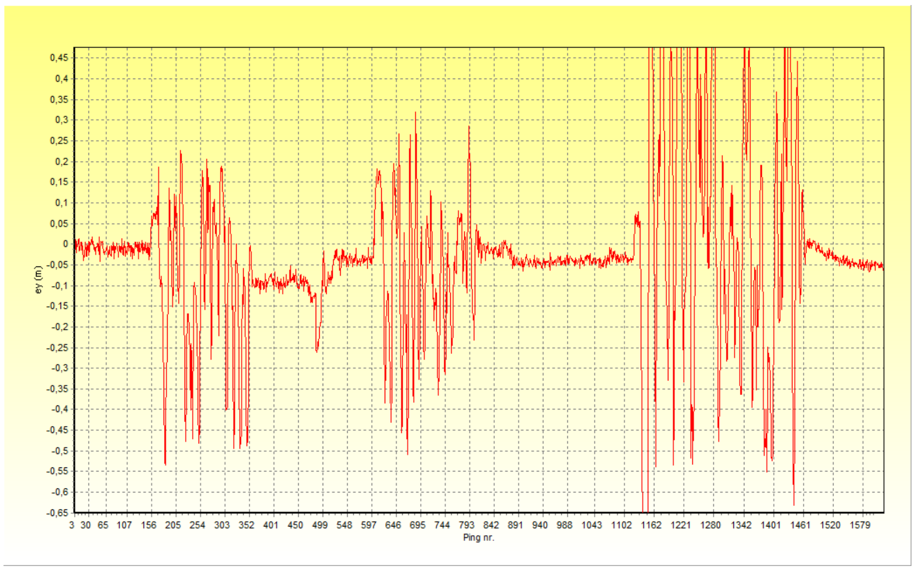


Figure 71: Test 10. Vertical position after applying attitude corrections.

As we saw in test 8, the motion sensing system is not able to correct for motion that is too rapid. Visually we can see that the motion sensing system has managed to smooth out some sections, such as when the rig was let go off.

In table 28 we summarise the results from the tests in degrees, rather than centimetres vertical displacement. We do this by using the tangents function. Tests 7. and 8. appear twice, with vertical standard deviation for both the whole data set, and an extracted data set.

No.	Acoustic range	Vertical SD (cm)	Vertical SD (degrees)
2.	11.74 m	1.420 cm	0.0692°
7.	10.00 m	3.188 cm	0.1827°
7. (1372-2398)	10.00 m	1.766 cm	0.1012°
8.	9.993 m	12.29 cm	0.7046°
8. (1-1800)	9.993 m	4.070 cm	0.2334°
10.	9.992 m	-	-

Table 28: Summary of test results from Sognsvann field test in degrees.

4.4.4 Observations from Field Test

The test condition at Sognsvann were very good. We had perfect weather, and the ice was safe. There was no injury to the participants, and no damage to our equipment. The mood in was good, and everyone was motivated. Our equipment worked, and we were able to collect ten sets of data.

We saw that the attitude system worked well in the environment at Sognsvann, which is an important success criteria. The sensor system was submerged in water, and there was no water ingress.

We saw that we could compensate sonar data using the attitude from the sensor system. We had some challenges when trying to synchronise the two sets of data. Since both data sets were time stamped, after finding the displacement between the two, the sensor data could be merged with the sonar data. Sonar 5 calculates the tilt angle from the sensor by looking at the two closest sensor readings, and linearly interpolating between the two[87].

[87] $tilt = at + b$.

5 Summary

The goal of this project was to build a system that could be used to compensate for a mobile hydrophone in hydro-acoustic surveys.

The first unit we built had limitations with respect to a micro-controller that was not powerful enough to run the attitude estimation algorithm. We therefore built a more powerful second unit that could run the algorithm.

The system had to operate in a typical environment for a hydro-acoustic survey. We successfully ran a hydro-acoustic survey in such an environment when we did the field test at Sognsvann. The system was submerged in water for several hours without any water ingress. The system should also be able to tolerate rough handling. The system has no parts that are particularly sensitive, so we have achieved this too.

The system was to be reasonably priced. The total cost of the parts in the second generation system was 140.12 USD. The single most expensive component was the MPU9150 evaluation board, costing 79.95 USD. The price of a single MPU9150 sensor is only 17 USD. A custom PCB board for the MPU9150 sensor would bring the total cost down considerably. Optionally a custom board can include all the other components, except for the Raspberry Pi single-board computer. This would bringing the cost even further down. In any case, the total cost of one unit is still low.

The system should be easy to use. We achieved this by writing a control program for the system. The attitude system will turn itself on automatically when power is applied. The user only needs to run the control program and click the *start* button to see the data coming through.

We ran several tests to investigate the performance of the second generation system, both in the field and in a laboratory set up.

In the static tests we investigated the raw output from the MPU9150 accelerometer and gyroscope when motionless.

Both the MPU9150 accelerometer and gyroscope sensor were sensitive to changes in temperature. The accelerometer showed a linear response to temperature changes, while the gyroscope showed a more complex response. The non-zero output from a motionless gyroscope is its bias. Small changes in the bias of the gyroscope is handled by the complimentary filter. The initial bias of the gyroscope can be removed through the control program by using the *set bias* function.

We looked at different P and I factors for the PI-regulator in the complimentary filter. We found a value for P and I that we think will work well.

We also looked at a typical settling time for the complimentary filter from start-up to a operational state. The settling time depends on the P and I factors.

Finally we looked at the attitude estimate in a static set-up. We found the standard deviation of the pitch and roll axis to be 0.016° and 0.032° respectively.

In the dynamic tests we wanted to use servos to generate motion test patterns, but we found the servos to be too inaccurate. We tested how well the complimentary filter could measure attitude against an external reference tilting the sensor at different angles. We found a discrepancy between the attitude estimate and the external reference.

We subjected the system to vigorous linear forces, and we found that the complimentary filter can suppress this linear acceleration well.

In the tachometer test we used a spinning wheel to apply a known motion onto the attitude estimation system.

We calculated standard deviations of the difference between the two measurements to obtain performance of the attitude estimation system. The standard deviation of this dynamic test was 0.5692° and 2.668° as best and worst value.

In the field test at Sognsvann we used a Simrad EY500 sonar together with our attitude system. The attitude system worked well in this environment.

The sonar and the attitude data was processed in Sonar5. With a motionless target, we used the attitude data to *smooth* out the motion we subjected the hydrophone to.

5.1 Improvements and further work

We believe that the systems accuracy can be improved. There is some jitter in the real-time clock we use to time-stamp the output data, but we can not improve on this. There is also some jitter in the attitude program. The attitude program asks the MPU9150 if it has new data ready before reading the data, and this time may vary. Implementing external interrupt may lower jitter.

In the complimentary filter algorithm we assume that the small angle approximation holds, but we may violate this approximation with large angular displacements. Higher internal sampling rate would help against this.

Then there is the noise and bias in the accelerometer and the gyroscope. The complimentary filter will recover from gyroscope error and bias, and accelerometer errors. The initial gyroscope bias can be zeroed from the control program. The accelerometer noise will be filtered away by the complimentary filter because the it behaves as a low pass filter. The attitude estimate of the complimentary filter will tend towards the accelerometer. We know that the bias of the accelerometer is affected by temperature, so compensating for temperature will improve the attitude estimate.

There was not enough time to do everything we wanted to do. We will here list suggestions for improvements and for further work, in no particular priority.

- An external synchronization pulse output that marks the time of a sample. This could be used with sonar systems that support external synchronisation.
- External interrupt from MPU9150 for when data is ready could reduce jitter in the attitude data. The MPU9150 supports this, but we did not have this feature available in the Raspberry Pi at the time of programming.
- Real-time clock synchronization from the control program would be useful for the operator. At present the control software can send the time, but the attitude system does not adjust its real-time clock.
- Elevating the attitude systems thread priority in the scheduler. This could also reduce jitter, particularly if external interrupt in implemented.

There are some tests that we would have liked to do, but did not find the time to do. We list them in no particular order of priority.

- We would have liked to see if increasing the internal sampling rate would increase the accuracy of the system under rapid motion.
- We would have liked to do a test on a boat or a vessel. We did try to find the time for a test aboard the research vessel RSV Braarud. With such a test we could have investigated how the attitude estimation system behaved on a vessel. We would also have had the opportunity to test our system against the system on that vessel.
- We would also have liked to test our system against a state-of-the-art attitude system, such as the Kongsberg Seatex MRU-6 or similar. These systems are costly, and out of the budget of this project.

6 Conclusion

We have built a system consisting of a Raspberry Pi single-board computer and a MPU9150 MEMS accelerometer and gyroscope sensor. We have written a program that implements a complimentary filter for attitude estimation. This system can compensate for the attitude of a hydrophone in hydro-accoustic surveys.

- We have shown that the MPU9150 MEMS sensor can be used to stabilise acoustic data.
- We have put together a low cost system.
- We have shown that a complimentary filter can estimate attitude using an accelerometer and a gyroscope.
- We have shown that it is possible to implement a complimentary filter on the Raspberry Pi single-board computer.
- We have shown that this attitude estimate can be used to compensate for a mobile hydrophone in hydro-acoustic surveys.

The system that we have built will be used by the hydro-acoustic group at the University of Oslo, and is expected to be an asset in hydro-acoustic surveys.

7 Bibliography

1. David Titterton and John Weston. Strapdown Inertial Navigation Technology, 2nd Edition. IET Radar, sonar, navigation and avionics series 17, 2004. ISBN 0-86341-260-2

2. Marcelo Alonso, Edward J. Finn. Physics. ISBN 0-201-56518-8

3. Robert Mahoney, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear Complimentary Filters on the Special Orthogonal Group. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 53, NO.5 JUNE 2008.

4. Robert Mahoney, Sung-Han Cha, Tarek Hamel. A coupled estimation and control analyses for attitude stabilisation of mini aerial vehicles. November 20, 2006.

5. Grant Baldwin, Robert Mahoney, Jochen Trumpf, Tarek Hamel, Thibault Cheviron. Complimentary filter design on the Special Euclidian group SE(3).

6. Mark Euston, Paul Coote, Robert Mahoney, Jonghyuk Kim and Tarek Hamel. A Complimentary Filter for Attitude Estimation of a Fixed-Wing UAV.

7. Alan Burns and Andy Wellings. Real-time systems and Programming Languages. Fourth Edition. ISBN 978-0-321-41745-9.

8. David C. Lay. Linear Algebra and its applications. Third Edition. ISBN 0-201-70970-8.

9. Helge Balk. Linear Kretselektronikk. Compendium.

10. William Premerlani and Paul BizardDirection. Cosine Matrix IMU Theory. http://diydrones.ning.com/profiles/blogs/dcm-imu-theory-first-draft

11. Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. April 30. 2010.

A Contents of DVD-ROM

Papers

- Robert Mahoney, Tarek Hamel, and Jean-Michel Pflimlin. Nonlinear Complimentary Filters on the Special Orthogonal Group. IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. 53, NO.5 JUNE 2008.

- Robert Mahoney, Sung-Han Cha, Tarek Hamel. A coupled estimation and control analyses for attitude stabilisation of mini aerial vehicles. November 20, 2006.

- Grant Baldwin, Robert Mahoney, Jochen Trumpf, Tarek Hamel, Thibault Cheviron. Complimentary filter design on the Special Euclidian group SE(3).

- Mark Euston, Paul Coote, Robert Mahoney, Jonghyuk Kim and Tarek Hamel. A Complimentary Filter for Attitude Estimation of a Fixed-Wing UAV.

- William Premerlani and Paul BizardDirection. Cosine Matrix IMU Theory. http://diydrones.ning.com/profiles/blogs/dcm-imu-theory-first-draft

Data-sheets

- AK8975 Magnetometer

- MPU9150 Multi-senor Evaluation Board

- MPU9150 Multi-sensor Register MAP

- MPU9150 Multi-sensor Product Specification

- ATMega8 Micro-controller

- LIS331DLH Accelerometer

- MMA8451Q Accelerometer

- IMU3000 Gyroscope Product Specification

- BMP085 Digital Pressure Sensor

- MAX3232 RS232 Transceiver

- MRU5+ Motion sensor

Application notes and white papers

- AN97055 Bi-directonal level shifter for I2C-bus and other systems.

PCB Schematics

- Atmega8 Micro-controller board
- Power and USART board
- Raspberry PI
- MMA8451Q board
- LIS331DLH board
- IMU3000 board

Source Code

- Raspberry RaspIMU (Source code is also included appendix C)
- RaspIMU Control Program
- RawReader
- Atmega8 Servo Control
- Atmega8 Combined Sensor Interface
- Atmega8 IMU3000 Interface
- Atmega8 LIS331DLH Interface
- Atmega8 MMA8451Q Interface
- Xmega Pulse Counter

Test Data

- IMU3000 Static Test
- LIS331DLH Test on DSV Acergy Osprey
- MMA8451Q Test on DSV Acergy Osprey
- MPU9150 Dynamic Test (Roll, pitch and Sawtooth)
- MPU9150 Filter tuning
- MPU9150 Static Test Histograms
- MPU9150 Shake Test
- MPU9150 Statistics Test
- MPU9150 Temperature Test
- RaspIMU Attitude Stability Test
- Tachometer Test
- Data from Sognsvann Field Test

Software

- HDD Raw Copy
- Lazarus IDE
- Putty
- Realterm
- TightVNC
- Win32DiskImager

Images

- Raspbian Wheezy (09.02.2013)
- RaspIMU (18.05.2013)

B NMEA PASHR String

No.	Name	Description	Format	Example
1	Identifier	Proprietary heading, roll, pitch and heave	$PASHR	$PASHR
2	Time	Time (hhmmss.sss) UTC This field will be blank until UTC time is known.	HHMMSS.SSS	134500.125
3	Heading	(True) heading in degrees	HHH.HH	270.34
4	Type	A character is output by the navigation system to represent that the heading is to true or magnetic north.	T/M	M
5	Roll	Roll of the navigation system, measured in degrees, with leading sign, leading 0s where needed and 2 decimal places. Positive values mean that the left side is up.	RRR.RR	0.12
6	Pitch	Pitch of the navigation system, measured in degrees, with leading sign, leading 0s where needed and 2 decimal places. Positive values mean that the front is up.	PPP.PP	1.04
7	Heave	Heave of the navigation system, measured in metres, with leading sign, leading 0s where needed and 2 decimal places.	aaa.aa	0.34
8	Roll Accuracy	Roll angle accuracy estimate (stdev) in degrees	r.rrr	0.01
9	Pitch Accuracy	Pitch angle accuracy estimate (stdev) in degrees	p.ppp	0.03
10	Heading Accuracy	Heading angle accuracy estimate (stdev) in degrees	h.hhh	0.05
11	Aiding Status	GPS position mode: 0; No GPS fix, 1; GPS position fix 3; RTK integer position fix.	Q1	2
12	IMU Status	IMU status: 0; IMU is OK 1; IMU error.	Q2	1
13	Checksum	2-byte XOR sum of data to check for transmission errors	*CS	*7D
14	CR + LF	Carried return and line feed marks the end of the string	*CR *LF	#13#10

C RaspIMU Source Code

```c
// Complimentary Attitude Filterfor MPU9150 sensor
// By Johan Kleiberg Jensen - Last update: 13052013
// For UiO Master project

// Included libraries
#include <stdio.h>      // Standard IO definitions
#include <fcntl.h>      // File control definitions
#include <unistd.h>     // UNIX standard function definitions
#include <errno.h>      // Error number definitions
#include <stdlib.h>     // Standard library
#include <string.h>     // String function definitions
#include <math.h>       // Maths library
#include "i2c-dev.h"    // I2C library
#include <termios.h>    // POSIX terminal control definitions
#include <sys/time.h>   // Real-time and CPU-time library
#include <stdbool.h>    // Boolean types
//#include <wiringPi.h> // GPIO library for accessing pins

//Definitions of register MPU9150 map from data-sheet- Consider
    moving to library
static int AUX_VDDIO=1;
static int SMPLRT_DIV=25;
static int CONFIG=26;
static int GYRO_CONFIG=27;
static int ACCEL_CONFIG=28;
static int FF_THR=29;
static int FF_DUR=30;
static int MOT_THR=31;
static int MOT_DUR=32;
static int ZRMOT_THR=33;
static int ZRMOT_DUR=34;
static int FIFO_EN=35;
static int I2C_MST_CTRL=36;
static int I2C_SLV0_ADDR=37;
static int I2C_SLV0_REG=38;
static int I2C_SLV0_CTRL=39;
static int I2C_SLV1_ADDR=40;
static int I2C_SLV1_REG=41;
static int I2C_SLV1_CTRL=42;
static int I2C_SLV2_ADDR=43;
static int I2C_SLV2_REG=44;
static int I2C_SLV2_CTRL=45;
static int I2C_SLV3_ADDR=46;
static int I2C_SLV3_REG=47;
static int I2C_SLV3_CTRL=48;
static int I2C_SLV4_ADDR=49;
static int I2C_SLV4_REG=50;
static int I2C_SLV4_DO=51;
static int I2C_SLV4_CTRL=52;
static int I2C_SLV4_DI=53;
static int I2C_MST_STATUS=54;
static int INT_PIN_CFG=55;
static int INT_ENABLE=56;
static int INT_STATUS=58;
static int ACCEL_XOUT_H=59;
static int ACCEL_XOUT_L=60;
static int ACCEL_YOUT_H=61;
static int ACCEL_YOUT_L=62;
static int ACCEL_ZOUT_H=63;
static int ACCEL_ZOUT_L=64;
static int TEMP_OUT_H=65;
static int TEMP_OUT_L=66;
static int GYRO_XOUT_H=67;
static int GYRO_XOUT_L=68;
static int GYRO_YOUT_H=69;
static int GYRO_YOU_L=70;
static int GYRO_ZOUT_H=71;
static int GYRO_ZOUT_L=72;
static int MOT_DETECT_STAT_US=97;
static int I2C_SLV0_DO=99;
static int I2C_SLV1_DO=100;
static int I2C_SLV2_DO=101;
static int I2C_SLV3_DO=102;
static int I2C_MST_DELAY_CT_RL=103;
static int SIGNAL_PATH_RES_ET=104;
static int MOT_DETECT_CTRL=105;
static int USER_CTRL=106;
static int PWR_MGMT_1=107;
static int PWR_MGMT_2=108;
static int FIFO_COUNTH=114;
static int FIFO_COUNTL=115;
static int FIFO_R_W=116;
static int WHO_AM_I=117;

// MPU9150 Device options
static int SAMPLERATEDIVIDER=39;        // 1kHz/40 = 25Hz sample rate
static int DLPF_CFG=1;                  // Gives 1kHz sampling and 188Hz BW
static int BYPASS_CFG=0;                // Set up for i2c non-by-pass mode

// AK8975C Device options
static int AK_ADR=12;                   // Slave adress of magnetometer
static int AK_CNTL_MODE=1;              // Single measurement mode

// General options
static int DEBUG_MODE=1;                // If 1, program will be verbose
static int ACC_LSB=16384;               // No. of bits pr g
static int GYRO_LSB=131;                // No. of bits pr deg/s
static int GYRO_HZ=25;                  // Data output rate
static int GYRO_DT=20;                  // Time constant = 20mS

// Other definitions
static double PI=M_PI;

// Convert 2's compliment to an integer
int comp2int(int msb, int lsb)          // Converts 2's compliment to integer
{
int x=0;
if ((msb&128)==128) {
        lsb ^= 255;
        msb ^= 255;
        msb &= 127;
        x = (int)((msb<<8)+lsb+1);
        return -x;
        }
else {
        x = (int)((msb<<8)+lsb);
        return x;
        }
}

// Main procedure
int main(void)
{
// begin Variable declarations
int file;
int file_AK;
int adapter_nr=0;
char filename[40];
int i2c_addr=104;
int read_bytes=14;
char buf[16]={0};
//Other variables
float Temperature=0;
int raccx=0,raccy=0,raccz=0;
int rgyrox=0,rgyroy=0,rgyroz=0;
double accx,accy,accz,gyrox,gyroy,gyroz;
double gmagnitude;
int i,p,n;
// Matrixes with initial values
double gvector[3]={0,0,1};
double R[9]={1,0,0,0,1,0,0,0,1};
double Rup[9]={1,0,0,0,1,0,0,0,1};
double Rbuff[9]={1,0,0,0,1,0,0,0,1};
// Correction variables, experimentally set
double KIPitchRoll=512;
double KPPitchRoll=512;
double ErrorRP[3]={0,0,0};
double OmegaCorrPAcc[3]={0,0,0};
double AccIntegral[3]={0,0,0};
double OmegaCorrIAcc[3]={0,0,0};
int gyrobiasx=-60, gyrobiasy=90, gyrobiasz=70;
//int gyrobiasx=0, gyrobiasy=0, gyrobiasz=0;
// Renormalise variables
double normalisevector1[3];
double normalisevector2[3];
double error;
double normalise;
// Filter angle Output
double Roll=0;
double Pitch=0;
double Yaw=0;
// Variables for constructing output string
char SerialString[128];
char SerialTime[16];
char SerialHeading[8];
char SerialHeadingType[8];
char SerialRoll[8];
char SerialPitch[8];
char SerialHeave[8];
char SerialRollStdev[8];
char SerialPitchStdev[8];
char SerialAidingType[8];
char SerialIMUStatus[8];
char SerialChecksum[8];
char SerialEOS[8];
char SerialTemperature[8];
// Variables for time stamping
char* SystemTimeString;
```

```
struct timeval SystemTime;
unsigned SystemTimeusec;
// Other variables for parameter passing from companion program
int rx_length = 0;
unsigned char rx_buffer[32];
int rec_time;
int rec_syncrate, rec_sync;
float rec_ax,rec_ay,rec_gx,rec_gy;
int rec_ki,rec_kp;
bool set_bias=false;
bool set_time=false;
bool set_sync=false;
bool set_rate=false;
// Variable declarations done

// Open terminal (Serial/UART) port
int fd;
fd = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY | O_NDELAY);
if (fd == -1)
        {
        printf("Unable␣to␣open␣serial␣port\n");
        //EXIT!!
        }
else
        {
        if (DEBUG_MODE) printf("Serial␣port␣open\n");
        //fcntl(fd, F_SETFL, 0);
        }
// Set baudrate etc
struct termios options;
tcgetattr(fd, &options);                // Get current options
cfsetispeed(&options, B57600);          // Set baud rates
cfsetospeed(&options, B57600);
options.c_cflag &= ~PARENB;             // 8N1
options.c_cflag &= ~CSTOPB;
options.c_cflag &= ~CSIZE;
options.c_cflag |= CS8;
options.c_cflag |= (CLOCAL | CREAD);    // Enable receiver and set local mode
tcsetattr(fd, TCSANOW, &options);       // Set new options

// Write to serial port
n = write(fd, "Port␣open\n",10);
if (n<0) printf("Serial␣port␣write␣failed\n");

// Open log file for writing
FILE *fp;
fp=fopen("log.txt","w");

// Open MPU9150 on I2C
if (DEBUG_MODE) printf("Opening␣I2C␣bus\n");
sprintf(filename,"/dev/i2c-%d",adapter_nr);
if ((file=open(filename,O_RDWR))<0){
        printf("ERRNO:␣%s\n",strerror(errno));
        exit(1);}
else if (DEBUG_MODE) {printf("I2C␣Bus␣aquired!\n");}
if(ioctl(file,I2C_SLAVE,i2c_addr)<0){
        printf("Failed␣to␣aquire␣device␣(MPU9150)␣in␣I2C␣bus...\n");
        printf("ERRNO:␣%s\n",strerror(errno));
        exit(1);}
else if (DEBUG_MODE) {printf("I2C␣device␣(MPU9150)␣aquired!\n");}
```

```

// Wake up MPU9150
buf[0]=107;
buf[1]=0;
if (write(file,buf,2)!=2){
printf("Device␣failed␣to␣wake␣up!\n");}else if (DEBUG_MODE) printf(
    "MP9150␣woken!\n");

// Set samplerate divisor for MPU9150
buf[0]=SMPLRT_DIV;
buf[1]=SAMPLERATEDIVIDER;
if (write(file,buf,2)!=2) printf("Device␣write␣failed\n");

// Set lowpassfilter for MPU9150
buf[0]=CONFIG;
buf[1]=DLPF_CFG;
if (write(file,buf,2)!=2) printf("Device␣write␣failed\n");

// Set MPU9150 in non-by-pass mode to access magnetometer (slave)
buf[0]=INT_PIN_CFG;
buf[1]=BYPASS_CFG;
if (write(file,buf,2)!=2) printf("Device␣write␣failed\n");

// Main loop begins here - Run forever
for(;;){
// Check if data ready on MPU9150
// Read register 3A/58 INT_STATUS, bit0 DATA_RDY_INT see if data is
     ready
do {
usleep(1000);
buf[0]=INT_STATUS;
if (write(file,buf,1)!=1) printf("Unable␣to␣write␣to␣device!\n");
if (read(file,buf,1)!=1) printf("Unable␣to␣read␣from␣device!\n");
}while ((buf[0]&1)!=1);

// Set register adress to read
buf[0]=ACCEL_XOUT_H;
if (write(file,buf,1)!=1){printf("Unable␣to␣write␣to␣device!\n");}

// Read registers data registera (accelerometer and gyroscope)
if (read(file,buf,read_bytes)!=read_bytes){
printf("Read␣Failed!!");}

// Get current time of sample (time-stamp)
gettimeofday(&SystemTime,NULL);
SystemTimeString = ctime(&SystemTime.tv_sec);
SystemTimeusec = SystemTime.tv_usec;
SerialTime[0] = SystemTimeString[11];
SerialTime[1] = SystemTimeString[12];
SerialTime[2] = SystemTimeString[14];
SerialTime[3] = SystemTimeString[15];
SerialTime[4] = SystemTimeString[17];
SerialTime[5] = SystemTimeString[18];
SerialTime[6] = '.';
SerialTime[7] = (char)(((SystemTimeusec/100000)%10000)+48); //Quick
    'n'Dirty convertion
SerialTime[8] = (char)(((SystemTimeusec/10000)%1000)-(10*((
    SystemTimeusec/100000)%10000))+48);
SerialTime[9] = ',';
SerialTime[10] = '\0';

// Convert accelerometer and gyroscope to integer raw values
```

```
Temperature = (short)((buf[6]<<8)+buf[7]);
Temperature = (Temperature+11900)/340;
raccx = comp2int(buf[0],buf[1]);
raccy = comp2int(buf[2],buf[3]);
raccz = comp2int(buf[4],buf[5]);
rgyrox = comp2int(buf[8],buf[9]);
rgyroy = comp2int(buf[10],buf[11]);
rgyroz = comp2int(buf[12],buf[13]);
fprintf(fp,"%2.2f,%i,%i,%i,%i,%i,%i\n",Temperature,raccx,raccy,
    raccz,rgyrox,rgyroy,rgyroz);

// Set bias (if command to set bias has been recieved)
if (set_bias == true)
        {
        gyrobiasx = rgyrox;
        gyrobiasy = rgyroy;
        gyrobiasz = rgyroz;
        set_bias = false;
        }

// Remove bias from gyro
rgyrox = rgyrox - gyrobiasx;
rgyroy = rgyroy - gyrobiasy;
rgyroz = rgyroz - gyrobiasz;

// Convert accelerometer and gyroscope data to real units
accx = (double) raccx/ACC_LSB;                  // Gives values in
    mg
accy = (double) raccy/ACC_LSB;
accz = (double) raccz/ACC_LSB;
gyrox = (double) rgyrox/(GYRO_LSB*GYRO_DT);     // Gives values in
    deg/sec
gyroy = (double) rgyroy/(GYRO_LSB*GYRO_DT);
gyroz = (double) rgyroz/(GYRO_LSB*GYRO_DT);
// Deggrees to radians
gyrox = gyrox * PI/180;
gyroy = gyroy * PI/180;
gyroz = gyroz * PI/180;
gyroz = 0;              // Keep 0 until compass is implemented

// Apply Corrections from previous data
gyrox = gyrox + OmegaCorrPAcc[0] + OmegaCorrIAcc[0];
gyroy = gyroy + OmegaCorrPAcc[1] + OmegaCorrIAcc[1];
gyroz = gyroz + OmegaCorrPAcc[2] + OmegaCorrIAcc[2];

// Accelerometer to vector calculations
gmagnitude = sqrt(pow(accx,2)+pow(accy,2)+pow(accz,2));
gvector[0] = accx/gmagnitude;
gvector[1] = accy/gmagnitude;
gvector[2] = accz/gmagnitude;

// Convert to Matrix form
// Construct update matrix
Rup[0] = 1;
Rup[1] = -gyroz;
Rup[2] = gyroy;
Rup[3] = gyroz;
Rup[4] = 1;
Rup[5] = -gyrox;
Rup[6] = -gyroy;
Rup[7] = gyrox;
Rup[8] = 1;
```

```

// Multiply Update matrix to rotation matrix and add back
// R=R*Rup;
Rbuff[0] = R[0]*Rup[0]+R[1]*Rup[3]+R[2]*Rup[6];
Rbuff[1] = R[0]*Rup[1]+R[1]*Rup[4]+R[2]*Rup[7];
Rbuff[2] = R[0]*Rup[2]+R[1]*Rup[5]+R[2]*Rup[8];
Rbuff[3] = R[3]*Rup[0]+R[4]*Rup[3]+R[5]*Rup[6];
Rbuff[4] = R[3]*Rup[1]+R[4]*Rup[4]+R[5]*Rup[7];
Rbuff[5] = R[3]*Rup[2]+R[4]*Rup[5]+R[5]*Rup[8];
Rbuff[6] = R[6]*Rup[0]+R[7]*Rup[3]+R[8]*Rup[6];
Rbuff[7] = R[6]*Rup[1]+R[7]*Rup[4]+R[8]*Rup[7];
Rbuff[8] = R[6]*Rup[2]+R[7]*Rup[5]+R[8]*Rup[8];
R[0] = Rbuff[0];
R[1] = Rbuff[1];
R[2] = Rbuff[2];
R[3] = Rbuff[3];
R[4] = Rbuff[4];
R[5] = Rbuff[5];
R[6] = Rbuff[6];
R[7] = Rbuff[7];
R[8] = Rbuff[8];

// Renormalise rotation matrix (to keep orthogonal)
normalisevector1[0] = R[0];
normalisevector1[1] = R[1];
normalisevector1[2] = R[2];
normalisevector2[0] = R[3];
normalisevector2[1] = R[4];
normalisevector2[2] = R[5];

error = -0.5*(normalisevector1[0]*normalisevector2[0] +
    normalisevector1[1]*normalisevector2[1] + normalisevector1[2]*
    normalisevector2[2]);

Rbuff[0] = R[0] * error;
Rbuff[1] = R[1] * error;
Rbuff[2] = R[2] * error;
Rbuff[3] = R[3] * error;
Rbuff[4] = R[4] * error;
Rbuff[5] = R[5] * error;

Rbuff[0] = Rbuff[0] + R[0];
Rbuff[1] = Rbuff[1] + R[1];
Rbuff[2] = Rbuff[2] + R[2];
Rbuff[3] = Rbuff[3] + R[3];
Rbuff[4] = Rbuff[4] + R[4];
Rbuff[5] = Rbuff[5] + R[5];
Rbuff[6] = ((Rbuff[1]*Rbuff[5]) - (Rbuff[2]*Rbuff[4]));
Rbuff[7] = ((Rbuff[2]*Rbuff[3]) - (Rbuff[0]*Rbuff[5]));
Rbuff[8] = ((Rbuff[0]*Rbuff[4]) - (Rbuff[1]*Rbuff[3]));

normalise = 1/sqrt(Rbuff[0]*Rbuff[0] + Rbuff[1]*Rbuff[1] + Rbuff
    [2]*Rbuff[2]);
R[0] = Rbuff[0] * normalise;
R[1] = Rbuff[1] * normalise;
R[2] = Rbuff[2] * normalise;

normalise = 1/sqrt(Rbuff[3]*Rbuff[3] + Rbuff[4]*Rbuff[4] + Rbuff
    [5]*Rbuff[5]);
R[3] = Rbuff[3] * normalise;
R[4] = Rbuff[4] * normalise;
R[5] = Rbuff[5] * normalise;
```

```

normalise = 1/sqrt(Rbuff[6]*Rbuff[6] + Rbuff[7]*Rbuff[7] + Rbuff
    [8]*Rbuff[8]);
R[6] = Rbuff[6] * normalise;
R[7] = Rbuff[7] * normalise;
R[8] = Rbuff[8] * normalise;

// Drift Calculations
ErrorRP[0] = ((gvector[1]*R[8])-(gvector[2]*R[7]))/65536;
ErrorRP[1] = ((gvector[2]*R[6])-(gvector[0]*R[8]))/65536;
ErrorRP[2] = ((gvector[0]*R[7])-(gvector[1]*R[6]))/65536;

OmegaCorrPAcc[0] = ErrorRP[0]*KPPitchRoll;
OmegaCorrPAcc[1] = ErrorRP[1]*KPPitchRoll;
OmegaCorrPAcc[2] = ErrorRP[2]*KPPitchRoll;

AccIntegral[0] = AccIntegral[0] + (ErrorRP[0]*KIPitchRoll)/(1024*
    GYRO_DT);
AccIntegral[1] = AccIntegral[1] + (ErrorRP[1]*KIPitchRoll)/(1024*
    GYRO_DT);
AccIntegral[2] = AccIntegral[2] + (ErrorRP[2]*KIPitchRoll)/(1024*
    GYRO_DT);

OmegaCorrIAcc[0] = AccIntegral[0];
OmegaCorrIAcc[1] = AccIntegral[1];
OmegaCorrIAcc[2] = AccIntegral[2];

//Retrieve Euler Angels
Pitch = (asin(R[7]))*180/PI;
Roll  = (atan2(-R[6],R[8]))*180/PI;
Yaw   = (atan2(-R[1],R[4]))*180/PI;

// Construct string to write to serial port
SerialString[0] = NULL;
sprintf(SerialTemperature, "%3.1f,", Temperature);
sprintf(SerialRoll, "%3.2f,", Roll);
sprintf(SerialPitch, "%3.2f", Pitch);
sprintf(SerialEOS, "\r\n");
strcat(SerialString,SerialTime);
strcat(SerialString,SerialTemperature);
strcat(SerialString,SerialRoll);
strcat(SerialString,SerialPitch);
strcat(SerialString,SerialEOS);

// Write serial string to serial port
n = write(fd,SerialString,strlen(SerialString));
if (n<0) printf("Serial␣write␣failed!\n");

// Check if commands have been recieved on USART (USART is in non-
    blocking read)
        // Read the buffer - get length first
        rx_length = read(fd,(void*)rx_buffer,2);
        if (rx_length < 0)
                {
                // Error while reading USART
                }
        else if (rx_length == 0)
                {
                //No data waiting
                }
        else
                {
```

```
printf("\nSerial␣data␣received");
// Bytes have been recieved - Check if they are commands
if ((rx_buffer[0] == 'S') && (rx_buffer[1] == 'B'))
        {
        // Set bias in next cycle
        printf("\nSet␣Bias");
        set_bias = true;
        }
if ((rx_buffer[0] == 'T') && (rx_buffer[1] == 'I'))
        {
        rx_length = read(fd,(void*)rx_buffer,8);
        if (rx_length>7)
                {
                // Get time
                rec_time = ((int)rx_buffer[0]-48)*100000 + ((int)rx_buffer[1]-48)*10000 + ((int)rx_buffer[3]-48)*1000 + ((int)rx_buffer[4]-48)*100 + ((int)rx_buffer[6]-48)*10 + ((int)rx_buffer[7]-48);
                printf("\nTime␣synchronised:␣%i", rec_time);
                printf("\n");
                }
        }
if ((rx_buffer[0] == 'S') && (rx_buffer[1] == 'Y'))
        {
        rx_length = read(fd,(void*)rx_buffer,1);
        if (rx_length>0)
                {
                rec_sync = (int)rx_buffer[0] - 48;
                // Disable external sync command
                if (rx_buffer[0] == '0') {set_sync = false;}
                // Enable external sync command
                if (rx_buffer[0] == '1') {set_sync = true;}
                }
        }
if ((rx_buffer[0] == 'S') && (rx_buffer[1] == 'R'))
        {
        rx_length = read(fd,(void*)rx_buffer,1);
        if (rx_length>0)
                {
                // Get external sync. rate
                rec_syncrate = (int)rx_buffer - 48;
                if (rx_buffer[0] == '0') {printf("\nSynchronization␣rate␣20Hz");}
                if (rx_buffer[0] == '1') {printf("\nSynchronization␣rate␣10Hz");}
                if (rx_buffer[0] == '2') {printf("\nSynchronization␣rate␣5Hz");}
                if (rx_buffer[0] == '3') {printf("\nSynchronization␣rate␣2.5Hz");}
                if (rx_buffer[0] == '4') {printf("\nSyncronization␣rate␣1.25Hz");}
                printf("\n");
                }
        }
if ((rx_buffer[0] == 'P') && (rx_buffer[1] == 'A'))
```

```
                            {
                            rx_length = read(fd,(void*)rx_buffer,27);
                            if (rx_length>26)
                                    {
                                    // Recieve advanced filter options
                                        and scaling
// Accelerometer and gyroscope scaling
rec_ax = (int)rx_buffer[0]-48 + ((int)rx_buffer[2])/10 + ((int)
    rx_buffer[3])/100 + ((int)rx_buffer[4])/1000;
rec_ay = (int)rx_buffer[5]-48 + ((int)rx_buffer[7])/10 + ((int)
    rx_buffer[8])/100 + ((int)rx_buffer[9])/1000;
rec_gx = (int)rx_buffer[10]-48 + ((int)rx_buffer[12])/10 + ((int)
    rx_buffer[13])/100 + ((int)rx_buffer[14])/1000;
rec_gy = (int)rx_buffer[15]-48 + ((int)rx_buffer[17])/10 + ((int)
    rx_buffer[18])/100 + ((int)rx_buffer[19])/1000;
// K and I filter factors
rec_ki = ((int)rx_buffer[20]-48)*1000 + ((int)rx_buffer[21])*100 +
    ((int)rx_buffer[22])*10 + ((int)rx_buffer[23]);
rec_kp = ((int)rx_buffer[24]-48)*1000 + ((int)rx_buffer[25])*100 +
    ((int)rx_buffer[26])*10 + ((int)rx_buffer[27]);
KIPitchRoll = rec_ki;
KPPitchRoll = rec_kp;
                                    }
                            printf("\n");
                            }
                    }
} // Loop returns here!

//Close file
fclose(fp);
if (file>=0) close(file);
printf("Log␣file␣closed\n");

// Exit gracefully
printf("Done...\n");
return 0;
}
```