Center for sensor systems (ZESS)

North finding system using low-cost MEMS inertial measurement unit

By

Hiwa Faiq Gul Muhammad

A thesis
submitted in partial fulfillment
of the requirements for the degree of
master of science in
mechatronics
at

the University of Siegen

September ٢٠١٢

North finding system using low-cost MEMS inertial measurement unit

Thesis Approved:

Supervisors

Prof. Dr. Otmar Loffeld

MSc. Ezzaldeen Edwan

Acknowledgements

I wish to express my sincere appreciation to my supervisors Prof. Dr.Otmar Loffeld and MSc.Ezzaldeen Edwan for their advice, guidance, encouragement and friendly supervision. My sincere appreciation to Mr. Wolf Twelsiek and Mr. Rolf Wurmbach for their support and supplying me all the hardware components needed in this project. Special thanks to my Friend Fernando Suárez Lainez for his kind help and support during the implementation phase of my thesis. Thanks to the DAAD scholarship program and MOHESR for funding me during my master program.

I would also like to give my special thanks to my family for their continuous support and encouragement during this work.

Abstract

Heading determination is said to be one of the important requirements in navigation systems and can be performed by different methods. Advances in Global Positioning System (GPS) technology, which is easy and low-cost, make GPS more preferable and alternative for the navigation systems. The problem with navigation systems that utilizes GPS only for heading calculation is that the GPS signals are not available in indoors and in underwater operational environment. Another way to find heading is by using a digital magnetic compass (DMC). These devices are compact and low-cost instruments that are capable of achieving an accuracy of milidgrees. However, heading accuracy of such devices is highly dependent on the working environment and can be easily degraded by a ferrous material or by electromagnetic interference.

This thesis examines the determination of the heading using low-cost MEMS inertial measurement sensors Gyroscopes and accelerometers. These devices can overcome the problems like degradation by ferrous material and electromagnetic field and can be used in indoor navigation since they are working on inertial principle. This method is based on the detection of earth rotation which is very small as compared to sensor range, so detecting such a weak signal requires precise error analysis and characterization. Tests and results of the inertial sensors are implemented with different approaches. The results show that the MEMS sensors that have been used in this work can be used for heading determination with a certain value of error.

Table of contents

١ Introduction

١,١ Thesis objective

The fundamental purpose of this thesis is to find north direction using low-cost MEMS gyroscope by different proposed methods. Error characterization associated with a MEMS gyroscope, most importantly bias instability, has also been studied. Bias instability decides whether the selected sensor can be used to accomplish required task or not.

Inertial sensor calibration is performed for both gyroscopes and accelerometers in order to find their scale factor, misalignment, static bias. The sensor calibration is also useful to find the rotation matrices to convert the sensor values from the sensors coordinate system to a unified coordinate system, in our case a triad frame.

١,٢ Literature survey

Heading calculation using low-cost IMU by the proposed methods requires error identification and compensation. A detailed error identification and analysis using Allan variance has been developed for this purpose [Haiying Hou], it defines and classifies the common errors found in gyroscopes and rate gyros: bias, scale factor, misalignment, and noise. A procedure has also been developed to obtain the statistics of each of these error sources. Development of a MEMS gyroscope for north finding applications is proposed using tuning fork type MEMS gyro [Burdess].The Angle Random Walk (ARW) and root Allan variance has been measured and found that it satisfies the requirement for gyro compassing. Then the long-term bias drift has been compensated for using Carouseling technique, for which the Earth rotation could be found successfully.

A system for finding true north with rotating linear accelerometers utilizing the Coriolis Effect to detect the horizontal component of the earth's spin rate requires expensive and precise accelerometer [Guofu]. Beside mathematical modeling, a signal processing algorithm to deal with the accelerometer output is presented.

Direct measurement of the Earth's rotation using Low-cost MEMS rate gyroscope is difficult to achieve due to considerable parameter variations of the current state-of-art sensors of this type [Rumen]. The external factors that affect the sensor measurement are modeled and compensated

mechanically by changing the sensor's orientation. Here the drift in the gyroscope is compensated for and the gravity effect on the gyro is also eliminated resulting in more accurate estimate of the Earth's rotation.

١,٣ Achievements of this thesis

The following are the achievements of the thesis.

- MEMS inertial sensors error characterization
- Gyro and accelerometer calibrations
- Heading calculation by Maytagging method
- Gyro-bias compensation using distributed accelerometer triad

١,٤ Organization of thesis

Chapter ١ presents the thesis objectives, achievements, literature survey, and Organization of thesis.

Chapter ٢ reviews theory and background of MEMS gyroscope and accelerometer. Error characterization is also presented using Allan variance method. Finally the sensor calibration is described for both gyro and accelerometer.

Chapter ٣ provides the hardware specifications for both of development board and the sensors that are used for system realization. The circuit that has been used for data collection from the sensors into the development board has been presented and the software that works behind the hardware is also illustrated.

Chapter ٤ includes the analysis and the experimental results.

Chapter ٥ presents the thesis conclusion and provides the recommendation of future research.

۲ Inertial sensors, error characterization, and calibration

۲,۱ MEMS inertial sensors

Micro-Electromechanical Systems, or MEMS, can be used to produce complex structures, devices and systems. MEMS refer to devices that have characteristic length of less than ۱ mm but more than ۱ micron, that combine electrical and mechanical components and which are fabricated using integrated circuit batch-processing technologies.

Most sensors are designed to convert a physical phenomenon into a measurable signal. For inertial sensors, this physical phenomenon is an inertial force. Often this force is converted into a linearly scaled voltage output with a specific sensitivity.

In mechanical sensors, the active structural elements convert a mechanical external input signal (force, pressure, acceleration, etc) into an electrical signal output (voltage, current, or frequency) applying an external force to the active part of the sensor. Active parts usually are elements such as suspended beams or membranes [۱۲].

In electromechanical conversion, the mechanical quantity is transformed into an electrical quantity such as capacitance, resistance or charge. Often, the electrical signal needs further electrical conversion into an output voltage, frequency or current. To optimize all the transfer functions, detailed electrical and mechanical modeling is required.

MEMS sensors can suffer in overall system sensitivity. The tradeoff between system size, cost and performance is often directly coupled. It should be taken into account that the properties of thin-film materials are often significantly different from their bulk or macro-scale form. Assumption of homogeneity, commonly used with accuracy for bulk materials, becomes unreliable when used to model devices that have dimensions on the same scale as individual grains and other microscopic fluctuations, affecting the properties of the material. Thus, local changes in grain size and other characteristics could significantly alter the performance of MEMS produced devices. Many different inertial microsensors have been made (e.g. single- and multi-axis accelerometers and gyroscopes) using either piezoresistors or capacitive position detection [۱۲].

٢,١,١ Accelerometers

A linear accelerometer is an inertial sensor that measures the component of translational acceleration along its input axis. An output signal is produced from the motion of a proof mass relative to the case, or from the force or torque required to restore the proof mass to a null position relative to the case [١٣].

MEMS accelerometers can be classified into different categories dependent upon following three parameters: the position detection of the seismic mass (piezoresistive signal pick-off sensors, capacitive signal pick-off sensors, piezoelectric sensing element sensors and resonant element sensors), the operation mode (open loop operation and closed loop operation), and the fabrication process of the sensing elements[١٢].

An open-loop silicon micromechanical accelerometer can sense proof-mass displacement piezoelectrically, piezoresistively, or electrostatically. Piezoelectric sensing of the strain in the proof-mass support restricts measurement to AC (above about ٥ Hz) rather than DC inputs because of leakage of the piezoelectric generated charge [١٣], [١٤].

Generally, the proof mass is suspended by compliant beams anchored to a fixed frame. The proof mass has a mass of M, the suspension beams have an effective spring constant of K, and there is a damping factor D affecting the dynamic movement of the mass, see Fig. ٢,١. External acceleration displaces the support frame relative to the proof mass, which in turn changes the internal stress in the suspension spring. Both the relative displacement and the suspension-beam stress can be used as a measure of the external acceleration [١٥].

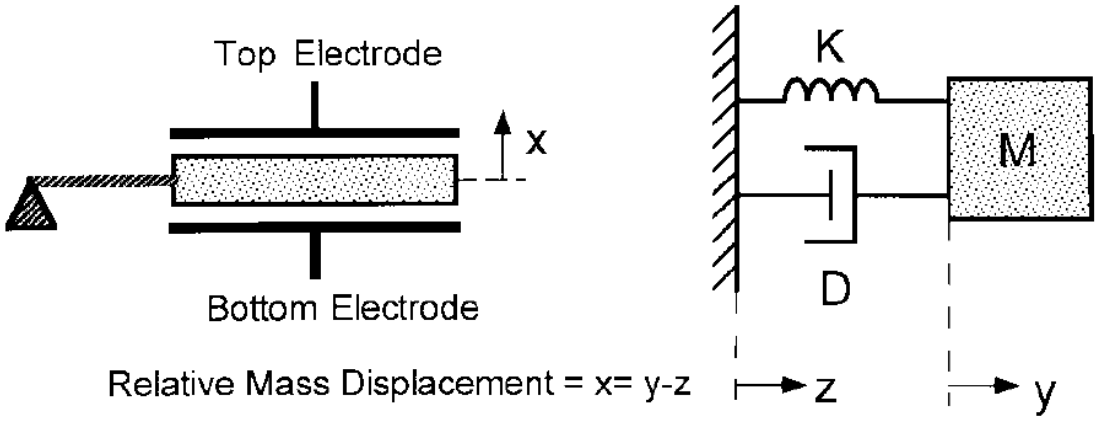

Figure ٢,١: General accelerometer structure and its mechanical lumped model [after ١٧]

٢,١,٢ Gyroscopes

A gyroscope is an inertial sensor that measures angular rotation about its input axis with respect to inertial space. The sensing of such motion could utilize the angular momentum of a spinning rotor, the Coriolis effect on a vibrating mass, or the Sagnac effect[١] on counter-propagating light beams in a ring laser or an optical fiber-coil[١٣].

Coriolis acceleration, named after Gaspard-Gustave Coriolis (١٧٩٢-١٨٤٣), is a fictitious acceleration (and not a real force, since it is based on motion relative to a non-inertial reference frame, which is rotating) that arises in a rotation reference frame and is proportional to the rate of rotation. To understand the Coriolis effect, imagine a particle traveling in space with a velocity vector v. An observer sitting on the x-axis of the xyz coordinate system, shown in Fig. ٢,٢, is watching this particle. If the coordinate system along with the observer starts rotating around the z-axis with an angular velocity Ω, the observer thinks that the particle is changing its trajectory towards the x-axis with an acceleration equal to ٢V × Ω. Although no real force has been exerted on the particle, to an observer attached to the rotating reference frame an apparent force has resulted that is directly proportional to the rate of rotation. This effect is the basic operating principle underlying all vibratory structure gyroscopes [١٥].

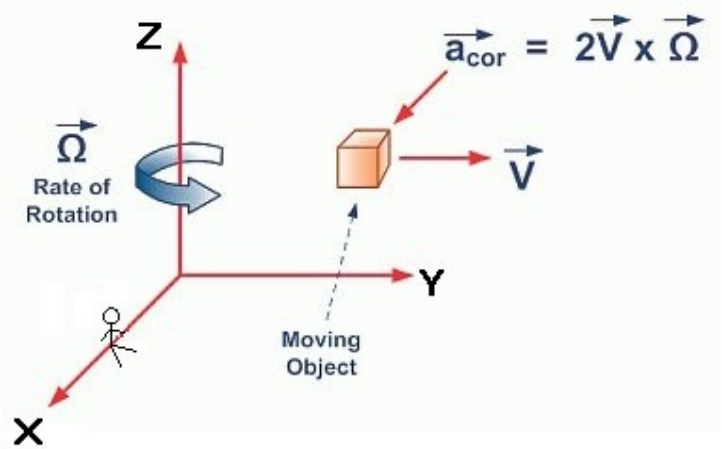

Figure ٢,٢: The Coriolis effect [after ١٧]

[١] The Sagnac effect (also called Sagnac Interference) is a phenomenon encountered in interferometry that is elicited by rotation.

٢,٢ Error characterization

There are various sources of errors that affect the performance of inertial sensors. Apart of noise in the system, the most commonly present errors are biases, scale factors, non-linearities and axes misalignment. These errors are briefly discussed in the following subsections.

٢,٢,١ Bias

The bias of MEMS sensors is the average of output signal that has no relation with the input quantity sensed by the sensors. That is even though there is no force acting onto the sensors; the sensors produce a non-zero output [١٦].

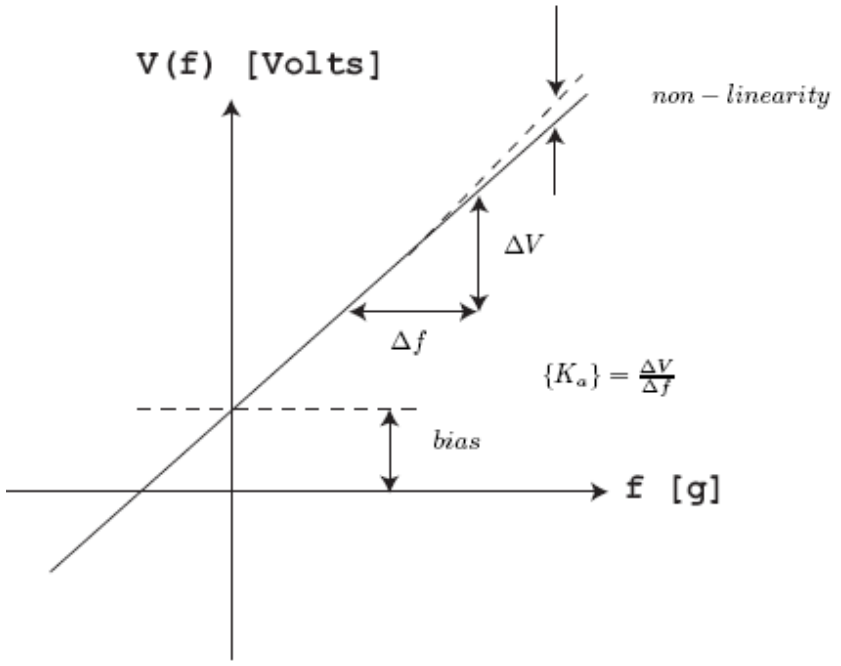

Figure ٢.٣: The relationship between the output voltage of the accelerometer (gyroscope)and the measured force (angular rate) is modeled as a linear function describing the scaling and bias of the sensors [after ١٦].

The relationship between the output voltage and the physical quantity acting along the sensor sensitive axes is given by the manufactures data sheet, but the true scaling may vary from sensor to sensor. Bias can be split into a static part called bias offset (refers to the offset in the measurement), a random part called bias drift and a temperature varying part. Both the bias offset and the temperature varying part are deterministic in nature and

therefore can be determined by calibration. The bias drift, on the other hand, are random in nature and should be modeled as a stochastic process.

٢,٢,٢ Scale factor

Scale factor is the ratio of change between the measured output and the change in sense input. It is generally evaluated as a slope of the straight line that can be fit by least square method to input-output data and is typically expressed as a percentage or ppm (parts per million).like the case of bias, scale factor can be divided into three parts, a static part, a random drift part and a temperature varying part [١٦].

٢,٢,٣ Misalignment

Axes misalignment is the error from the imperfection of mounting the sensors. It often results in a non-orthogonality of the axes. As a result, each axis is affected by measurements of the other two axes in the body frame. Since axes misalignments are a manufacturing imperfection can therefore easily be detect and compensated by calibration [١٦].

٢,٢,٤ Non-linearity

Non-linearity is the deviation of the sensor output from the input-output derived from a least square method over the operating range. The deviation is expressed as a percentage of the full-scale output.

٢,٢,٥ Allan variance and gyroscope error characterization

Allan variance was developed by David W. Allan in ١٩٦٦ ("Statistic of Atomic Frequency Standards"). The Allan variance method of data analysis is a time domain analysis technique originally developed to study the frequency stability of oscillators. In general, the method can be applied to analyze the noise characteristics of any precision measurement instrument. The attractiveness of this method is that the Allan variance, when plotted in logarithmic scales, can discriminate different contributing error sources by simply examining the varying slopes on the Allan plot. Because of the analogies, the Allan variance method analysis has been adapted to random drift characterization of inertial sensors (IEEE Sdt.٩٥٢-١٩٩٧).

Methodology

Assume that there are N consecutive data points, each having a sample time of τ_0, then

- Form groups of n consecutive data points τ_0, ٢ τ_0,.. n τ_0, with ١ ≤ n < N/٢, see Fig.٢.٤
- Each group is called cluster. Associated with each cluster is a time τ, which is equal to n τ_0.
- Obtain averages of the sum of the data points contained in each cluster over the length of that cluster

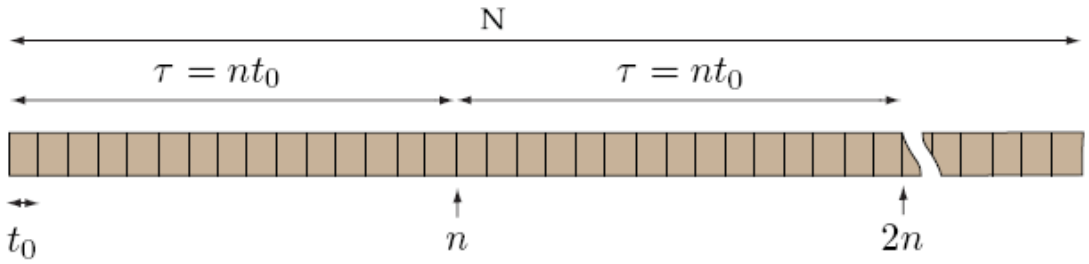

Figure ٢.٤: Scheme of data structure used in Allan variance algorithm [after ١٦]

Consequently, the Allan variance[٢] is defined as [٩]:

$$AVAR^2(\tau) = \frac{1}{2(m-1)} \sum_i \left(y(\tau)_{i+1} - y(\tau)_i \right)^2 \qquad (٢.١)$$

Where AVAR(τ) is the Allan Variance as a function of the averaging time, τ; y_i is the average value of the measurement in bin i; and m is the total number of bins.

A log-log plot of the square root of the Allan variance, AVAR(τ), versus τ provides a means of identifying and quantifying various noise terms that exist in the inertial sensor data.

Noise source analysis

In general, any number of random noise components may be present in the data depending on the type of device and the environment in which the data is obtained. If the noise sources are statically independent, then the computed Allan variance is sum of the squares of each error type [١٦].

The following subsections will show the most affective type of noise which affects the output data of the sensors.

---

[٢] Frequently the term Allan variance is also used to refer to its square root, $\sigma(\tau)$

Bias instability

Bias Instability is also known as "$1/f$ noise" or "flicker noise". This is a low frequency bias fluctuation in the measured rate data. The origin of this noise is the electronics, or other components susceptible to random flickering. Because of its low-frequency nature, it shows as the bias fluctuations in the data.

Bias Instability is a fundamental measure of the 'goodness' of a gyro. It is defined as the minimum point on the Allan Variance curve as shown in Fig.٢،٥, usually measured in °/hr. It represents the best bias stability that could be achieved for a given gyro, assuming that bias averaging takes place at the interval defined at the Allan Variance minimum.

Combined effects of all noise sources

In general, any number of random processes can be present in the data. Thus, a typical Allan variance plot looks like the one shown in the Fig. ٢.٥. Experience shows that in most cases, different terms appear in different regions of τ. This allows easy identification of various random processes that exist in the data. If it can be assumed that the existing random processes are all statistically independent then it can be shown that the Allan variance at any given τ is the sum of Allan variances due to the individual random processes at the same τ [١٦].

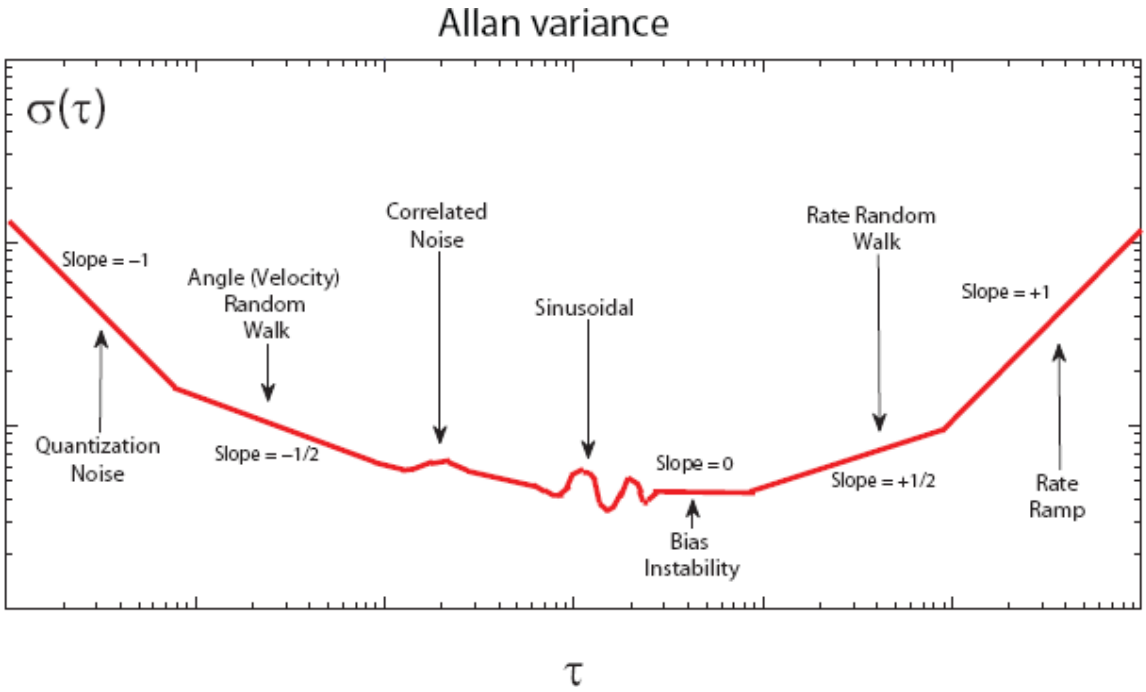

Figure ٢.٥: σ (τ) Sample plot of square root of Allan variance analysis results (after IEEE Std.٩٥٢-١٩٩٧).

٢,٣ Calibration of inertial sensors

Calibration is the process of comparing instrument outputs with known reference information and determining coefficients that force the output to agree with the reference information over a range of output values. Calibration is necessary, because the sensor sensitivity axes usually do not coincide exactly with the body frame axes. This is due to manufacturing imperfections when soldering the sensors onto the board as well as imperfections of the sensors themselves. This may also include non-orthogonality of the sensitivity axes in addition to simple misalignment, especially if a cluster of one-axis sensors is used. As actual scale and bias values usually differ from the nominal values, they have to be determined, too. The calibration process requires a mechanical platform to precisely manipulate the IMU. A minimum of one actuated degree of freedom is needed to calibrate the gyroscopes. However such a system requires extensive and tedious user manipulation, as the IMU has to be repositioned several times. Thus, it is desirable to have three degrees of freedom platform able to rotate the IMU around arbitrary axes in space, minimizing the necessary user interactions.

Another purpose of calibration in this work was to find transformation matrix (as well as misalignment and scale factors) to transform data from the sensor coordinates to a unified coordinate called triad coordinate, since there is more than one sensor and the orientation of each is different from the others. The calibrations stated in the next sections is performed for the last task of this thesis (section ٤,٣), because we have also performed the calibration for (section ٤,١) but it we are not going to describe it since it is similar to the one used in section (٤,٣).

٢,٣,١ Accelerometer calibrations

Accelerometers suffer from several sources of errors, namely misalignment, scale factor, bias, and noise. We can calibrate for three types of errors: scale factor, misalignment, and the three bias parameters in a single calibration procedure of each sensor. A ٩-element sensitivity matrix can be found, whose diagonal elements contain misalignment parameters. Likewise, we find the three bias elements [١].

In matrix form, the accelerometer triad output is related to the specific force as [١]:

$$\begin{bmatrix} Z_x \\ Z_y \\ Z_z \end{bmatrix} = \begin{bmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{bmatrix} \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} + \begin{bmatrix} b_{ax} \\ b_{ay} \\ b_{az} \end{bmatrix} + \begin{bmatrix} v_{ax} \\ v_{ay} \\ v_{az} \end{bmatrix} \qquad (٢.٢)$$

There are many methods to solve for m and b elements. Following a similar approach to the one given in, we rearrange equation (٢.٢) into the following form [١]:

$$z = O\underline{\theta} + \underline{v}_a \qquad (٢.٣)$$

$$\underline{\theta} = \begin{bmatrix} \underline{m}1 \\ \underline{m}2 \\ \underline{m}3 \\ \underline{b}_a \end{bmatrix}, \quad \underline{m}1 = \begin{bmatrix} m_{xx} \\ m_{xy} \\ m_{xz} \end{bmatrix}, \quad \underline{m}2 = \begin{bmatrix} m_{yx} \\ m_{yy} \\ m_{yz} \end{bmatrix}, \quad \underline{m}3 = \begin{bmatrix} m_{zx} \\ m_{zy} \\ m_{zz} \end{bmatrix}$$

$$\underline{v}_a = \begin{bmatrix} v_{ax} & v_{ay} & v_{az} \end{bmatrix}^T \qquad (٢.٤)$$

$$\underline{b}_a = \begin{bmatrix} b_{ax} & b_{ay} & b_{az} \end{bmatrix}^T$$

The observation matrix for every measurement is given by [١]

$$O = \begin{bmatrix} a_x & a_y & a_z & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & a_x & a_y & a_z & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_x & a_y & a_z & 0 & 0 & 1 \end{bmatrix} \qquad (٢.٥)$$

We stack N observation matrices from N observations to form the total

Measurement vector and the total observation matrix shown next [١]:

$$\begin{bmatrix} \underline{z}^1 \\ . \\ . \\ \underline{z}^N \end{bmatrix} = \begin{bmatrix} O^1 \\ . \\ . \\ O^N \end{bmatrix} \underline{\theta} + \begin{bmatrix} \underline{v}^1 a \\ . \\ . \\ \underline{v}_a{}^N \end{bmatrix}, \quad z^t = O^t \underline{\theta} + \underline{v}^t a \qquad (٢.٦)$$

To have an observable system, we need a minimum of four measurements with proper values of the specific force values. To extract the m and b elements, we use least square estimation [١]

$$\dot{\underline{\theta}} = (\quad)^{-1} O^{tT} \underline{z}^{t} \qquad (٢.٧)$$

The implementation of these equations in MATLAB is shown in appendix II

The four measurements have been taken with a different frame orientation with data logging for approximately ٤٥ seconds for each position. First the Z-axis has been pointed upward (Fig. ٢.٦) and then Y-axis pointed upward (Fig. ٢.٧) and then X-axis (Fig. ٢.٨) and finally Z-axis pointed downward (Fig. ٢.٩).

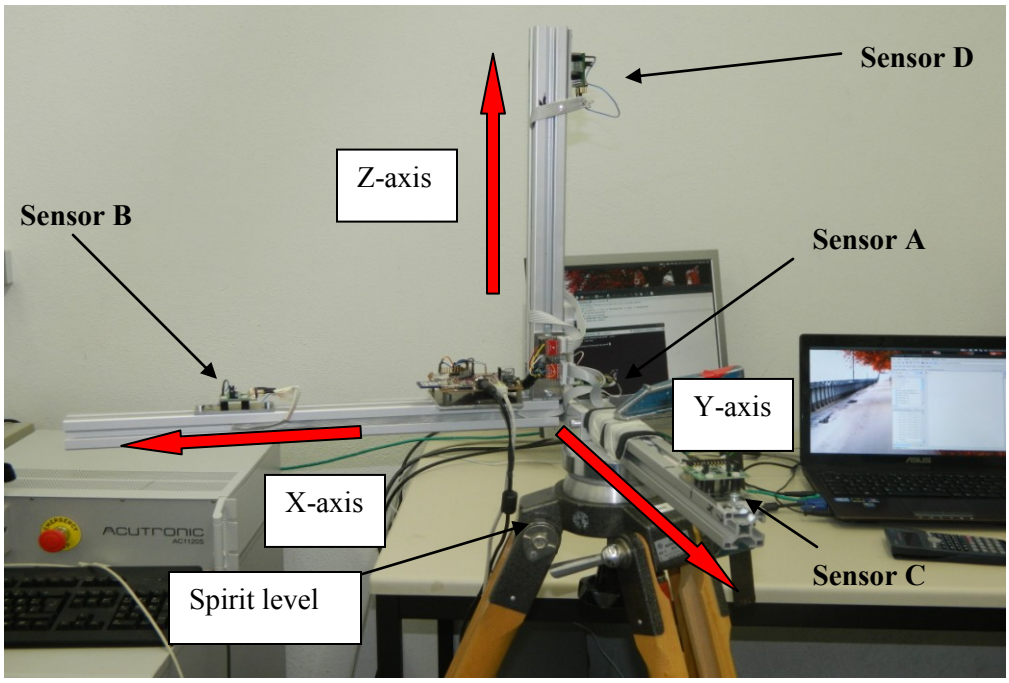

Figure ٢.٦: Z-axis upward position

So called (spirit level or bubble level) shown in the Fig.٢,٦ is used to obtain a horizontal alignment of the frame (zero degree with the local horizontal plane) to reduce errors (false reading) in the accelerometers while we are performing calibration.

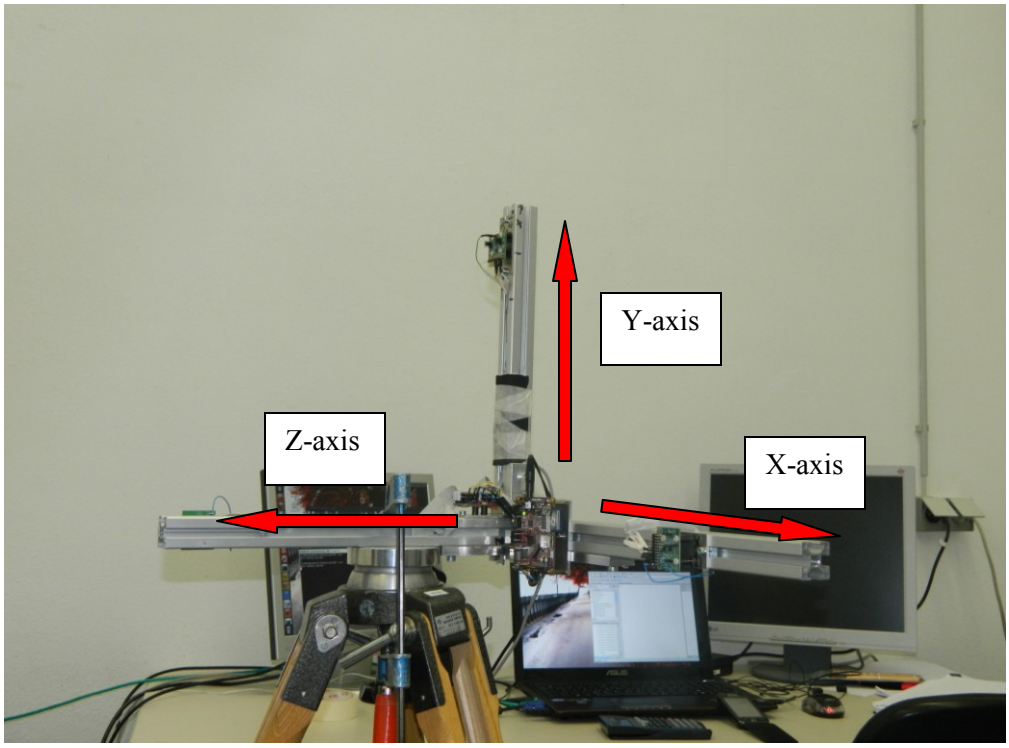

Figure ٢.٧: Y-axis upward position

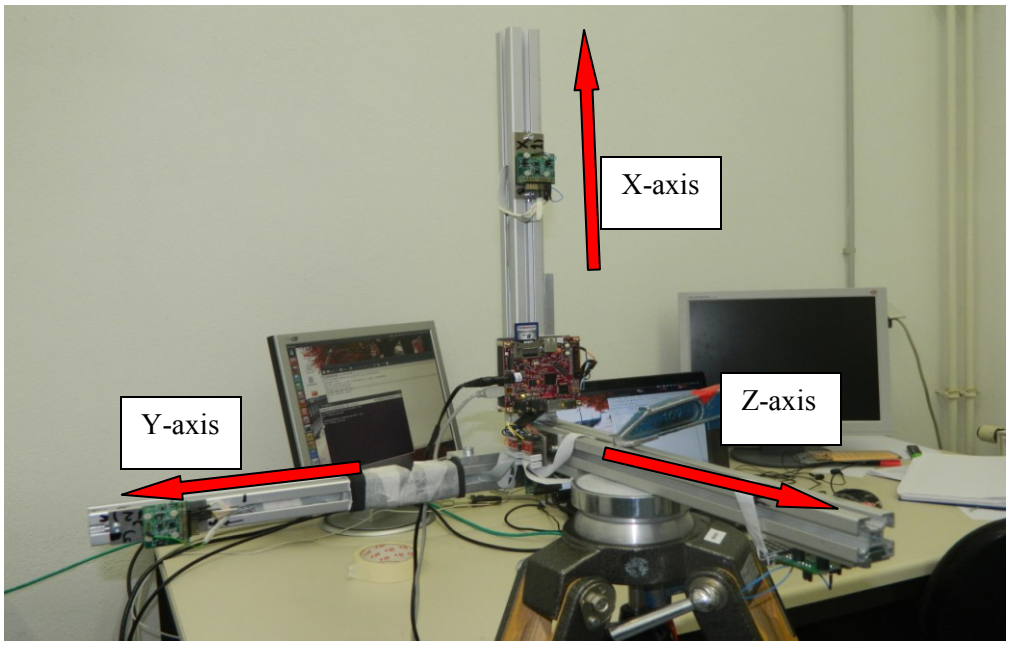

Figure ٢.٨: X-axis upward position

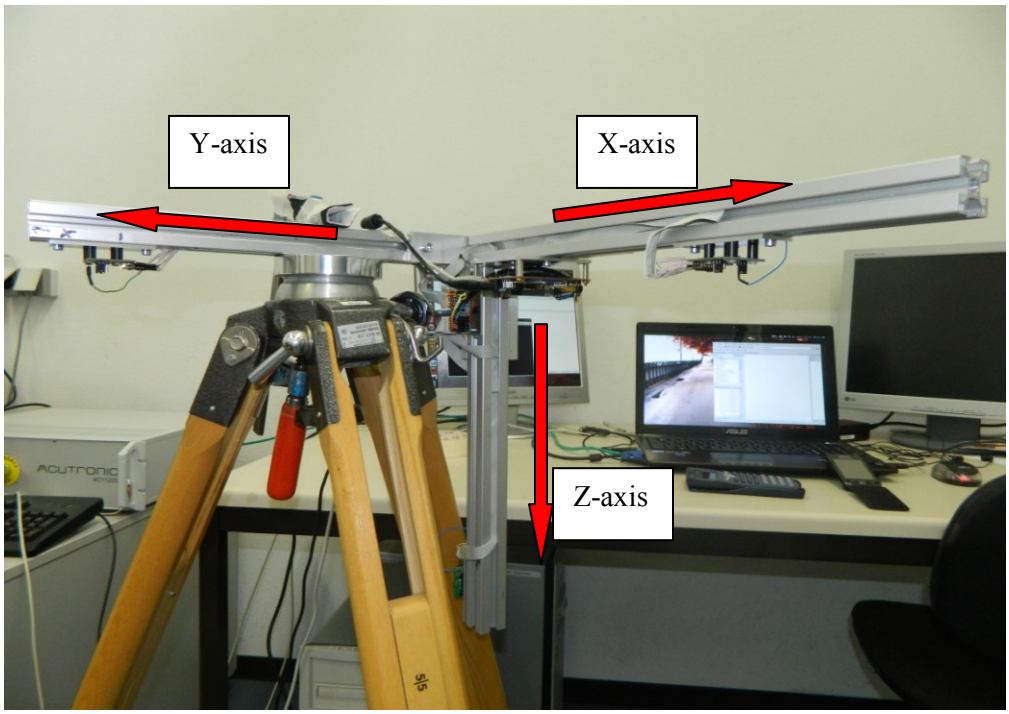

Figure ٢.٩: Z-axis downward position

The sensors have been labeled so that it will be easier to perform calibration and to recognize output collected data. As shown in figures above, the accelerometer sensitive axis need to be pointed either upward or downward so that we read the earth's gravity vector plus some bias on one axis, and on the others only bias. It is essential that this experiment is performed on a flat surface so that the gravity vector will not affect the other two axes and they should be held parallel to the local horizontal plane.

As shown in figures bellow, each plot represents accelerometer's output and they are different from each other, since the orientation of the sensors are different and that gives different values for each sensor's axis. For that purpose we need to calibrate the sensors and represent them in a unified coordinate for all of them, which in this case is the triad's axes.

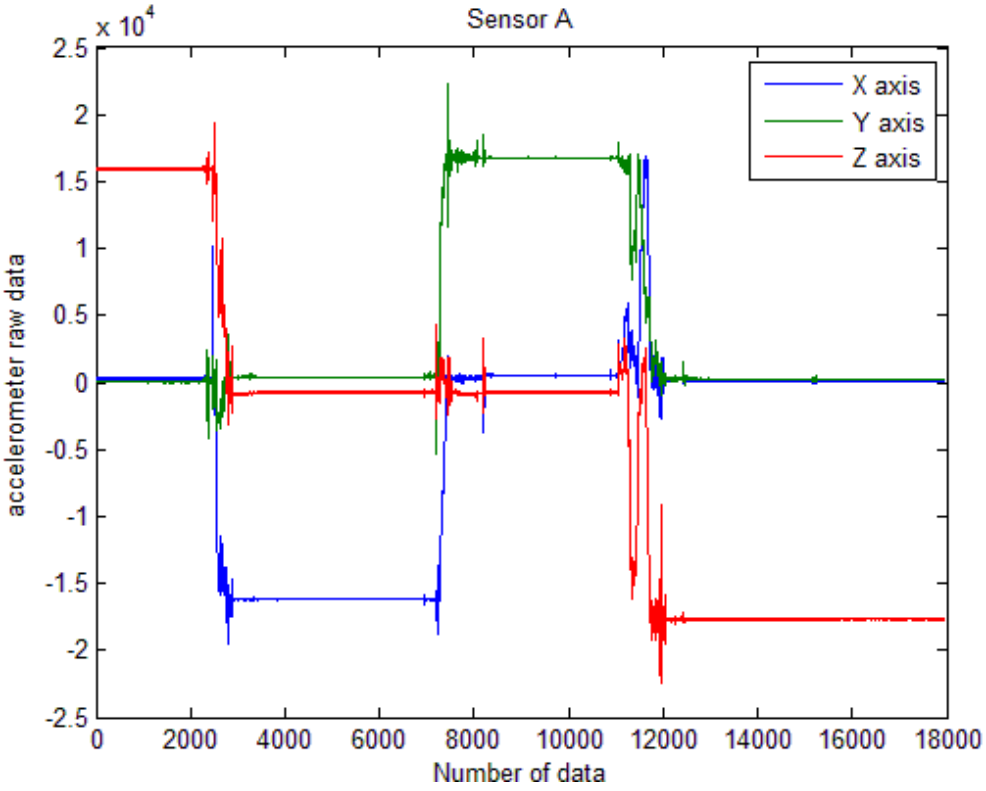

Figure ۲،۱۰: Sensor triad A's output raw data

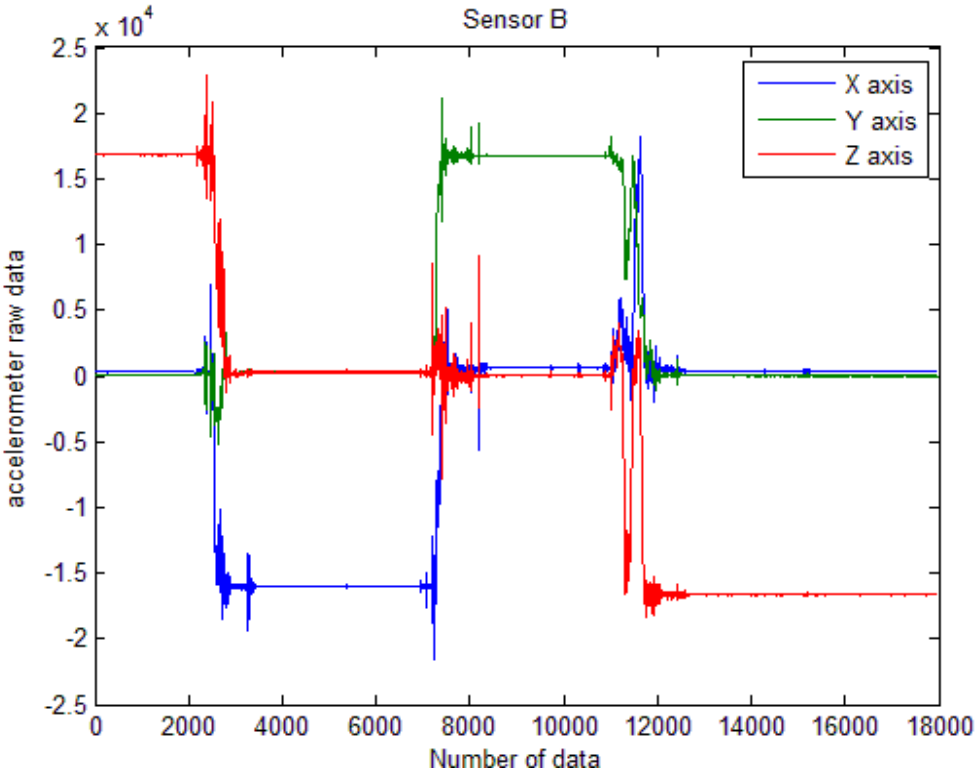

Figure ۲،۱۱: Sensor triad B's output raw

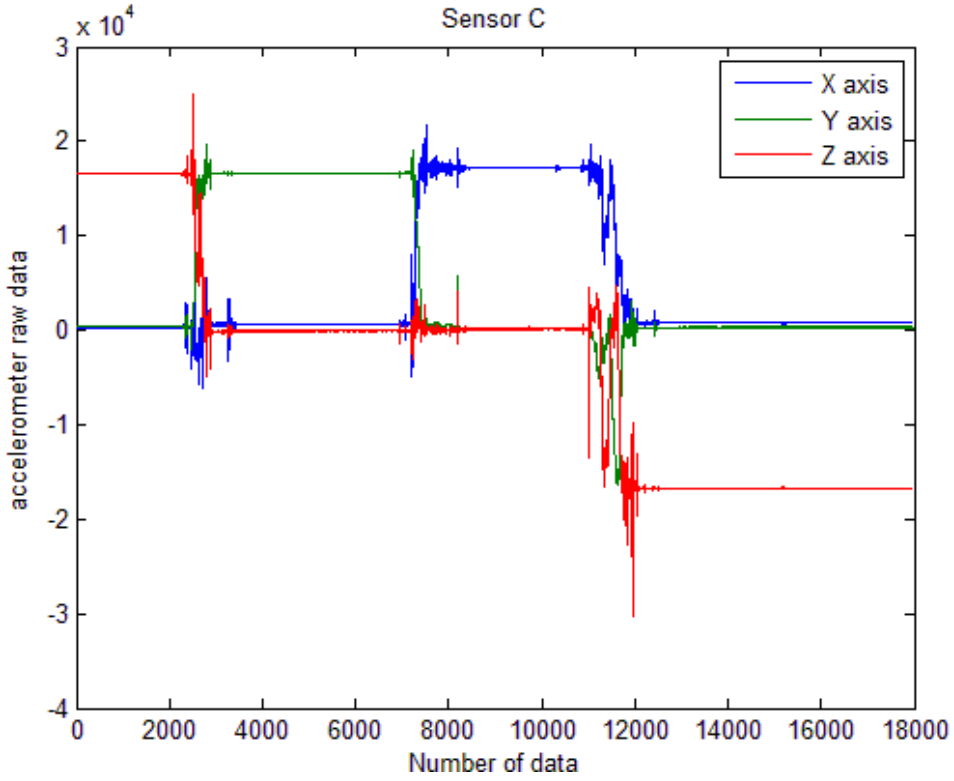

Figure ٢،١٢: Sensor triad C's output raw

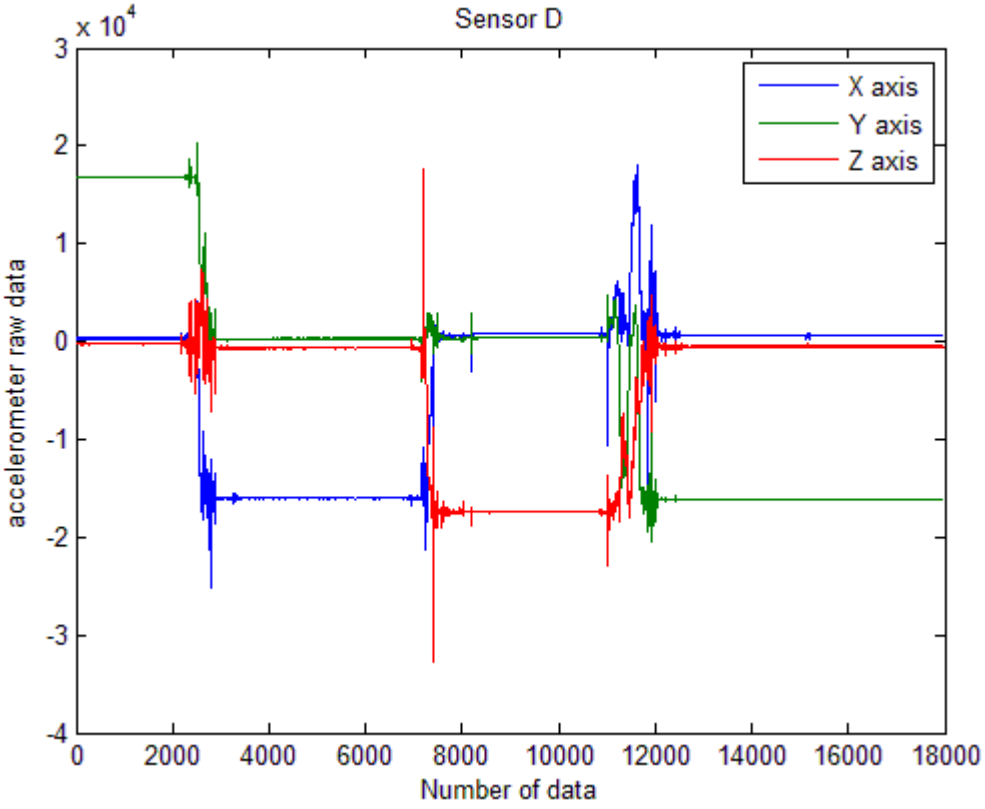

Figure ٢،١٣: Sensor triad D's output

Four measurements were collected and processed in MATLAB program as in [١] and are used to find the misalignment, scale factor, and bias. By this program one can also find the direction cosine matrix to transform from sensor's coordinate frame to the triad's frame.
Shown in table ٢،١ is the summary of the accelerometers calibration, after this calibration, any data coming out from the accelerometers will be in m/s^2.

Sensor triad A:

Misalignment and scale factor			Bias	
-١٣,٣٣	-١٦٧٩,٥	-٣٣,٦٠٩	٤٥١,١٢	x
١٦٨٢,٤	-١٥,٧٠٧	٢٣,٨٧	٢٦٥,٠١	y
-٢٦,٢٢٥	-٢,٥٢٤٢	١٧٠٦,٤	٦٥,٨٠٦	z

Sensor triad B:

Misalignment and scale factor			Bias	
١٦٧٧,٦	-٢٧,٧٣١	-٥,٧١٦٦	٧٢١,٨٦	x
٢٢,١١٩	١٦٦٧,٩	١٩,٦١٣	٢٨٦,١٤	y
٣,٩١٢٤	٤,١٣٤٦	١٦٩٨,٣	-٦١,٩٩	z

Sensor triad C:

Misalignment and scale factor			Bias	
-٧,٤١٥٦	-١٦٩٠,١	-٤٣,٧٣٧	٥٤٥,٩٥	x
-٨,٢٥٥٧	-١٨,٨٧٧	١٦٧٩,٢	٣١٣,٠٦	y
-١٧١٨,٣	٨,٣٠٨٦	-٧,٥٧٨٧	-٤٩١,٨٥	z

Sensor triad D:

Misalignment and scale factor			Bias	
-٦,١٨٣٣	-١٦٨٠,٩	-١,٢١٦٢	٢٧٦,٦٤	x
١٦٧٧,١	-٨,٣٢٦٣	٨,١٧٤٧	٣٣٠,٣٣	y
-٥,٤٨٥	١٣,٤٨٢	١٧١٣,٤	-٨٦٣,٤	z

Table ٢،١: Accelerometer calibration parameters

Colored in green and bolded, are the scale factor of the sensors for each axis, and the fourth column is the bias colored in red and the rest are the misalignments. From the above data, it is obvious that the sensor triads have different alignment.

٢,٣,٢ Gyroscope calibrations

Gyro calibration procedure is similar to the accelerometer calibration, but instead of changing the orientation of the sensor, the gyro is mounted on a precise rotating turntable which is controlled by a computer, and rotates at the precisely known angular rate.

The equations used for gyro calibration is similar to the one used for accelerometers, but instead of a_n in equation ٢.٢ as the earth's gravity to the sensors, we rotate the turn table with a specified turn rate ω_n. The detail of the program in MATLAB is shown in Appendix II.

Shown bellow is the setup used to perform the calibration of the Gyroscope. As we can see, the triad is mounted on a leveled turntable and rotated at ١٠ RPM clockwise and counterclockwise, so we will get six measurements since we have ٣ axes with ٣ rotations.

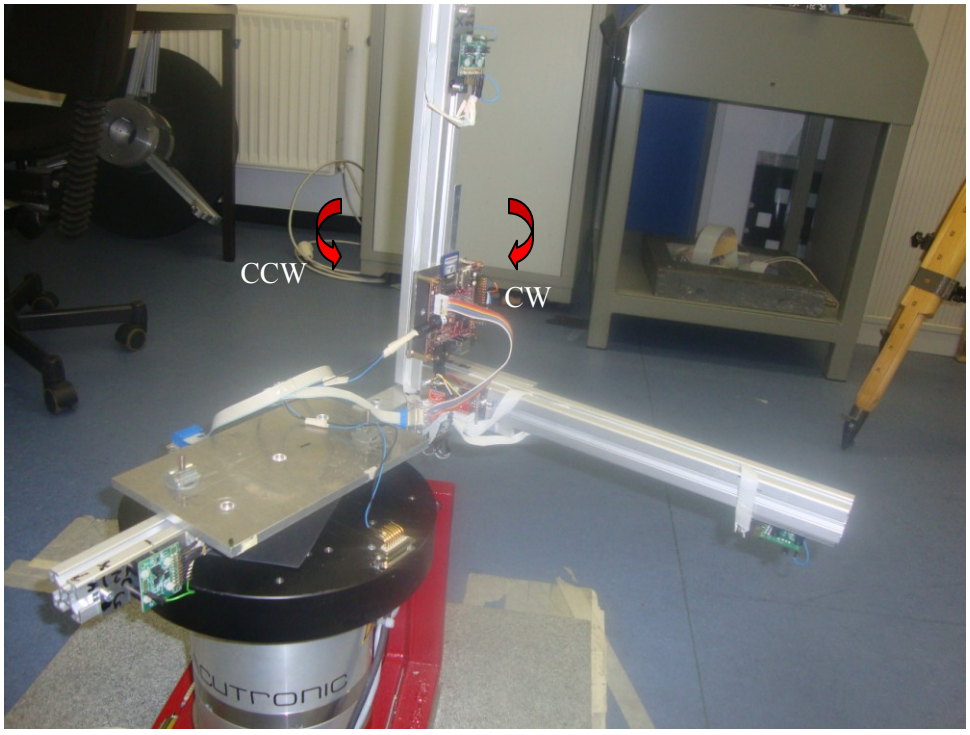

Figure ٢.١٤: rotation around X-axis

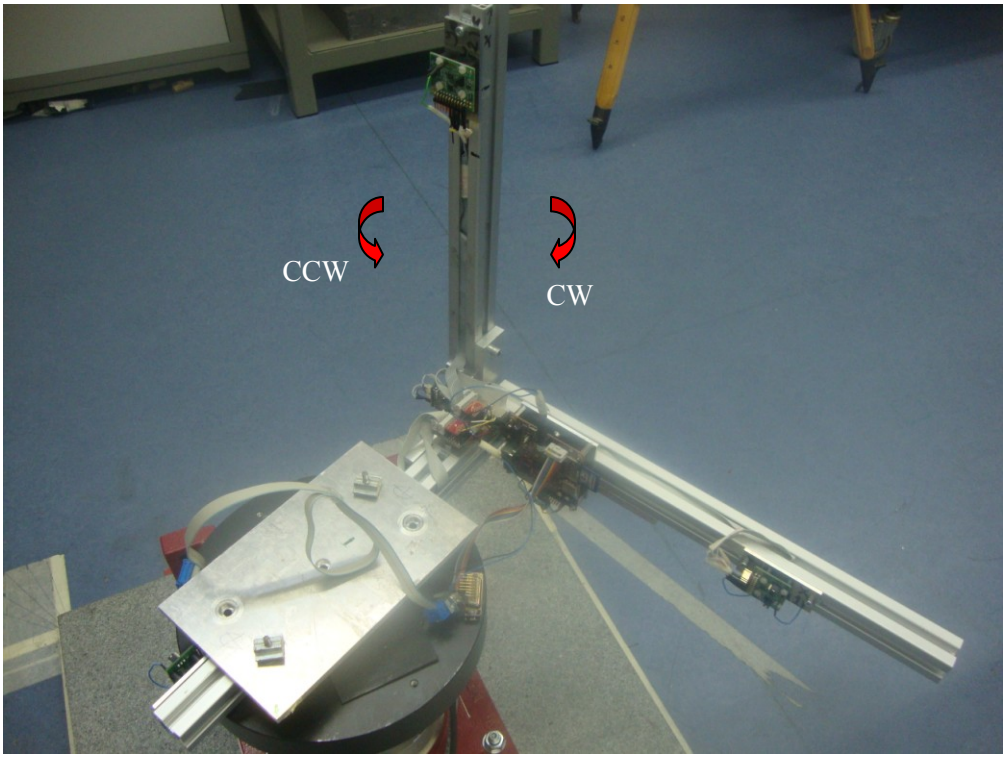

Figure ٢.١٥: rotation around Y-axis

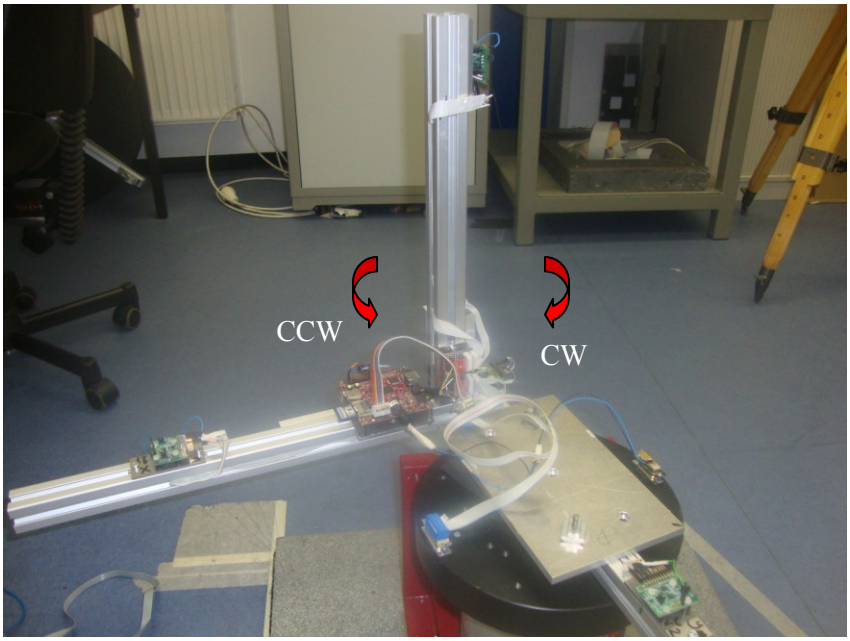

Figure ٢.١٦: rotation around Z-axis

The output of the gyroscope after performing the above rotations is shown in Fig. ٢,١٧

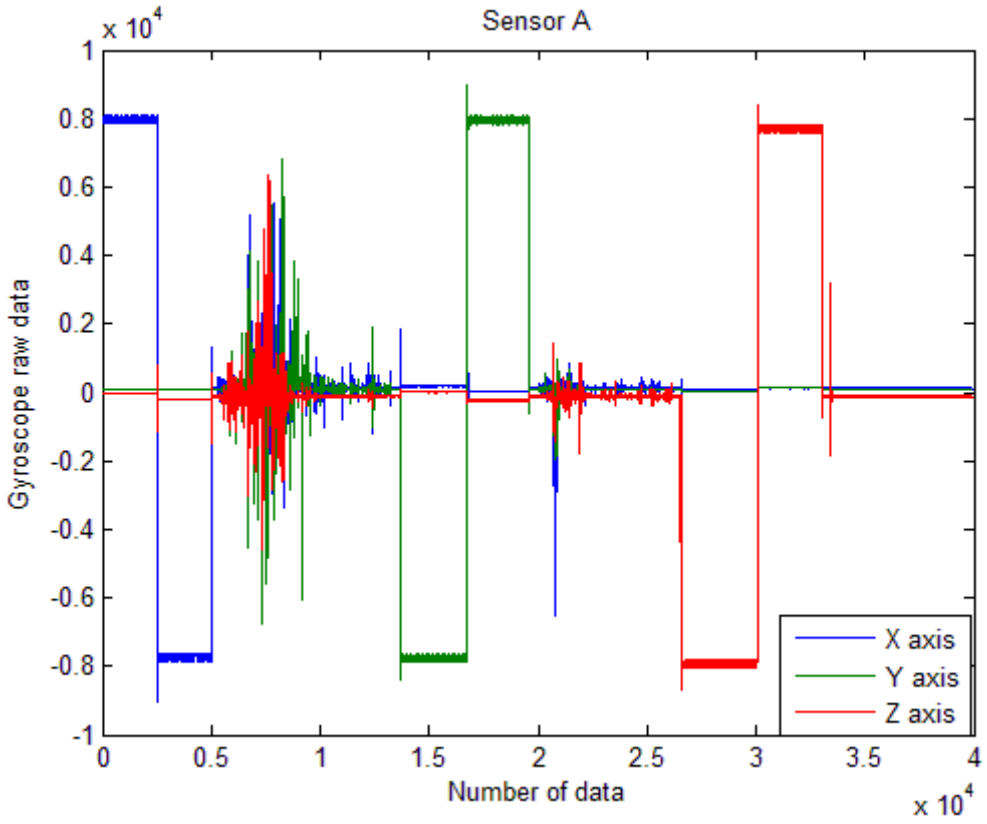

Figure ٢.١٧: Gyro triad A's output

Shown in table ٢,٢ is the summary of the gyro calibration

Misalignment and scale factor			Bias	
٧٠,٣٠٢	٧٥٠٧	-٧٫٥٦٢٦	١١٦,٣٥٢	x
-٧٥١٧,٥	٢,٣	-٥٣,١١٧	٩١,١٥٢	y
١١٤,٧٧	٩٥,١٨٩	-٧٤٧٣,٧	-١١٤,٦٦٣	z

Table ٢,٢: Gyro calibration Parameters

Colored in green and bolded, are the scale factor of the gyroscope for each axis, and the fourth column is the bias colored in red and the rest are the misalignments.

After doing this calibration, any value read from gyroscope will be in rad/s.

٣ Experimental setup

٣,١ Hardware

The hardware components used in this work are a Laptop with Linux Ubuntu ١١,٠٤ operating system for sensor data analysis and BeagleBoard for data logging from IMUs , MPU٦٠٥٠-ev ٩ DOF inertial measurement units from Invensense. Two logic level converters are used since the logic voltage from BeagleBoard side is ١,٨V and from the IMU side is ٣,٣V.

The following sections provide the setup details and the characteristics of BeagleBoard and summarize the IMU features and the connection diagram between them.

٣,١,١ BeagleBoard as single-board computer

The BeagleBoard (Fig.٣,١) is a powerful single-board computer developed by Texas Instruments, featuring their OMAP٣٥٣٠ system on a chip. This OMAP٣٥٣٠ builds in an ARM Cortex-A٨ at ٧٢٠ MHz CPU clock. The board is supported by a large community and is designed with open source development in mind. It measures about ٣x٣" and has all the functionality of a basic computer. It has many expansion options, the BeagleBoard can be used as the backbone for a large variety of projects [٢].

The OMAP٣٥٣٠ includes an HD-video capable TMS٣٢٠C٦٤x+ DSP for accelerated video and audio decoding, and an OpenGL ES ٢,٠ capable ٢D/٣D GPU. Video outputs can be provided by the on-board S-video or DVI-D (HDMI connector) outputs. Several communication issues are solved by the OMAP through I٢C, SPI, UART communication which are available in the expansion header of the board allowing a high flexibility to communicate with a large variety of sensors [٢].

The board also includes an MMC+/SD/SDIO interface, USB ٢,٠, ٣,٥mm stereo audio in/out connectors, RS٢٣٢ and JTAG ports. It can consume up to ٢W of power, which can be provided via USB or an external ٥V source, via the on-board barrel jack. Because of the very low power consumption, the board requires no additional cooling [٢].

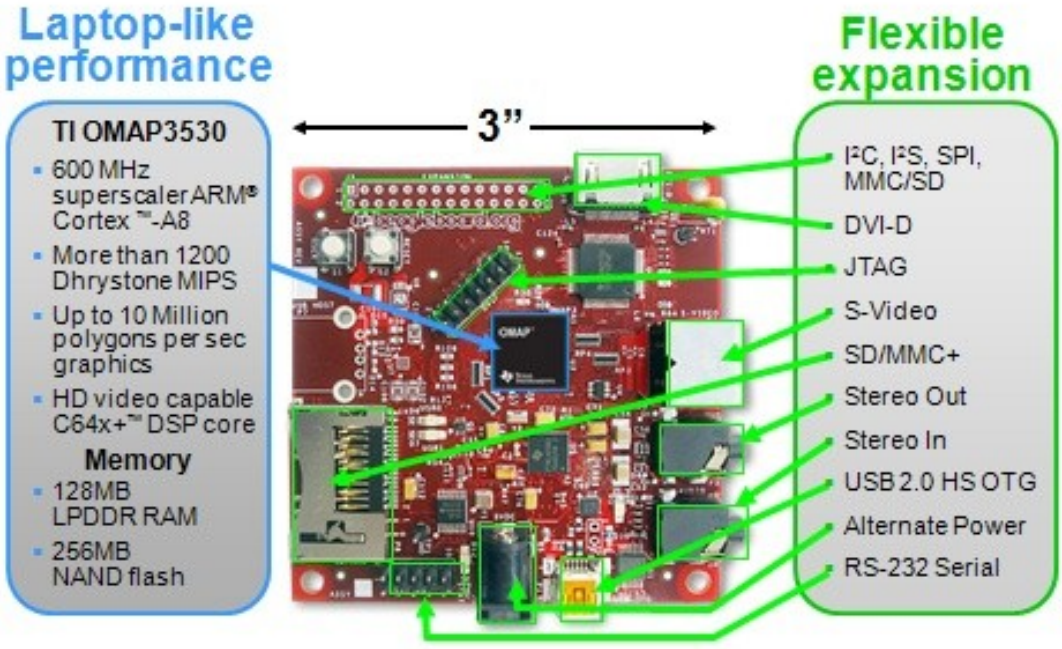

Figure ٣.١: Beagle Board's hardware [after wikipedia.org/wiki/BeagleBoard].

BeagleBoard requires several connections and configurations. The following connections are needed to accomplish the first board's start up.

Power Supply

Powering up the BeagleBoard can be achieved in two ways: the first one is by connecting an USB OTG (on-the-go) cable which will deliver the power to switch on the board. The second way is to connect an external power supply adapter of ٥V to the jack barrel connector which is available for such purpose. The last option is recommended because at the moment to run for the first time, the system requires a bit of extra power and while supplying the board through USB could lead in a problem due to USB OTG cannot deliver enough current for this operation resulting in error or hanging issues when trying to boot up the system [٣].

Serial Connection

Next, it will be shown how to communicate with the BeagleBoard which is accomplished by Ethernet over USB, but the first communication will be realized serially since the Ethernet needs some settings after the board is initialized for the first time.

To connect the board using RS٢٣٢, a serial cable is needed to plug it into the ١٠ pin-header of the board. This cable is known as DB٩-M to IDC-١٠

SERIAL PORT (AT-EVEREX) shown in Fig.٣,٢. In Fig.٣,٣, a null-modem cable can be seen which is needed as well for connecting to the host PC.

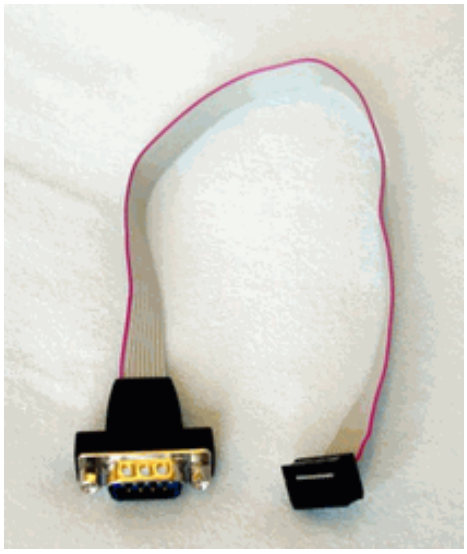

Figure ٣,٢: DB٩-M to IDC-١٠ serial converter
(after www.ocean-server.com)

Figure ٣,٣: Null modem cable (after www.ethersol.com)

To start sending data from the board to the PC, a terminal program is needed to display the data out. For this purpose the well-known software **Putty** is recommended. Fig ٣,٤ and Fig ٣,٥ clarify the settings using serial communication for BeagleBoard in Putty.

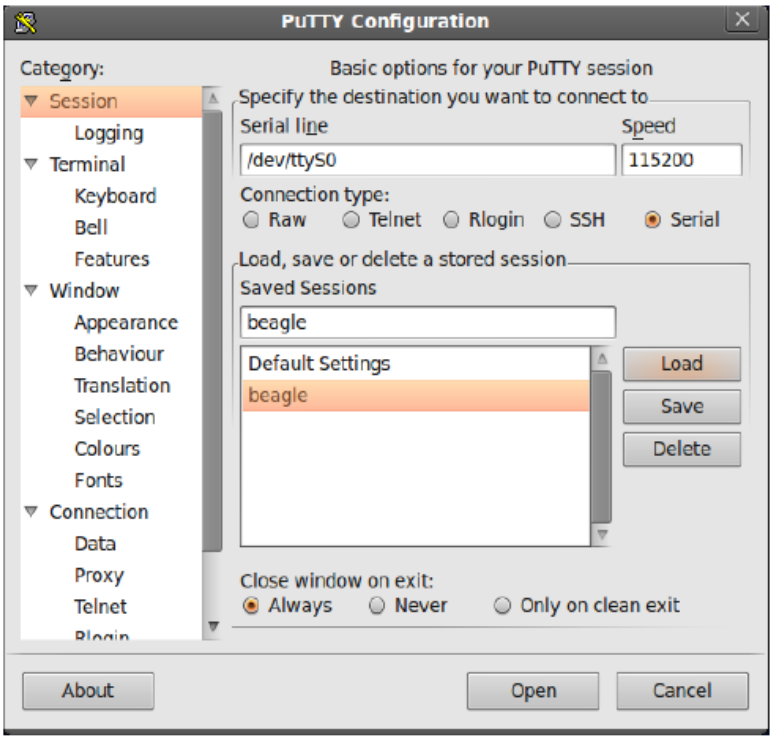

Figure ٣.٤: Saving Putty session

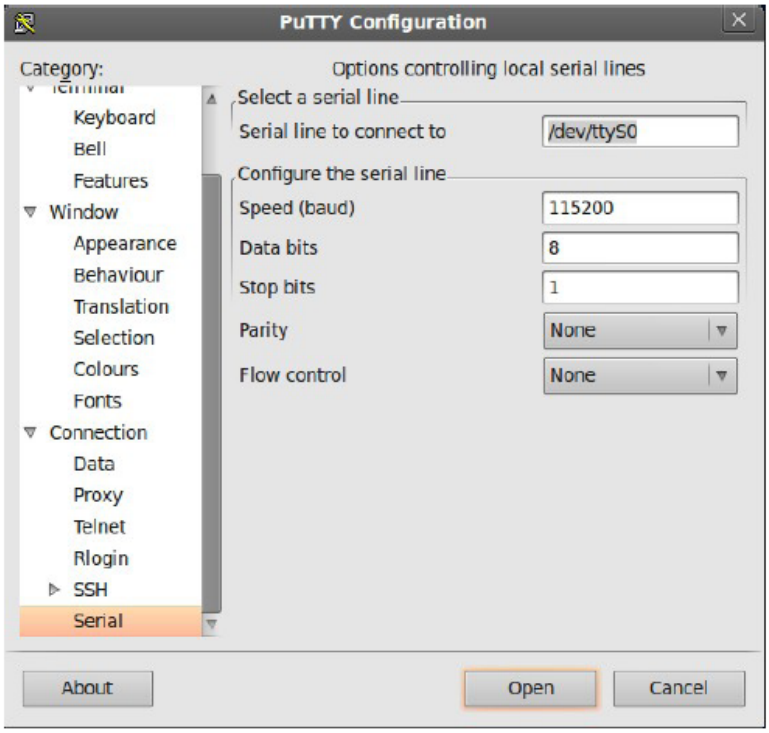

Figure ٣.٥: Serial configuration for BeagleBoard

SD Memory Card Configuration

It is necessary to use a MMC SD memory card to run the operative system. The memory card needs a specific formatting for proper operation. This section shows how to create a dual partition card, booting from a FAT partition that can be read by the OMAP٣ ROM boot loader and Linux/Windows utilizing an EXT٣ partition for Linux root file system.

SD device detection

We plug the SD Card into the SD Card Reader and then plug the SD Card Reader into your system. After doing that, we do the following to determine which device it is on your system. This example uses ٢GB SD card [٣].

dmesg | tail

```
...
[ ٦٨٥٤,٢١٥٦٥٠] sd ٧:٠:٠:٠: [sdc] Mode Sense: ٠b ٠٠ ٠٠ ٠٨
[ ٦٨٥٤,٢١٥٦٥٣] sd ٧:٠:٠:٠: [sdc] Assuming drive cache: write through
[ ٦٨٥٤,٢١٥٦٥٩] sdc: sdc١
[ ٦٨٥٤,٢١٨٠٧٩] sd ٧:٠:٠:٠: [sdc] Attached SCSI removable disk
[ ٦٨٥٤,٢١٨١٣٥] sd ٧:٠:٠:٠: Attached scsi generic sg٢ type ٠
...
```

In this case, it shows up as **/dev/sdc.**

Checking if the automounter has mounted the SD card

Note there may be more than one partition (only one is shown in the example below).

df -h

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sdc١	٤٠٠M	٩٤ M	٣٠٧ M	٢٤٪	/media/disks

Note the "Mounted on" field in the above (in this case /media/disk) and use that name in the umount commands below.

Unmounting the SD card

umount /media/disk

Starting fdisk

We choose the whole device (/dev/sdc), not a single partition (/dev/sdc١).

sudo fdisk /dev/sdc

Print the partition record
In the following steps the commands between [ ] are the user inputs.

Command (m for help): [p]
Disk /dev/sdc: ٢٠٢١ MB, ٢٠٢١٦٥٤٥٢٨ bytes
٢٥٥ heads, ٦٣ sectors/track, ٢٤٥ cylinders
Units = cylinders of ١٦٠٦٥ * ٥١٢ = ٨٢٢٥٢٨٠ bytes
Device Boot Start End Blocks Id System
/dev/sdc١ * ١ ٢٤٦ ١٩٧٤٢٤٠+ c W٩٥ FAT٣٢
(LBA)

Partition ١ has different physical/logical endings:

phys=(٢٤٤, ٢٥٤, ٦٣) logical=(٢٤٥, ٢٠٠, ١٩)

So we know the starting point. Make sure to write down the number of bytes on the card (in this example, ٢٠٢١٦٥٤٥٢٨ bytes).

Deleting existing partitions

We delete any previous partitions.

Command (m for help): [d]
Selected partition ١

We repeat this step in case there are more partitions.

Set the memory card's geometry

If the print out of the partition record does not show ٢٥٥ heads, ٦٣ sectors/track, then do the following additional steps to set the proper geometry of the card.

- Go into expert mode:

  Command (m for help): [x]

- Set the number of heads to ٢٥٥:

  Expert Command (m for help): [h]
  Number of heads (١-٢٥٦, default xxx): [٢٥٥]

- Set the number of sectors to ٦٣:

  Expert Command (m for help): [s]
  Number of sectors (١-٦٣, default xxx): [٦٣]

- Now calculate the number of cylinders of the SD card:

$$Cylinders = \frac{number\ of\ bytes\ on\ the\ SD\ card}{255 * 63 * 512}$$

So for this SD card model, the number of cylinders to use is ٢٤٥ (i.e. truncate, don't round).

$$\frac{2021654528}{255 * 63 * 512} = 245.9\ cylinders$$

- Set the number of cylinders to the number calculated:

  Expert Command (m for help): [c]
  Number of cylinders (١-٢٥٦, default xxx): [٢٤٥]

- Return to normal mode:

  Expert Command (m for help): [r]

Checking the modifications

We print the partition record to check the work done so far.

Command (m for help): [p]
Disk /dev/sdc: ٢٠٢١ MB, ٢٠٢١٦٥٤٥٢٨ bytes
٢٥٥ heads, ٦٣ sectors/track, ٢٤٥ cylinders
Units = cylinders of ١٦٠٦٥ * ٥١٢ = ٨٢٢٥٢٨٠ bytes
Device Boot Start End Blocks Id System

Creating boot partition

We create the FAT٣٢ partition to provide booting for the board.
Command (m for help): [n]
Command action
e extended
p primary partition (١-٤)

[p]
Partition number (١-٤): [١]
First cylinder (١-٢٤٥, default ١): [(press Enter)]
Using default value ١
Last cylinder or +size or +sizeM or +sizeK (١-٢٤٥, default ٢٤٥): [+٥٠]
Command (m for help): [t]
Selected partition ١
Hex code (type L to list codes): [c]
Changed system type of partition ١ to c (W٩٥ FAT٣٢ (LBA))

Important: We have to mark it as bootable.

Command (m for help): [a]
Partition number (١-٤): [١]

Creating Linux partition

Create the ext٣ partition which will contain the root file system.

Command (m for help): [n]
Command action
e extended
p primary partition (١-٤)
[p]
Partition number (١-٤): [٢]
First cylinder (٥٢-٢٤٥, default ٥٢): [(press Enter)]
Using default value ٥٢
Last cylinder or +size or +sizeM or +sizeK (٥٢-٢٤٥, default ٢٤٥): [(press Enter)]
Using default value ٢٤٥

We use the print command to check the work done so far.

Command (m for help): [p]
Disk /dev/sdc: ٢٠٢١ MB, ٢٠٢١٦٥٤٥٢٨ bytes
٢٥٥ heads, ٦٣ sectors/track, ٢٤٥ cylinders
Units = cylinders of ١٦٠٦٥ * ٥١٢ = ٨٢٢٥٢٨٠ bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sdc١	*	١	٥١	٤٠٩٦٢٦	c	W٩٥ FAT٣٢ (LBA)
/dev/sdc٢		٥٢	٢٤٥	١٥٥٨٣٠٥	٨٣	Linux

Saving settings

We save the new partition record on the SD Card. This is an important step because all the work up to now is temporary.

Command (m for help): [w]
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: Re-reading the partition table failed with error ١٦: Device or resource busy.
The kernel still uses the old table.
The new table will be used at the next reboot.
partitions, please see the fdisk manual page for additional information.
Syncing disks.

Formatting partitions

The two partitions are given the volume names **LABEL١** and **LABEL٢** by these commands. These volume labels can be substituted.

> *sudo mkfs.msdos -F ٣٢ /dev/sdc ١ -n LABEL ١*
> mkfs.msdos ٢,١١ (١٢ Mar ٢٠١٢)
> *sudo mkfs.ext٣ -L LABEL ٢ /dev/sdc ٢*

mke٢fs ١,٤٠-WIP (١٤-Mar-٢٠١٢)
Filesystem label=
OS type: Linux
Block size=٤٠٩٦ (log=٢)
Fragment size=٤٠٩٦ (log=٢)
١٩٥٠٧٢ inodes, ٣٨٩٥٧٦ blocks
١٩٤٧٨ blocks (٥,٠٠٪) reserved for the super user
First data block=٠
Maximum filesystem blocks=٤٠٢٦٥٣١٨٤
١٢ block groups
٣٢٧٦٨ blocks per group, ٣٢٧٦٨ fragments per group
١٦٢٥٦ inodes per group
Superblock backups stored on blocks:
٣٢٧٦٨, ٩٨٣٠٤, ١٦٣٨٤٠, ٢٢٩٣٧٦, ٢٩٤٩١٢
Writing inode tables: done
Creating journal (٨١٩٢ blocks): done

Writing superblocks and filesystem accounting information.

Now the SD memory card is configured properly in order to copy the boot files and the file system which will boot up the operative system on the BeagleBoard.

٣,١,٢ MPU-٦٠٥٠ sensors evaluation board

The MPU-٦٠٠٠/MPU-٦٠٥٠™ family of parts are the world's first and only ٦-axis Motion Tracking devices designed for the low power, low cost, and high performance requirements of smartphones, tablets and wearable sensors [٤].

The MPU-٦٠٥٠ incorporates InvenSense's MotionFusion™ and run-time calibration firmware that enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices in motion-enabled products, and guarantees that sensor fusion algorithms and calibration procedures deliver optimal performance for consumers [٤].

Motion interface is rapidly becoming a key function in many consumer electronics devices including smartphones, tablets, gaming consoles, and smart-TVs as it provides an intuitive way for consumers to interact with electronic devices by tracking motion in free space and delivering these motions as input commands.

According to [٤], The MPU-٦٠٠٠/٦٠٥٠ devices combine a ٣-axis gyroscope and a ٣-axis accelerometer on the same silicon die together with an onboard Digital Motion Processor™ (DMP™) capable of processing complex ٩-axis Motion Fusion algorithms. The parts' integrated ٩-axis Motion Fusion algorithms access external magnetometers or other sensors through an auxiliary master I²C bus, allowing the devices to gather a full set of sensor data without intervention from the system processor.

The InvenSense MotionApps™ Platform that comes with the MPU-٦٠٥٠ abstracts motion-based complexities, offloads sensor management from the operating system and provides a structured set of APIs for application development.

For precision tracking of both fast and slow motions, the parts feature a user-programmable gyro full-scale range of ±٢٥٠, ±٥٠٠, ±١٠٠٠, and ±٢٠٠٠°/sec (dps) and a user-programmable accelerometer full-scale range of ±٢g, ±٤g, ±٨g, and ±١٦g [٤].

Table bellow shows the MPU٦٠٥٠ main characteristics

MEMS Gyro MPU٦٠٥٠	MEMS Accelerometer MPU٦٠٥٠
Gyroscope range: ±٢٥٠ °/s, ±٥٠٠ °/s, ±١٠٠٠ °/s, ±٢٠٠٠	Accelerometer range: ±٢g, ±٤ g,±٨g, ±١٦g
Gyroscope ADC resolution: ١٦-bit	Accelerometer ADC resolution: ١٦-bit
Scalable measurement range: ١٣١،٦٥,٥،٣٢,٨،١٦,٤ LSB/(°/s)	Scalable measurement range: ١٦٣٨٤, ٨١٩٢,٤٠٩٦،٢٠٤٨ LSB/g
Total RMS Noise: DLPFCFG=٢ (١٠٠Hz) ٠,٠٥ °/s-rms Rate Noise Spectral Density At ١٠Hz ٠,٠٠٥ °/s/√Hz	Power spectral density @١٠Hz = μ٤٠٠g/√Hz
Initial ZRO Tolerance at ٢٥°C : ±٢٠ °/s	Zero G output : X & Y axis = ±٥٠ mg, Z axis= ±٨٠mg

Table ٣,١: MPU ٦٠٥٠ gyro & accelerometer characteristics [after ٤]

These two sensors along with a digital tri-axis compass and a temperature sensor has been collected together into a single evaluation board shown in Fig.٣,٦.

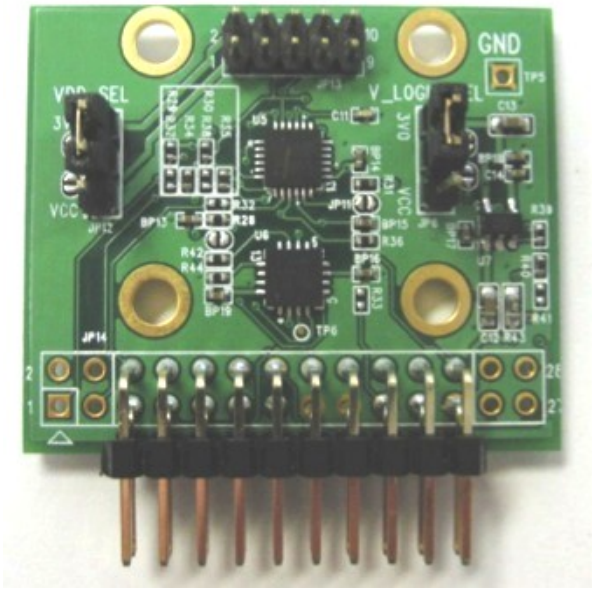

Figure ٣.٦: MPU٦٠٥٠ Evaluation board [after ٥]

This evaluation board can be used by itself using I٢C or SPI (MPU٦٠٠٠ only) or it can be connected to InvenSence 's ARM evaluation board for

connecting to the host computer using USB interface. Shown in Fig.٣.٧ is the system diagram of the evaluation board.

Figure ٣,٧: MPU٦٠٥٠ system diagram (after Invensense webpage)

There are some jumpers on this evaluation board for special purposes, like jumper (J١٤) which is ١٠x٢ pins including power supply, I٢C pins, address pin, external clock, interrupt, and synchronization pin. Jumper (J١٣) on the top of the board is intended for connecting additional devices to EV board, such as camera image stabilization processor, or a digital-output compass, or a GPS.
The three-pin power selection header (JP١٢) is used to select which voltage supply is fed to MPU. The ٣-pin VLOGIV selection header (JP٦) is used to select between ٣V and VDD as a logic supply voltage. When VDD is selected as a logic supply voltage, the input voltage VDD should be between ٢,٥-٣,٣V. But it is recommended to use ٣V (Jumper ٦, ١-٢ short) to have the expected performance as stated in the datasheet. On the I٢C bus, there are open drain pull up resistors connected, so there is no need that the user connect external pull up resistors.

The MPU٦٠٥٠ EVB has the address of ٠x٦٨ when the address pin voltage is zero (pin ٢٤on jumper JP١٤), but when the voltage of this pin is set to VDD (٣V, Jumper ٦, ١-٢ short), the address will be changed to ٠x٦٩. That was very useful in our project, since we have ٤ of this evaluation board, and two I٢C buses on BeagleBoard. So, one can connect two of this evaluation board on the same bus.

MPU٦٠٥٠ setup and data acquisition

The IMU used in this project works with I٢C protocol at a maximum speed of ٤٠٠ kHz. Setup is performed by writing to the various configuration registers. We only needed to write to specified registers which can wake up the device, set the required registers, and then reading data.

The procedure is as follow:

١. Initialize I٢C bus with the aimed sensor (by giving the bus number and IMU address).

٢. Wakeup the sensor by writing zero to the bit number six of the power management register and specify clock source by writing to the last three LSB of the same register.
٣. Enable data-ready interrupt to synchronize the data reading with the BeagleBoard by writing one to the bit number zero of the interrupt enable register.
٤. Specify the sample rate by writing to the register sample rate divider. The sample rate can be specified by dividing the gyroscope output rate by sample rate divider according to the following

Sample rate = Gyroscope output / (١+sample rate divider).

٥. Specify the bandwidth and digital low pass filter by writing to the register configuration
٦. Select the gyroscope and accelerometer full scale range by writing to the registers gyro configuration and accelerometer configuration respectively.
٧. Write to the bit number six of the register user control to enable the FIFO buffer.

Stated above are the initialization and setup of the MPU٦٠٥٠ EVB, the reader is advised to read the MPU٦٠٥٠ register map and description for more detail of each register function and the other register settings.

٣,١,٣ Circuit diagram

Shown in Fig.٣.٨ is the circuit which has been made to connect the MPU٦٠٥٠ EVB to the BeagleBoard through I٢C.

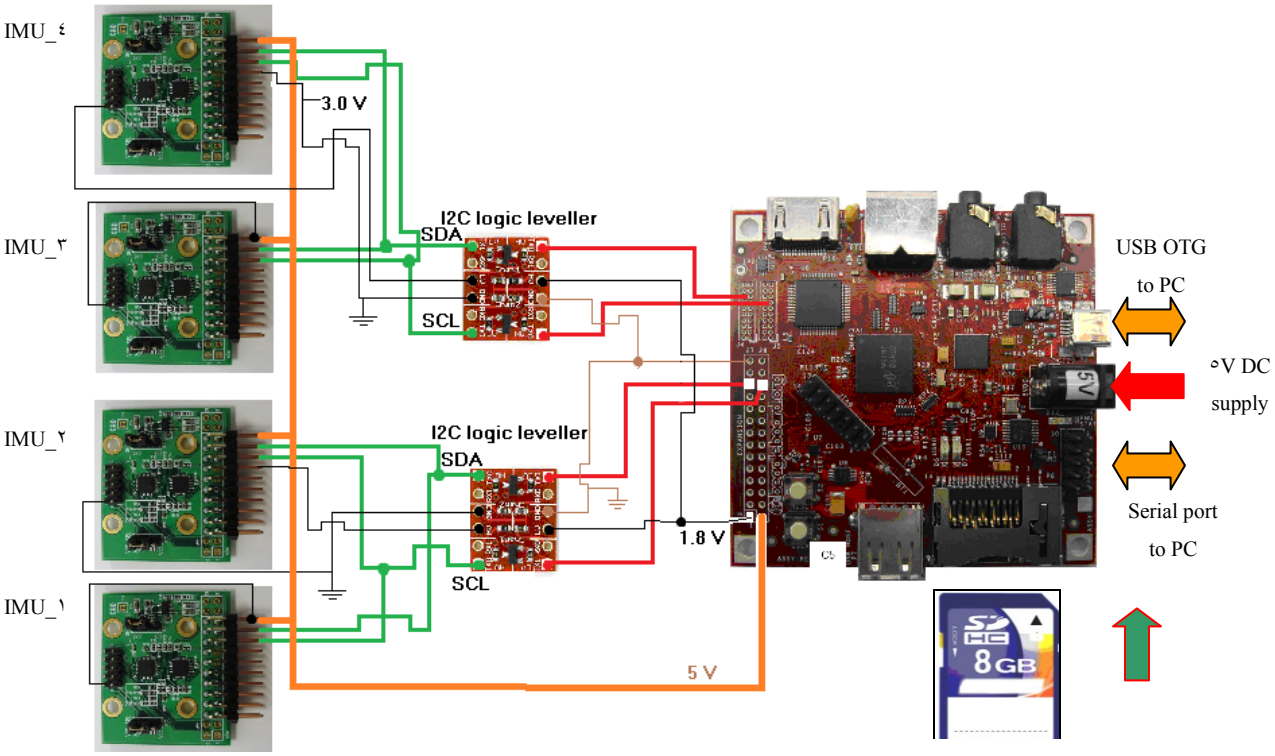

Figure ٣,٨: Circuit diagram

Because of the voltage different in logic lines between BeagleBoard and IMU, the logic level converter has been used with ٢-line data transfer in both directions and ٢-line data transfer in on direction. This logic level converter can transfer data up to ٤٠٠ kHz, which is the maximum frequency of our IMU.

IMU_١ and IMU_٢ are connected on the I٢C bus ٢ and their addresses are ٠x٦٩ and ٠x٦٨ respectively.IMU_٣ and IMU_٤ are connected on the I٢C bus ٣, with the addresses ٠x٦٩ and ٠x٦٨ respectively. The sensor side of the logic leveler is the high voltage (٣,٣V), while the right side is the low voltage (١,٨V).

٣,٢ Software environment

For the sake of reading and transferring data from /to hardware, there are some steps that have to be made on both Laptop and BeagleBoard. For the Laptop, Ubuntu ١١,٠٤ Linux operating system has been installed alongside Windows ٧. The installation steps were straightforward as stated in the Ubuntu website. While for the BeagleBoard, it was somehow difficult to set it up for the required hardware configuration.

٣,٢,١ Operating system

Ångström, which is open-source, free software, Linux-based operating system has been installed on the BeagleBoard, which gives the minimum requirement (Console) for interfacing to the sensors from one side, and to connect to the Laptop from the other side. In the following section the details of how to install Ångström and the other requirements to setup the BeagleBoard will be clarified. Later on, the requirements of how to send and receive data between BeagleBoard and the Laptop will be stated.

- Getting booting files

For first time booting of the BeagleBoard, the pre-built binaries and source code can be found in the following link:

http://www.angstrom-distribution.org/demo/beagleboard/

The following binary files are needed:
• MLO
• u-boot.bin
• uImage
• Angstrom-Beagleboard-demo-image-glibc-ipk-
٢٠١١٫١beagleboard.rootfs.tar.bz٢

The ROM code on the BeagleBoard loads the MLO from MMC card for MMC booting. This booting is automatically established when a SD card is inserted on the board. The MLO binary then loads uboot.bin which is in charge to load the kernel image.
The MLO allows the first boot-loader called X-loader starts up. The u-boot.bin loads the second bootloader which is needed to start basic communication with the hardware and it will provide a basic environment based on a shell. In the u-boot prompt some variables can be set (kernel boot arguments), such as serial communication arguments, which kernel image should be loaded, place where the operative system will be loaded from (in our case from the external SD card), and many other options. The binary file uImage will pass the kernel image which is the kernel for our Linux system. The kernel is the responsible for the correct interaction with the hardware of the board and it's very important that this file is compiled properly in order to boot the Linux system successfully. In following chapters it will be described how to build our own uImage to get a customized kernel for our specific requirements.

- Copying the binary files into the SD card

We insert the SD card in the PC memory card reader. The three binary files described above have to be saved in the boot partition (FAT٣٢) of the SD card by typing in the Laptop's Linux shell the following:

cp MLO /media/LABEL ١
cp u-boot.bin /media/LABEL ١
cp uImage /media/LABEL ١

The commands order is very important. The files can be directly copied using a graphical file explorer instead typing the commands above. Finally the fourth file downloaded is the root file system which will be stored and uncompressed in the second partition of the SD card (EXT٣).

- starting up Ångström

Finally, the SD memory card is ready. Once Ångström is loaded, the prompt of Linux-Angström should be displayed. Now, the board is ready to start installing the necessary tools for developing the first application. To login in the system, we type **root** and press enter. Fig.٣،٩ is the typical layout of the Ångström's operating system at login step.

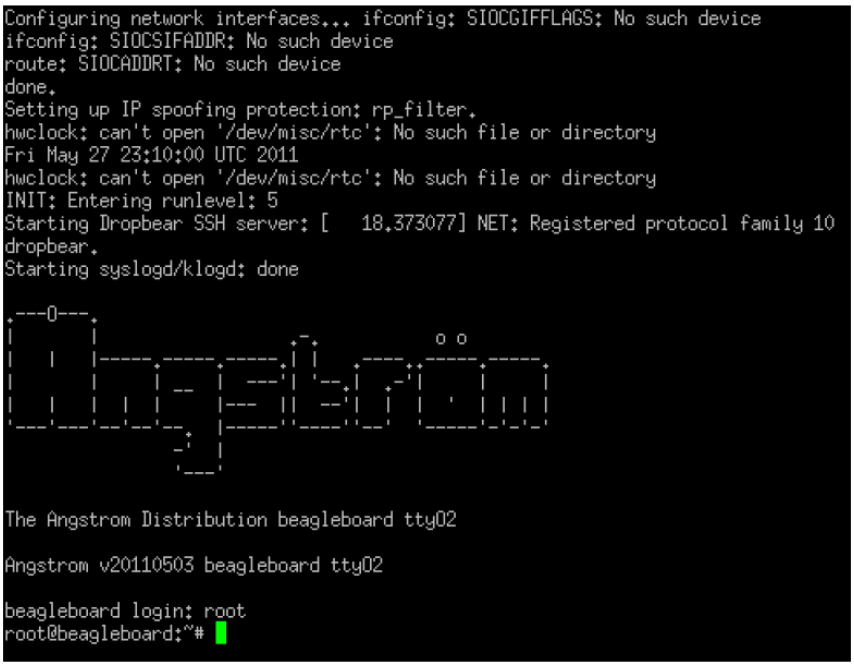

Figure ٣،٩: Ångström's terminal

- Connecting BeagleBoard with Laptop through USB

It is required to connect BeagleBoard to the host computer through USB for easy file transfer between them, and also to make internet available in the BeagleBoard. In this scenario, the host computer will serve as a router for the Ångström device and it is required to configure accordingly. The detailed step by step procedure of this task is stated in [٣].

- Building applications for BeagleBoard

We need to install a cross-compiler in the Laptop which will let us compile the programs for the BeagleBoard architecture (ARM processor). When the

program is compiled, we will transfer the binary file to the BeagleBoard through USB using **scp** file transfer protocol. When we try to compile the executable file out of the compilation in the Laptop, it will results in error since the compiled file is build for ARM architecture. it is recommended to read [٣] where it is clearly stated how to build applications for BeagleBoard. There is also another way (user friendly) to compile debug and programs for the BeagleBoard using Eclipse platform which is highly recommended if the user writes long and complicated codes and is stated step by step for BeagleBone, which is the same as BeagleBoard, in the webpage of Dr. Derek Molloy.

- BeagleBoard I^2C setup

In this project we are required to have two I^2C buses so that we will be able to connect the four sensors on them. BeagleBoard has three I^2C buses, in which two of them are pinned out and can be used by the user. However, by default bus ٢ is disabled due to a lack of pull-up resistors on the board, so external pull-ups to ١,٨V must be added and the kernel recompiled to enable I^2C-٢. The detail of how to enabling it and setting the bus frequency of both I^2C s to maximum of ٤٠٠ kHz is clearly stated in [٣], so it is not described in here particularly due to the size of the thesis .

٣,٢,٢ Real time issues

Unlike classical microcontrollers, the BeagleBoard processor requires an operating system for proper operation, so a strong knowledge of how they work is required. For real time tasks where the execution time is critical, the operating system workload is a big issue. Background processes create latencies; however, they are essential for proper operation of the whole system. since the operating system that has been installed on BeagleBoard is Linux-based, by default it has no build-in real time operating system(RTOS) coming with it. According to [٣], if the real time operating system is installed properly, the latency can be reduced from milliseconds to microseconds, and the accuracy of the data reading is increased. The working principle of a proposed RTOS used in [٣], which is Xenomai and I-pipe, add a extra layer between the hardware and the Linux kernel, to manage real-time tasks separately. As shown in the Fig.٣,١٠. The real-time operating system has not been used in this thesis due to the time limitation, since it requires kernel reconfiguration and recompilation.

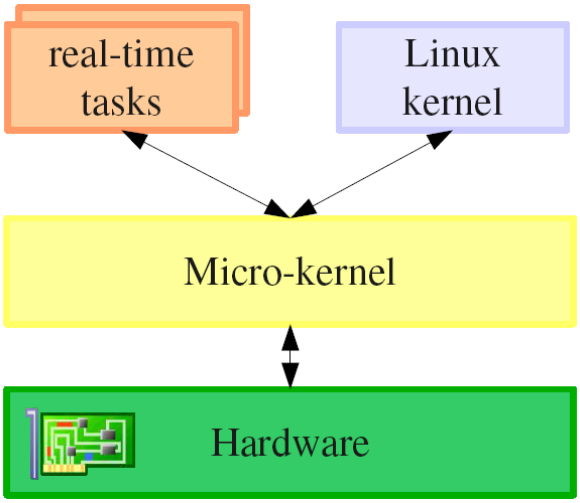

Figure ٣,١٠: RTOS principle

٤ Experiments and results

٤,١ Heading calculation by Maytagging technique

The true north can be found if the earth rotation is measured properly by the gyro. The earth rotation rate according to World Geodic System ١٩٨٤ (WGS٨٤) is Ω_e=٧٢٩٢١١٥ x ١٠$^{-١١}$ rad/s [٦]. The lowest input range of the MPU٦٠٥٠ is ±٢٥٠°/s while the magnitude of the signal needs to be detected is approximately ٠,٠٠٤ °/s. Under this condition, it can be seen that detecting such a week signal requires precise error analysis.

The proposed method for error characterization is the Allan variance that is already described in section ٢,٢.١. The type of error that rapidly changes in a short period of time in the gyro's output signal is called bias instability. Bias instability measurement describes how the dynamic bias of a device may change over a specified period of time, typically around ١٠٠ seconds, in fixed conditions (usually including constant temperature) [٧]. Bias instability is usually specified as a ١σ value with units ◦/h, or ◦/s for less accurate devices. Under the random walk model bias instability can be interpreted as follows; If Bt is the known bias at time t, then a ١σ bias stability of ٠,٠١◦/h over ١٠٠ seconds means that the bias at time (t + ١٠٠) seconds is a random variable with expected value Bt and standard deviation ٠,٠١◦/h [٧].

For capturing the bias instability and other errors in our gyroscope, data from all MPU٦٠٥٠ modules has been logged for about ٦٤ hours at a sample rate of ٥٠ Hz and at room temperature and the output for one of them is shown in Fig.٤,١. There were ٤ units of this module; each unit has ٣ axis gyroscope. In our work, only x and y axis are used for finding north direction.

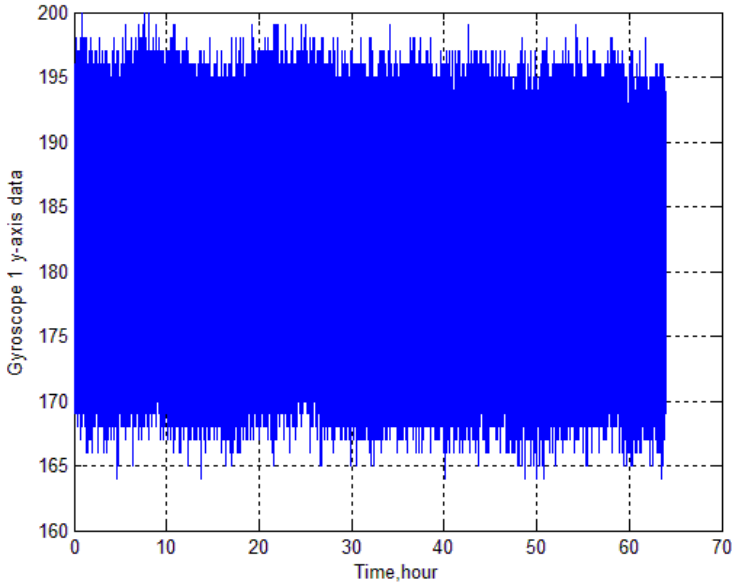

Figure ٤,١: Raw data as a function of time

Bellow is the Allan variance graph for each gyroscope generated after the MATLAB code created by [١٧].

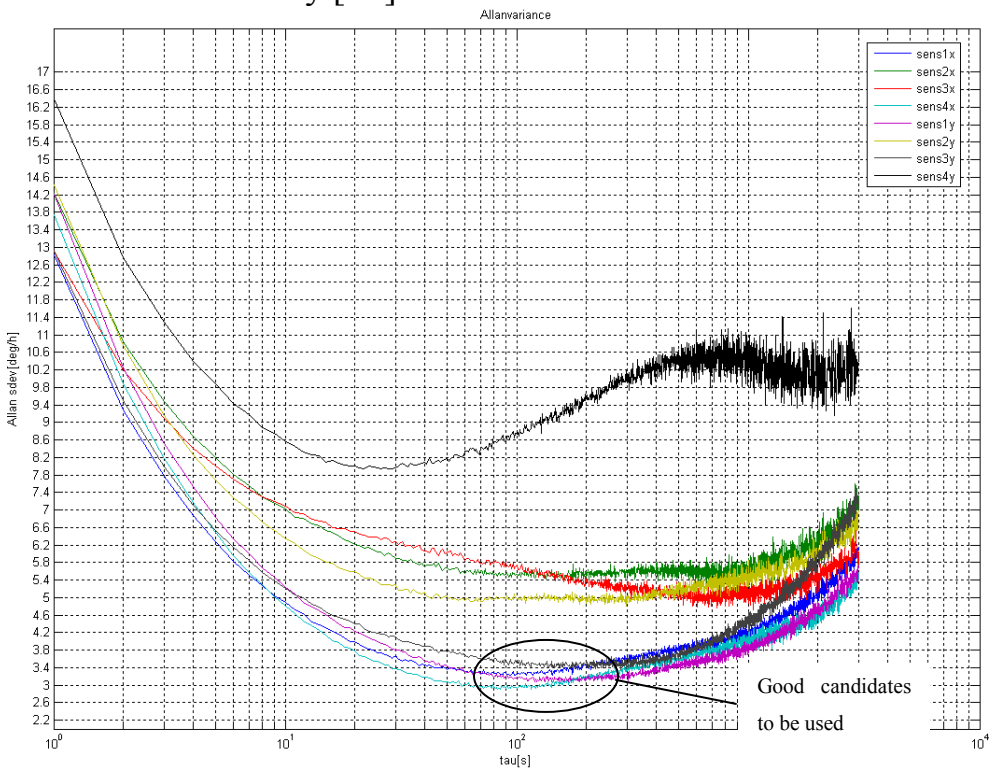

Figure ٤,٢: Allan deviation

From Figure ٤,٢, it is obvious that the bias stability is different for each sensor and also for each axis. For example the x axis of fourth sensor is about ٣ °/h while for its y axis it is about ٨ °/h.

Based on this plot, the x and y axis of first sensor and x axis of the fourth sensor and y axis of the third sensor are used in this project since the include the least bias instability and they are almost at the same averaging time τ.

Measuring the Earth's rotation rate

In order to measure the magnitude of the Earth's rotation rate, the external factors that affect the output data of the gyroscope sensor must be carefully compensated for. One of these factors is the gravitational force which has a large effect over the gyroscope data. To compensate for this factor, the sensitive axis of the gyro can be aligned parallel with the local horizontal plane as shown bellow [٦].

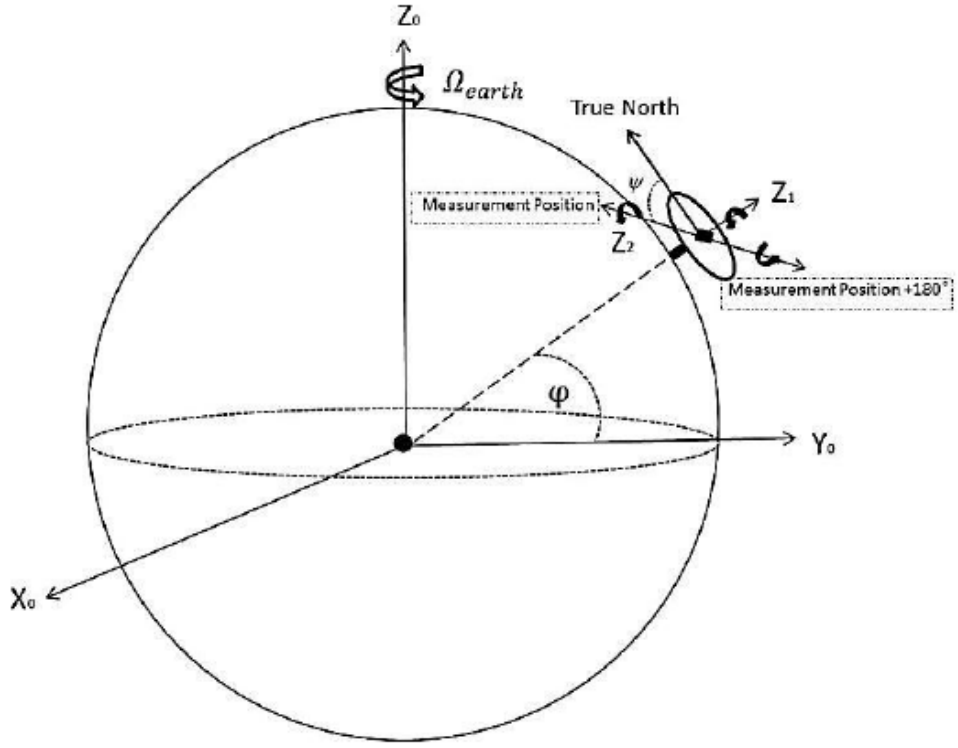

Figure ٤,٣: GPS/INS coordinate system [after ٨]

By using this method, the gravity effect can be eliminated because its value will be the same in all measurement positions. Also to compensate for gyroscope static bias, the turntable is used and rotated ١٨٠ degree.

The mathematical model for the gyroscope reading at each position is [٦]

$$\omega_{position} = \omega_{earth} \cos\varphi \cos\psi + b + b_g \quad (٤,١)$$

$$\omega_{position+١٨٠^{o}} = - \omega_{earth} \cos\varphi \cos\psi + b + b_g \quad (٤,٢)$$

Where b is gyro bias, φ is the latitude, and b_g is g-sensitivity which is unknown but constant value. Ψ represents the angle between the sensing axis and the north direction. By subtracting equation ٤,١ from ٤,٢ we get [٦]

$$\Omega_e \cos\varphi \cos\psi = \frac{\omega_{position} - \omega_{position+180^{o}}}{2} \quad (٤,٣)$$

So, if the latitude and earth rate are known, the only unknown would be the angle ψ.
The experiments were conducted in Siegen, Germany at latitude ٥٠,٨° N. the sampling rate of the sensor was ٥٠ Hz, and the MATLAB R٢٠٠٨a is used for data processing and plotting.

The sensor axes were aligned with the north direction as shown in Fig.٤,٤, and then the data were logged for about ١٠٠ second according to Allan variance plot. Then the setup was rotated ١٨٠° to the opposite position. We took ١٨ position, starting from ٠ (north) for y axis and ٩٠ (East) for x axis and incrementing ١٠ ° clockwise each time.

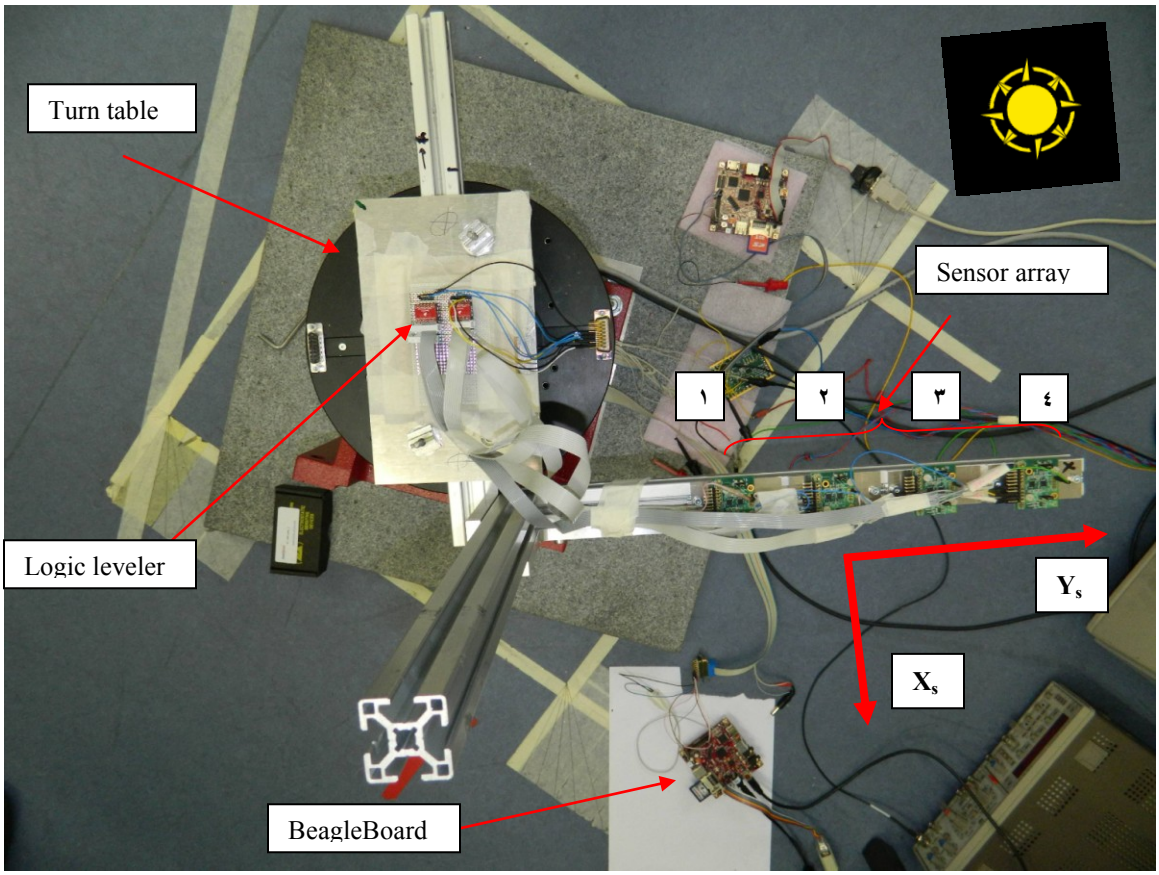

Figure ٤,٤: sensor's axes alignment

According to the setup shown above, the first measurement at zero position (y-axis is pointing to the north and x-axis is pointing to the east) for y-axis should be maximum and for x-axis should be almost zero after we remove the bias from measurements (by rotating ١٨٠ degree).

After the measurements were completed, the files have been uploaded to MATLAB program for analysis and plotting.

Here, we first tried to check whether the sensor can detect the earth rotation or not. To do so, we gave the value of angle ψ to be zero when the y-axis of the sensors are pointing to the north direction and then increasing it ١٠° for each new position. so, according to equation ٤,٣ the angles φ and ψ are known, and $\omega_{position}$ is the sensor reading at zero position, and $\omega_{position+١٨٠^{\circ}}$ is also the sensor reading at opposite position then the only unknown will be the earth's rotation Ω_e.
Fig.٤,٥ shows how the angle ψ is assigned and how the rotations are performed

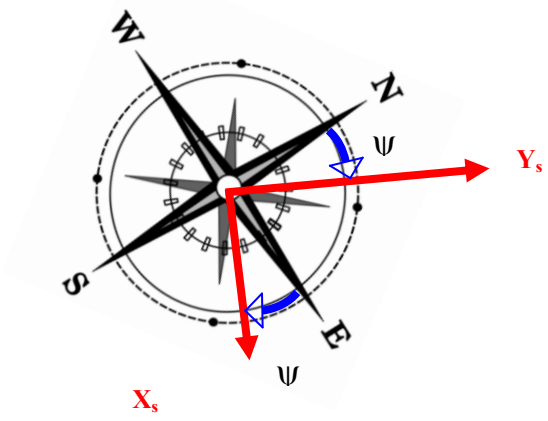

Figure ٤,٥: Angle ψ and sensor's rotation direction

The earth rotation value detected by the sensor's y-axis is shown bellow.

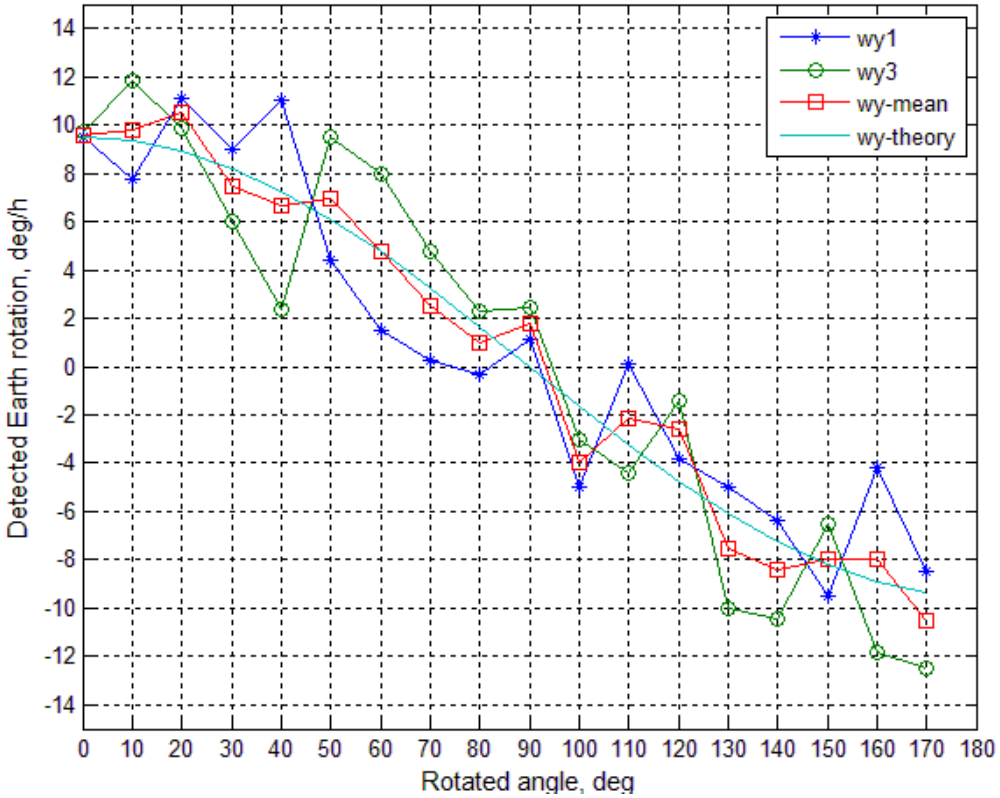

Figure ٤,٦: measured Earth rotation with sensor's y-axis.

In the Fig.٤,٦, the blue line is the theoretical value of the earth rotation that is supposed to be sensed by the sensors. It is the calculated by multiplying the earth rotation with the cosine of the rotated angle ψ, in which angle ψ starts from ٠° to ١٨٠°. The y-axis of first sensor, plotted in green line, and the y-axis of the third sensor, plotted in red line, are fluctuating around the theoretical value of earth rotation.

The error in the earth rotation detection can be reduced by taking the mean value of both sensors as it can be seen in cyan color, which is more close to the theoretical value of earth rotation.

The same measurement for sensor x-axis is shown in the figure bellow.

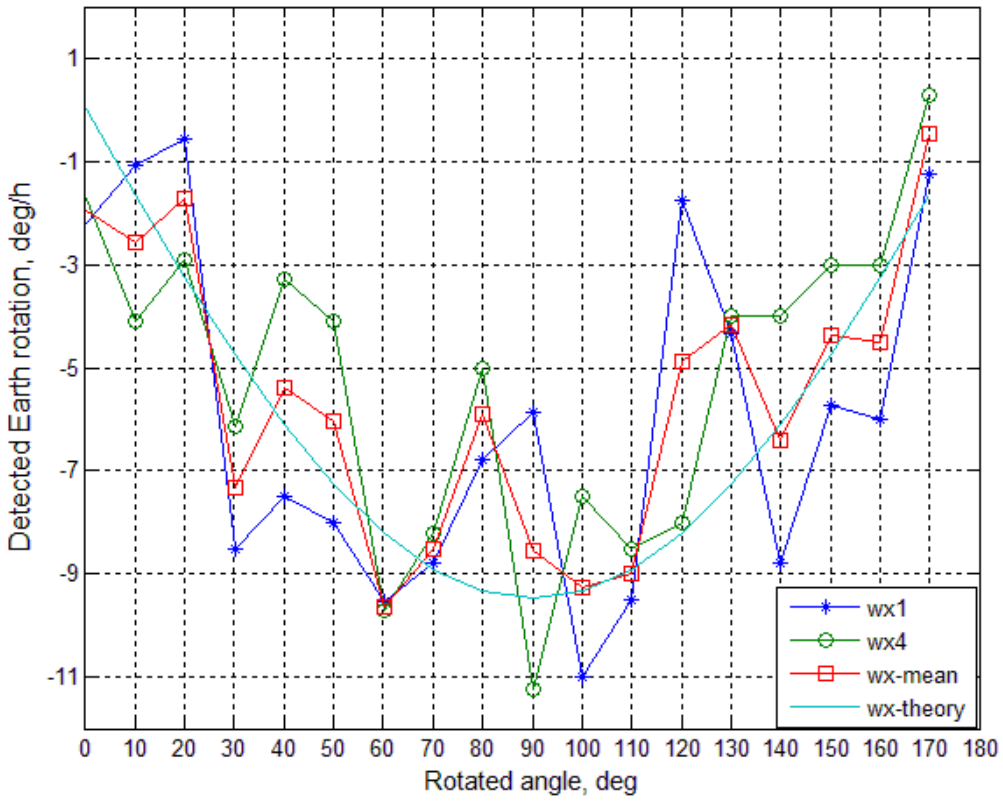

Figure ٤,٧: measured Earth rotation with sensor's x-axis.

As the angle ψ increases, the angular rotation sensed by the sensors increased until they reach their maximum value at angle ١٨٠° then they start to decrease again as the angle gets larger. Here is also the mean value of the sensors output give a better result which is closer to the theoretical value.

After it has been found that the sensor is able to detect the earth rotation, the next step is to measure the north direction angle ψ.

Equation ٤ can be rearranged and expressed in terms of angle ψ [٦]

$$\psi = \cos^{-1}\left(\frac{\omega_{position} - \omega_{position+180^{\circ}}}{2 * \cos\phi * \Omega_e}\right) \qquad (٤,٥)$$

Based on this equation, the following figure has been generated, which represents the detected angle with north direction using y-axis of the sensors.

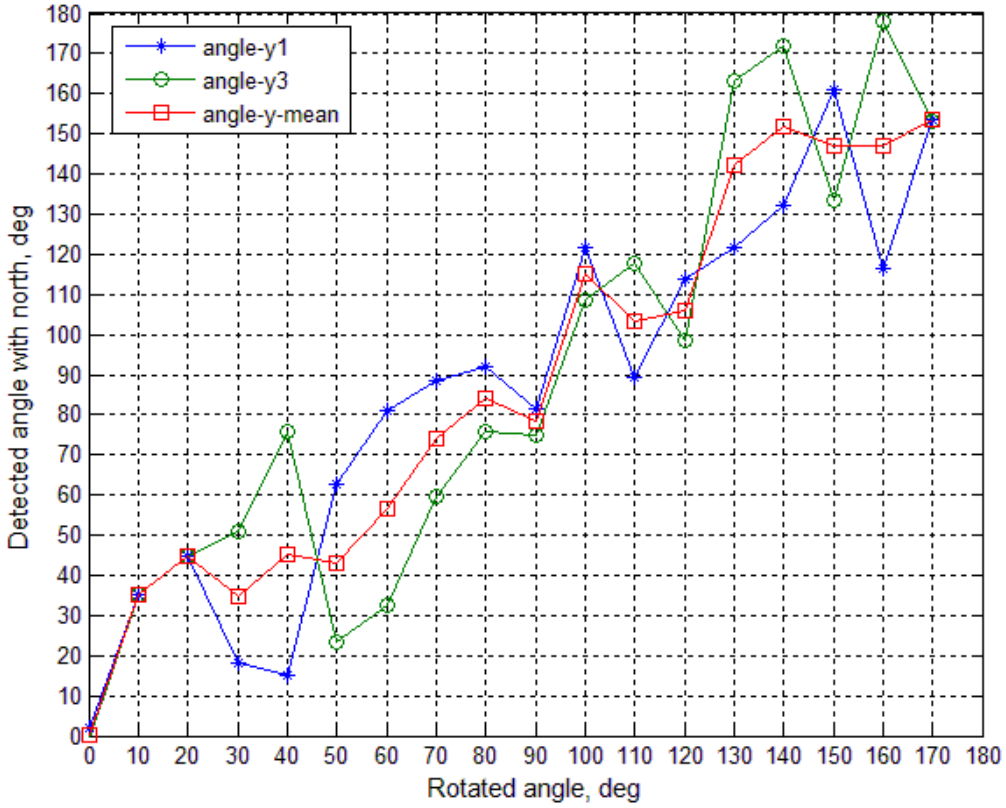

Figure ٤,٨: measured angle ψ using y-axis of sensors

The maximum value of the Earth's rotation in Siegen is about ٩.٤٨ deg/h, but as it can be seen from Fig.٤,٦, at the beginning and the end of the graph the sensors reading exceed this value because of added noise, and if we substitute them in equation ٤,٥, we will have undefined number for angle ψ. to solve this problem, we need to normalize these exceeded values to the range bellow the Earth's rotation.

Here it should be mentioned that the north finding system using this method has ١٨٠ º ambiguities, since we are using cosine function. For example when the result of (ω_{pos} -$\omega_{pos+١٨٠.}$) in equation ٤,٥ is ٠,٥, the angle ψ would be ٦٠ º, but when it is -٠,٥, the angle ψ would be ١٢٠º. This can be solved by checking the sign of x and y axis of the sensors. For example, when the value of gyro y-axis is positive and the x-axis is negative, then we can tell which angle we are measuring with the north direction. Or it can be done,

e.g., by rotating ٣٦٠° at a certain spacing and fitting a sine curve on the measurements [٦].

Figure below shows the sign of the detected earth rotation, and it matches the result obtained shown in figures ٤,٦ and ٤,٧.

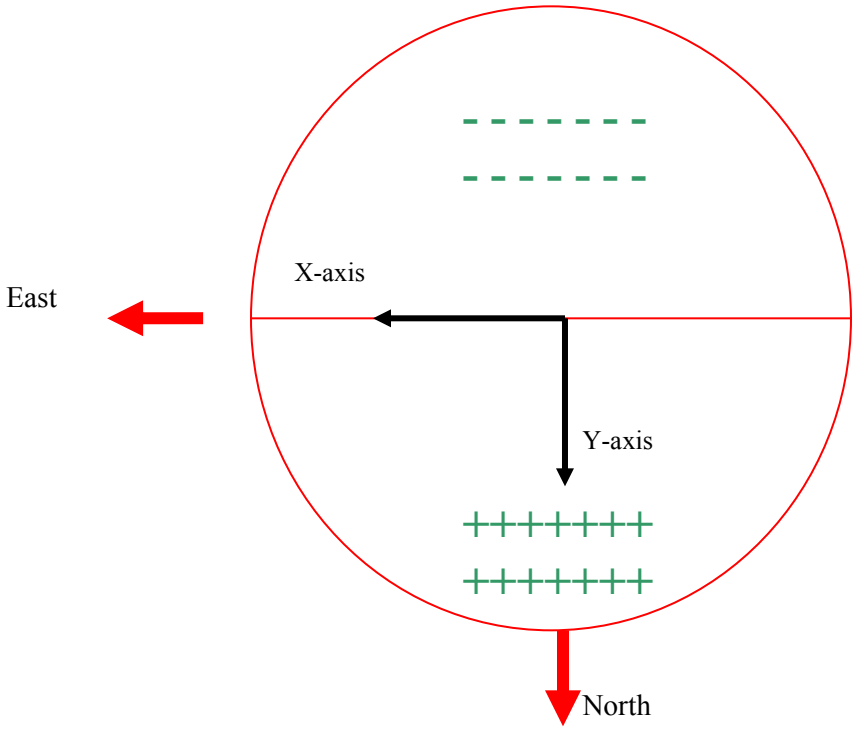

Figure ٤,٩: Gyroscope output sign ambiguity

٤.٢ Gyro-bias compensation using distributed accelerometer triad

In this method, the bias in the gyroscope is detected by means of a distributed accelerometer triad. If the gyroscope bias is detected, the gyroscope can be used to detect the Earth rotation.

As it is stated in section ٤,١, the gyroscope is used to find the Earth rotation by logging data at first position for a specified time, then rotating the setup with ١٨٠ degree to the second position. The purpose of this rotation was to remove the bias from the gyroscope. Here we are going to detect this bias and subtract it from gyro bias.

The idea behind this method is the accelerometers is used to calculate rotational speed using extended Kalman filtering, and then the output of

gyroscope is compared with the rotational speed calculated from accelerometer triad to detect the gyroscope bias.

Kalman filtering

The Kalman filter is a tool that can estimate the variables of a wide range of processes. In mathematical terms we would say that a Kalman filter estimates the states of a linear system. The Kalman filter not only works well in practice, but it is theoretically attractive because it can be shown that of all possible filters, it is the one that minimizes the variance of the estimation error. Kalman filters are often implemented in embedded control systems because in order to control a process, you first need an accurate estimate of the process variables [١٠]. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.

Figure bellow shows the simple working principle of the Kalman filter.

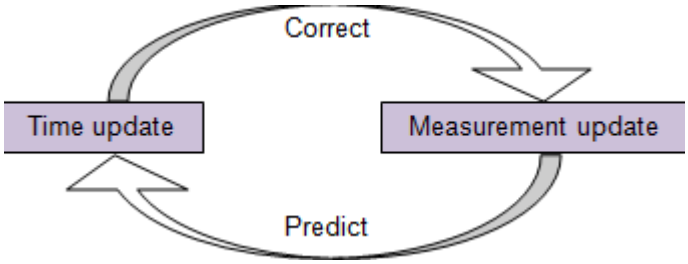

Figure ٤,١٠: Update time and measurement in Kalman filter

In Kalman filter, a process is estimated by using feedback control. It computes the process state at some time and then obtains feedback in the form of (noisy) measurements. The equations fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The time update projects the current state estimate ahead in time while the measurement update adjusts the time estimate.

The Kalman filter removes noise by assuming a pre-defined model of a system. Therefore, the Kalman filter model must be meaningful. It should be defined as follows [١١]:

١. **Understand the situation:** Look at the problem. Break it down to the mathematical basics. If you don't do this, you may end up doing unneeded work.
٢. **Model the state process:** Start with a basic model. It may not work effectively at first, but this can be refined later.
٣. **Model the measurement process:** Analyze how you are going to measure the process. The measurement space may not be in the same space as the state (e.g., using an electrical diode to measure weight, an electrical reading does not easily translate to a weight).
٤. **Model the noise:** This needs to be done for both the **state** and **measurement** process. The base Kalman filter assumes Gaussian (white) noise, so make the variance and covariance (error) meaningful (i.e., make sure that the error you model is suitable for the situation).
٥. **Test the filter:** Often overlooked, use synthetic data if necessary (e.g., if the process is not safe to test on a live environment). See if the filter is behaving as it should.
٦. **Refine filter:** Try to change the noise parameters (filter), as this is the easiest to change. If necessary go back further, you may need to rethink the situation.

Multiple distributed accelerometer triad

We focus on configurations consisting of twelve mono-axial accelerometers. Without loss of generality; we consider the set of four tri-axial accelerometers shown in Fig.٤,١٢ as an example configuration that follows the rules for extracting the angular information vector (AIV) without inhibiting singularity of the coefficient matrix [١].

Mainly we consider this configuration because a minimum of twelve accelerometers are needed to determine the magnitude of the angular velocity and its direction (algebraic sign cannot be determined uniquely). The greatest amount of angular motion information, which is in the nine angular terms that we show next, can be extracted from this configuration. Moreover, this configuration has a low geometric dilution of precision (GDOP) factor for both angular and translational acceleration without the

central accelerometer triad. Finally yet importantly, it is the most practical configuration because IMUs exist in triads of gyros and accelerometers [١].

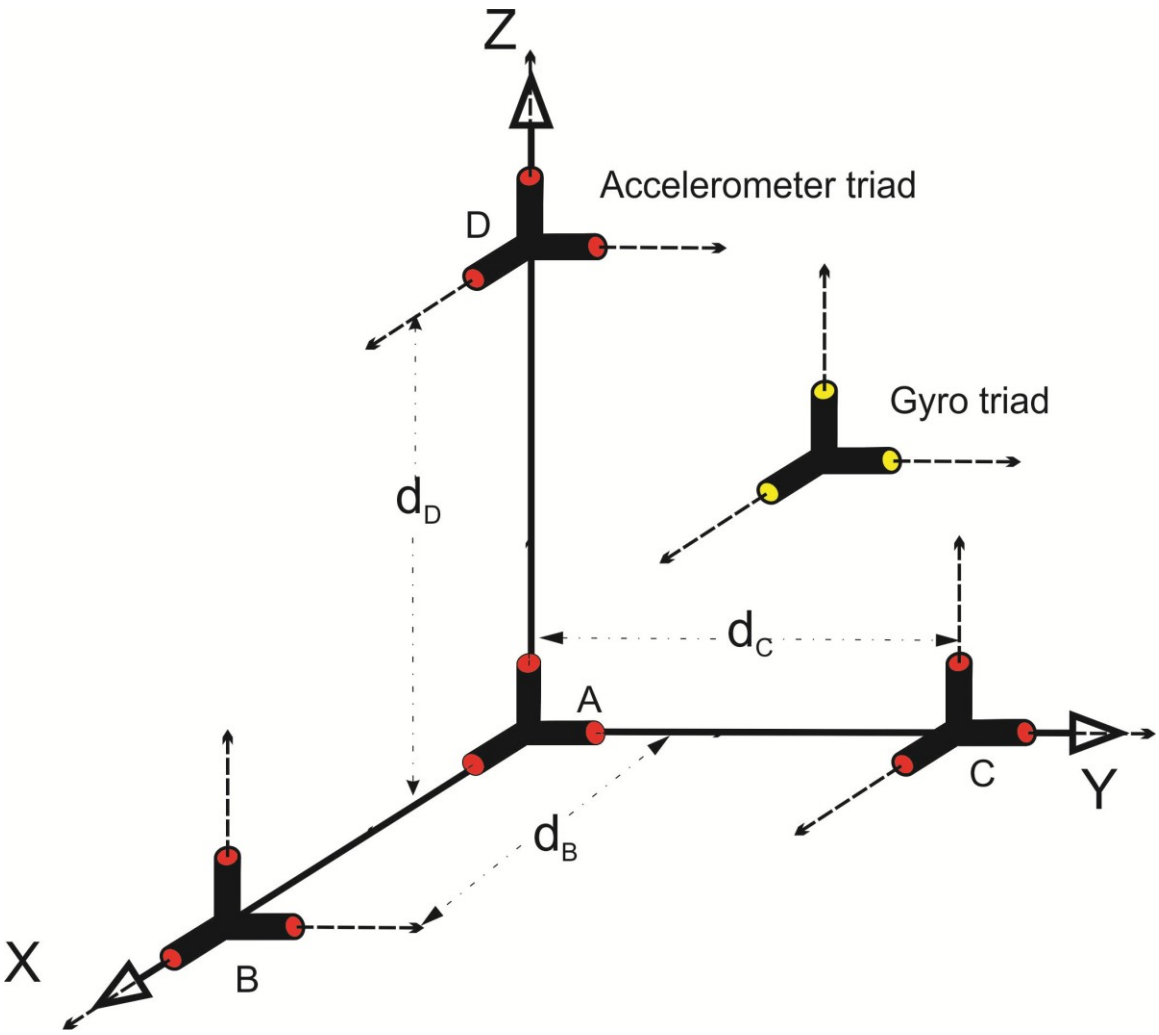

Figure ٤،١٢: Configuration of multiple distributed accelerometers

For the case of uniform distribution distance d (which is ٣٣ cm), The AIV is [١]:

$$\dot{\omega}_x = \left(a_z^C - a_z^A - a_y^D + a_y^A\right)/2d$$

$$\dot{\omega}_y = \left(a_x^D - a_x^A - a_z^B + a_z^A\right)/2d \qquad (٤.٦)$$

$$\dot{\omega}_z = \left(a_y^B - a_y^A - a_x^C + a_x^A\right)/2d$$

$$\omega_x \omega_y = \left(a_y^B - a_y^A + a_x^C - a_x^A\right)/2d$$

$$\omega_x \omega_z = \left(a_z^B - a_z^A + a_x^D - a_x^A\right)/2d \qquad (٤.٧)$$

$$\omega_y \omega_z = \left(a_z^C - a_z^A + a_y^D - a_y^A\right)/2d$$

$$\omega_x^2 = \left(a_x^B - a_x^A - a_y^C + a_y^A - a_z^D + a_z^A\right)/2d$$

$$\omega_y^2 = \left(a_y^C - a_y^A - a_x^B + a_x^A - a_z^D + a_z^A\right)/2d \qquad (٤.٨)$$

$$\omega_z^2 = \left(a_z^D - a_z^A - a_x^B + a_x^A - a_y^C + a_y^A\right)/2d$$

All previous equations are linear combinations of accelerometers measurements. The notation a_{axis}^{triad} refers to the distributed accelerometer measurement with the superscript referring to the triad location and a subscript referring to the axis index.

The calibration of the triads are performed after that as stated in section ٢,٣,١ so that the sensors axes are unified to the triad axis and also to find misalignment, scale factor, and bias.

The three-State model EKF

Based on the previously derived AIV (٤,٦, ٤,٧, and ٤.٨), we can formulate the EKF setup. The quadratic terms do not give a unique angular velocity vector solution. Instead we get two solutions. In our setup, we consider only a fixed accelerometers configuration. For the determination of the algebraic sign in a completely GF-IMU, the gyros have been used to insure a correct sign convergence in the GF-IMU. We are interested in estimating the angular velocity component along each body axis in ٣D.
In reality, the continuous angular velocity vector is replaced with the angular change vector because the computerized implementation is discrete. The angular change is the sampled angular velocity multiplied by the sampling time and is given as [١]

$$\underline{x} = [x_1\, x_2\, x_3]^T = [\sigma_x\, \sigma_y\, \sigma_z]^T \qquad (٤.٩)$$

١. Initialization

The initial state vector can be set as

$$\hat{\underline{x}}_0^+ = E\{\underline{x}_0\} = [\sigma_{x_0}\, \sigma_{y_0}\, \sigma_{z_0}]^T \qquad (٤,١٠)$$

The initial estimation error covariance is given as

$$P_0^+ = E\{(\underline{x}_0 - \hat{\underline{x}}_0^+)(\underline{x}_0 - \hat{\underline{x}}_0^+)^T\} \qquad (٤,١١)$$

٢. Prediction

In discrete time, the actual output of each accelerometer is the velocity change; hence the output of (٤.٦) is the angular velocity change vector α . The process model based on Euler integration is

$$\begin{aligned} \sigma_{x_k} &= \sigma_{x_{k-1}} + \alpha_{x_{k-1}} \Delta t + w_{x_{k-1}} \\ \sigma_{y_k} &= \sigma_{y_{k-1}} + \alpha_{y_{k-1}} \Delta t + w_{y_{k-1}} \\ \sigma_{z_k} &= \sigma_{z_{k-1}} + \alpha_{z_{k-1}} \Delta t + w_{z_{k-1}} \end{aligned} \qquad (٤,١٢)$$

We then define the process input as the following

$$\underline{u} = \underline{\alpha}\, \Delta t = [\alpha_x \Delta t \;\; \alpha_y \Delta t \;\; \alpha_z \Delta t]^T \qquad (٤,١٣)$$

Using (٤.١٣), the process given in (٤.١٢) has a linear form of

$$\begin{aligned} \underline{x}_k &= F_{k-1}\underline{x}_{k-1} + G_{k-1}\underline{u}_{k-1} + \underline{w}_{k-1} \\ F_{k-1} &= G_{k-1} = l_{3x3} \end{aligned} \qquad (٤,١٤)$$

We assume that the uncertainty in the process is mainly due to the uncertainty in the angular velocity change. Here, we consider the error of each accelerometer as white Gaussian noise for simplicity. For an accelerometer error accounting for the remaining bias, we developed a solution utilizing the dynamic models to estimate the bias parameters in the nine angular information terms. In that solution, the bias parameters and the angular acceleration vector are augmented within the state-space model to form a ١٥-state model. For that process update, we used the Wiener process or simply the nearly constant acceleration model.

When using such a model, all the bias parameters in the AIV become observable under the condition that the angular acceleration has a non-zero magnitude.

The three-state model has the advantage of simple calculations because only three states need to be estimated. Moreover, there is no need to make an assumption about the dynamics of the motion, and hence such a solution fits most scenarios. Each accelerometer discrete time measurement is composed of true value plus a white noise component

$$a_{meas} = a_{true} + w_{acc} / \sqrt{\Delta t} \qquad (٤,١٥)$$

The white noise w_{acc} has the unit of g /Hz, where g is the gravity or its equivalent derivatives. The discrete white noise depends on the square root of the sampling time $\sqrt{\Delta t}$. The measured velocity change of each accelerometer is expressed as

$$\Delta v_{meas} = \Delta v_{true} + w_{acc} \sqrt{\Delta t} \qquad (٤,١٦)$$

The variance R_a of each accelerometer measurement of velocity change is

$$R_a = E\{(\Delta v_{meas} - \Delta v_{true})^2\} = E\{w_{acc}^2\}\Delta t \qquad (٤,١٧)$$

All accelerometers are modeled with a common upper bound of the noise variance, as they would be in reality. The error corrupting the angular acceleration vector given in (٤.٦) is inherited from the accelerometers' errors, as it is a linear combination of accelerometers. This combination results in a correlated process noise, and its covariance is computed as

$$Q_{k-1} = (\Delta t)^2 \left(\frac{R_a}{d^2}\right) \begin{bmatrix} 1 & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & 1 \end{bmatrix} \qquad (٤.١٨)$$

The predicted or a priori estimation error covariance is updated as

$$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + Q_{k-1} \qquad (٤.١٩)$$

The predicted or priori state estimate is updated as

$$\underline{\hat{x}}_k^- = F_{k-1} \underline{\hat{x}}_{k-1}^+ + G_{k-1} \underline{u}_{k-1} \qquad (٤,٢٠)$$

٣. Measurement update

We plug the measured velocity changes of the accelerometers in the six quadratic terms in to (٤.٧) and (٤.٨) and multiply the resulting sum by the sampling time to derive the measurement of the state vector. Considering the existence of white Gaussian noise in each accelerometer measurement, the observation inherits also a white Gaussian noise $\underline{v}$ vector

$$\underline{y_k} = \underline{h}_k(\underline{x},\underline{v}) = [x_1x_2 \;\; x_1x_2 \;\; x_2x_3 \;\; x_1^2 \;\; x_2^2 \;\; x_3^2]_k^T + [\underline{v_1}.....\underline{v_6}]_k^T \qquad (٤.٢١)$$

The Jacobian of the measurement vector is computed as

$$H = \frac{\partial \underline{h}_k(\underline{x},\underline{v})}{\partial \underline{x}}\Big|_{\underline{x}=\underline{\hat{x}_k^-}} = \begin{bmatrix} x_2 & x_1 & 0 \\ x_3 & 0 & x_1 \\ 0 & x_3 & x_2 \\ 2x_1 & 0 & 0 \\ 0 & 2x_2 & 0 \\ 0 & 0 & 2x_3 \end{bmatrix}_{\underline{x}=\underline{\hat{x}_k^-}} \qquad (٤.٢٢)$$

The measurement is corrected, and its covariance is computed as

$$R_k = (\Delta t)^2 \left(\frac{R_a}{d^2}\right) \begin{bmatrix} 1 & \frac{1}{4} & \frac{1}{4} & 0 & 0 & -\frac{1}{2} \\ \frac{1}{4} & 1 & \frac{1}{4} & 0 & -\frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{4} & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 0 & -\frac{1}{2} & 0 & -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} & \frac{3}{2} \end{bmatrix} \qquad (٤.٢٣)$$

The process noise is correlated with the measurement noise, and its cross-covariance is computed as

$$E\{\underline{w}_k \underline{v}_j^T\} = M_k \delta_{k-j+1}$$

$$M_k = (\Delta t)^2 \left(\frac{R_a}{d^2}\right) \times \begin{bmatrix} -\frac{1}{4} & \frac{1}{4} & 0 & 0 & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & 0 & -\frac{1}{4} & \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & -\frac{1}{4} & \frac{1}{4} & -\frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \qquad (٤.٢٤)$$

The Kalman gain is updated since the cross-covariance has been considered to have better performance.

$$K_k = (P_k^- H_k^T + M_k)(H_k P_k^- H_k^T + H_k M_k + M_k^T H_k^T + R_k)^{-1} \qquad (٤.٢٥)$$

The corrected or posteriori estimation error covariance is updated as

$$P_k^+ = P_k^- - K_k(H_k P_k^- + M_k^T) \qquad (٤.٢٦)$$

The corrected or posteriori state estimate is updated as

$$\underline{\hat{x}}_k^+ = \underline{\hat{x}}_k^- + K_k(\underline{y}_k - \underline{h}_k(\underline{\hat{x}}_k^-, \underline{0})) \qquad (٤.٢٧)$$

Then we used MTLAB program to detect the bias of the gyroscope by comparing the rotational speed obtained from the accelerometer triad with the gyroscope rotational speed.

The gyro-bias compensation by using gyro-free IMU configuration is based on the fact that the gyro-free IMU gives the angular information vector (AIV) which can be corrected from the bias.

Hence, the bias-free AIV can correct for the gyro bias using proper integration filter. This system has been implemented using ٤x MPU٦٠٥٠ accelerometers and ١ gyro triad as shown in Fig.٤,١٢. This method is not able to be used to find the north direction because the dynamic part of the gyro bias is dependent upon the Earth's gravity. However, this method can correct for the relatively large bias values as we will show in the next coming figures. The bias of the AIV can be captured easily because in a static position, the angular acceleration due to the Earth rotation is should be zero, and hence what could be measured is the bias.

Similarly, quadratic terms due to the Earth's rotation are extremely small and hence their values should be very close to zero and it can be used to find the bias in the static position. So, the system can be used to have a bias-free gyro system.
We can use a profile of motion that consists of a static and dynamic phases. The static phase corrects for the AIV-bias, and the dynamic phase corrects for the gyro-bias.
Figures below show how the gyroscope added bias can be detect using a described filter. For that, we used different value of biases and checked the convergence of the estimated bias using described method above.

Fig.٤,١٣ shows the estimation gyro bias for the added gyro bias of ٠,١ rad/s.

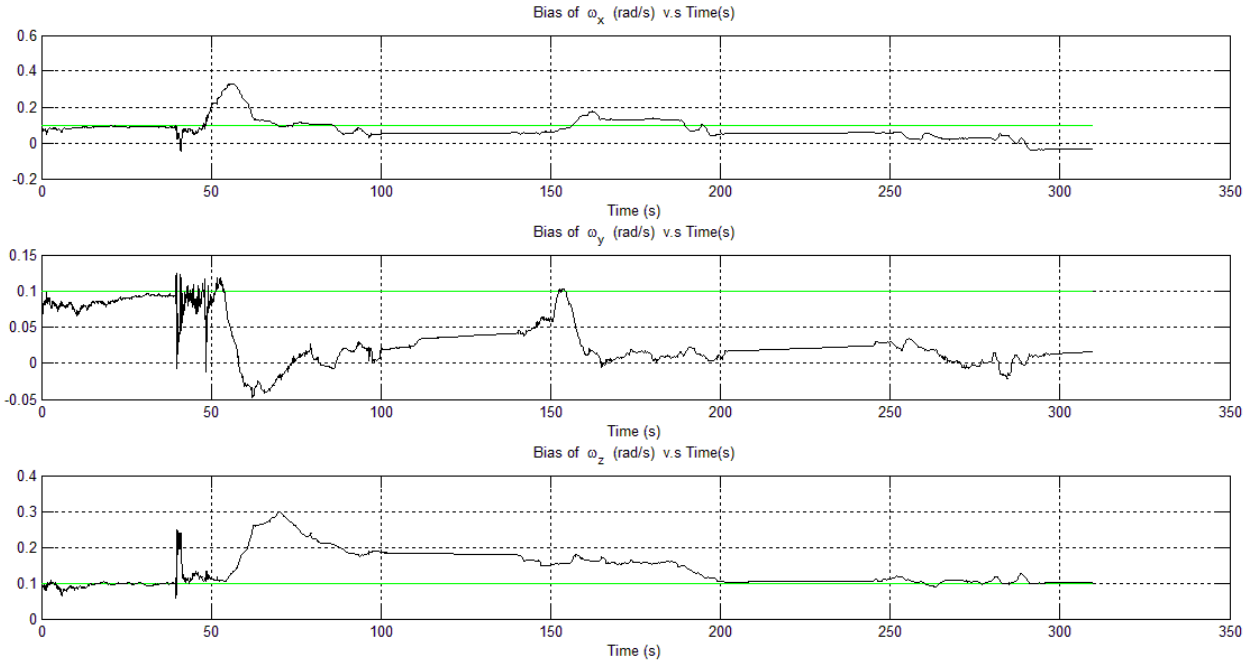

Figure ٤,١٣: Estimated gyro bias when added bias is ٠.١ rad/s

Then the added bias has been increased to ٠,٦ rad/s to see the filter behavior. From Fig.٤,١٤, it is clear that the bigger the added bias, the better the filter performance would be, and the more the detected bias.

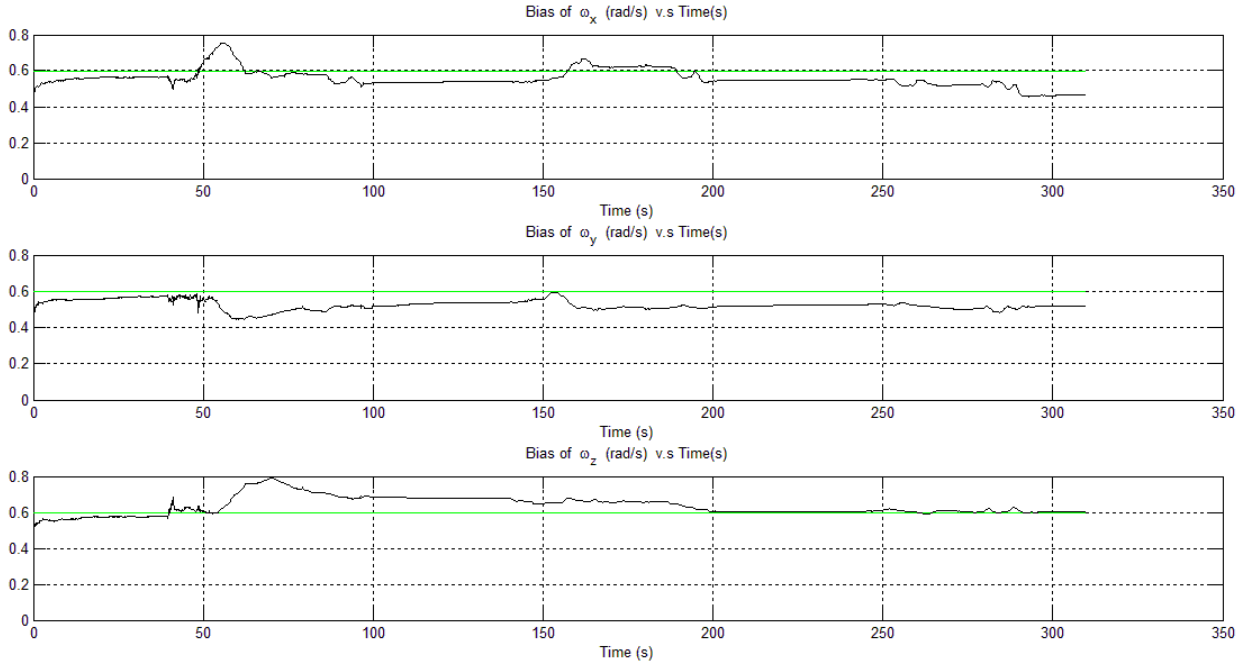

Figure ٤,١٤: Estimated gyro bias when added bias is ٠,٦ rad/s

٥ Conclusion and future work

٥,١ Conclusion

This thesis has described a method for finding a heading direction using low cost MEMS inertial measurement units. The Gyroscope which was used as a stand-alone sensor in the first method is determined to be good enough for the detection of the Earth rotation.

The characterization of the errors that are involved with the MEMS Gyroscope was very crucial, since the noise level of the sensor can be measured and then compared with the signal that is needed to be measured, in this case the Earth's rotation, to see whether the signal to noise ratio is low enough to have a correct measurement.

Allan variance, which is a powerful tool used for IMU sensor error characterization, was used to characterize the errors of gyroscope. The bias instability is measured for each axis of the gyroscopes and shown that each gyroscope axis has different bias instability value. The best candidates, those who have lowest bias instability level, was selected and used since the bias instability threshold must not exceed the Earth's rotation measured signal.

IMU calibration was performed to obtain the meaningful physical value out of a raw data coming out of sensors. Also it united different coordinate

frame of the accelerometer triads into one coordinate frame. Then the bias, misalignment, scale factor was measured, as the output parameters from the calibration process.

Earth rotation has been measured using Maytagging (gyrocompassing) method. The north direction from the detected rotation was calculated and compared with the theoretical value of the turntable rotated angle. The results showed that the proposed type of gyroscope was suited for detecting the low signal of Earth's rotation with some percentage of error that was caused by the bias fluctuation from temperature change between the measurements.

Finally, the accelerometer triad was used to detect the bias fluctuation of the gyroscope. Extended Kalman filter was used to calculate rotational speed out of the linear acceleration from the accelerometer. Then the rotational speed from accelerometers was compared with the gyroscope rotational speed to track the bias fluctuation of the gyroscope. While the frame was moved in a ٣D space, the gravity affected the gyroscope measurements and let to making it somehow difficult to track the whole part of gyroscope bias, so we added some bigger value to check the functionality of the filter.

٥.٢ Future work

- The north finding system should be tested on an actual ground navigation platform.
- Kalman filter can be used with taking temperature compensation into account to increase the performance of the gyroscope.
- Real-time system can be realized on the BeagleBoard's operating system like Xeonomai to reduce the latency of the data logging process and gaining higher sampling rate out of the system.

Appendix I.

C- code

/****MPU٦٠٥٠.c******/**

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <linux/i٢c-dev.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sched.h>
#include <fcntl.h>
#include <stdint.h> //used for int١٦_t
#include <errno.h>
#include "MPU٦٠٥٠.h"
#include "i٢c.h"

unsigned char XACCH;
```

```
unsigned char YACCH;
unsigned char ZACCH;
unsigned char XACCL;
unsigned char YACCL;
unsigned char ZACCL;
unsigned char XGYROH;
unsigned char YGYROH;
unsigned char ZGYROH;
unsigned char XGYROL;
unsigned char YGYROL;
unsigned char ZGYROL;
unsigned char FIFO;
unsigned char TEMPH;
unsigned char TEMPL;
unsigned char interrupt;

float sensor١[٥٠٠٠٠٠٠]={٠},sensor٢[٥٠٠٠٠٠٠]={٠},
sensor٣[٥٠٠٠٠٠٠]={٠},
sensor٤[٥٠٠٠٠٠٠]={٠};
int١٦_t aX١,aY١,aZ١,gX١[٥٠٠٠٠٠٠]={٠},gY١[٥٠٠٠٠٠٠]={٠},
gZ١[٥٠٠٠٠٠٠]={٠},aX٢,aY٢,aZ٢;
 //index  ١ reffers to rensor on bus-٣ with address ٠x١٨
 //index  ٢ reffers to rensor on bus-٣ with address ٠x١٩
 int١٦_t aX٣,aY٣,aZ٣,gX٣,gY٣,gZ٣,aX٤,
 aY٤,aZ٤,gX٤,gY٤,gZ٤,temp;
int write_address(unsigned char reg){
    buf[٠] = reg;
    if (write(i٢c_file,buf,١) != ١) {
        printf("Failed to write to the i٢c bus.\n");
        printf("%s\n",strerror(errno));
        return ٠;
    }
    return ١;
}

/*********************************************************
*********/
//Write a byte to an address
int write_byte(unsigned char reg, unsigned char data){

    buf[٠] = reg;
    buf[١] = data;

    if (write(i٢c_file,buf,٢) != ٢) {
        printf("Failed to write to the i٢c bus.\n");
        printf("%s\n\n",strerror(errno));
        return ٠;
    }
    return ١;
```

```
}

/*********************************************************/
//Read a byte from the current address
int read_current_byte(unsigned char * data){

    if (read(i٢c_file,buf,١) != ١) {
        printf("Failed to read from the i٢c bus.\n");
        printf("%s\n\n",strerror(errno));
        return ٠;
    }
    *data = buf[٠];
    return ١;
}
/*********************************************************/
//Read a byte from the passed register
int read_byte(unsigned char reg, unsigned char * data){

    //Write the register's address
    if(write_address(reg) == ٠)
        return ٠;

    //Read from that address
    return read_current_byte(data);
}

/****************** MAIN*****************************/
int main() {

     int add=٠x٦٨,i;

     for(i=٠;i<٢;i++)
          {
          init_i٢c(add,٢);
          write_byte(PWR_MGMT_١,PWR_MGMT_١_V_R); //reset
the device
          write_byte(USER_CTRL,٠x٠٠ );
          write_byte(USER_CTRL,٠x٠٣ );
           close(i٢c_file);
          add=٠x٦٩;
       }
         add=٠x٦٨;
     for(i=٠;i<٢;i++)
          {
          init_i٢c(add,٣);
          write_byte(PWR_MGMT_١,PWR_MGMT_١_V_R); //reset
the device
          write_byte(USER_CTRL,٠x٠٠ );
          write_byte(USER_CTRL,٠x٠٣ );
```

```
           close(i٢c_file);
           add=٠x١٩;
       }

      add=٠x١٨;
  //initialize sensors on bus ٢
     for(i=٠;i<٢;i++)
          {
          init_i٢c(add,٢);  //open i٢c bus, address
٠x١٨.when i becomes ١, sensor with address ٠x١٩ will be
initialized
             write_byte(PWR_MGMT_١,  PWR_MGMT_١_V); //wake-
up the sensor
          write_byte(INT_ENABLE,  INT_ENABLE_V); //enable
data ready interrupt
          write_byte(PWR_MGMT_٢,PWR_MGMT_٢_V_R);
//deactivate standby mode
             write_byte(SMPLRT_DIV,  SMPLRT_DIV_V);
//sample rate = gyro output
          write_byte(CONFIG,CONFIG_V); //gyro output = ١kHz
          write_byte(GYRO_CONFIG,  GYRO_CONFIG_V); //gyro
scale +/- ٢٥٠ deg/sec
          write_byte(ACCEL_CONFIG,  ACCEL_CONFIG_V); //acc
scale +/- ٢g
          write_byte(USER_CTRL,USER_CTRL_V );
          write_byte(FIFO_EN,٠x٠٨);       //enable FIFO
buffer
           close(i٢c_file);

          add=٠x١٩;
       }
         add=٠x١٨;  //reset address, since it became ٠x١٩
from previous step

   //initialize sensors on bus ٣
     for(i=٠;i<٢;i++)
          {
          init_i٢c(add,٣);  //open i٢c bus, address
٠x١٨.when i becomes ١, sensor
with address ٠x١٩ will be initialized
          write_byte(PWR_MGMT_١,  PWR_MGMT_١_V); //wake-up
the sensor
          write_byte(INT_ENABLE,  INT_ENABLE_V);
          write_byte(PWR_MGMT_٢,PWR_MGMT_٢_V_R);
//deactivate standby
               mode
             write_byte(SMPLRT_DIV,  SMPLRT_DIV_V);
//sample rate = gyro
         output
```

```
          write_byte(CONFIG,CONFIG_V); //gyro output = ١kHz
          write_byte(GYRO_CONFIG,  GYRO_CONFIG_V); //gyro
scale +/- ٢٥٠
deg/sec
          write_byte(ACCEL_CONFIG,  ACCEL_CONFIG_V); //acc
scale +/- ٢g
          write_byte(USER_CTRL,USER_CTRL_V );
          write_byte(FIFO_EN,٠x٠٨);       //enable FIFO
buffer
          close(i٢c_file);
          add=٠x٦٩;
       }
     /*****************************************************/
          //write data to a txt file
           FILE *p = NULL;
           char *file =
"/home/programs/bias_detection.txt";
           p = fopen(file, "w");
           if (p== NULL) {
           printf("Error in opening a file.");
           }

   usleep(١٠٠٠٠٠); //wait ١٠٠ msec for the data to be ready

/*******************************************************/
     printf("data reading started\n ");
      long n=٠,j=٠;
      int number_of_samples=١٠٠٠;  //data logging for ٢٠
second

       while(n<number_of_samples)
        {
          //reading from sensor ١ (i٢cbus -٢ address ٠x٦٩)
          init_i٢c(٠x٦٩،٢);
          read_byte(INT_STATUS, &interrupt);   //using
polling of data when a
               new data is available
          close(i٢c_file);
          if(interrupt & ٠x٠١){
          init_i٢c(٠x٦٩،٢);
          for(j=n*٦;j<(n*٦+٦);j++)
          {
          read_byte(٠x٧٤,&FIFO);
          sensor١[j]=FIFO;

          }
          read_byte(GYRO_XOUT_H, &XGYROH);
          read_byte(GYRO_XOUT_L, &XGYROL);
          gX١[n]=((XGYROH)<<٨) | XGYROL;
```

```
          read_byte(GYRO_YOUT_H, &YGYROH);
          read_byte(GYRO_YOUT_L, &YGYROL);
          gY١[n]=((YGYROH)<<٨) | YGYROL;
          read_byte(GYRO_ZOUT_H, &ZGYROH);
          read_byte(GYRO_ZOUT_L, &ZGYROL);
          gZ١[n]=((ZGYROH)<<٨) | ZGYROL;
          close(i٢c_file);  //close the connection
/*********************************************************/
               init_i٢c(٠x٦٨،٢);
          for(j=n*٦;j<(n*٦+٦);j++)
          {
          read_byte(٠x٧٤,&FIFO);
          sensor٢[j]=FIFO;

          }
          close(i٢c_file);  //close the connection
          init_i٢c(٠x٦٩،٣);
          for(j=n*٦;j<(n*٦+٦);j++)
          {
          read_byte(٠x٧٤,&FIFO);
          sensor٣[j]=FIFO;

          }
          close(i٢c_file);  //close the connection
          init_i٢c(٠x٦٨،٣);
          for(j=n*٦;j<(n*٦+٦);j++)
          {
          read_byte(٠x٧٤,&FIFO);
          sensor٤[j]=FIFO;

          }
          close(i٢c_file);  //close the connection


          n=n+١;
          }
     }

n=٠;
    while(n<number_of_samples)
        {
     for(j=n*٦;j<(n*٦+١);j++)
          {
           XACCH=sensor١[j];
             XACCL=sensor١[j+١];
          YACCH=sensor١[j+٢];
          YACCL=sensor١[j+٣];
          ZACCH=sensor١[j+٤];
          ZACCL=sensor١[j+٥];
```

```
          }
          aX١=((XACCH)<<٨) | XACCL;
          aY١=((YACCH)<<٨) | YACCL;
          aZ١=((ZACCH)<<٨) | ZACCL;

/*******************************************************/
     for(j=n*١;j<(n*١+١);j++)
          {
          XACCH=sensor٢[j];
          XACCL=sensor٢[j+١];
          YACCH=sensor٢[j+٢];
          YACCL=sensor٢[j+٣];
          ZACCH=sensor٢[j+٤];
          ZACCL=sensor٢[j+٥];
          }
          aX٢=((XACCH)<<٨) | XACCL;
          aY٢=((YACCH)<<٨) | YACCL;
          aZ٢=((ZACCH)<<٨) | ZACCL;
/*******************************************************/
     for(j=n*١;j<(n*١+١);j++)
          {
          XACCH=sensor٣[j];
          XACCL=sensor٣[j+١];
          YACCH=sensor٣[j+٢];
          YACCL=sensor٣[j+٣];
          ZACCH=sensor٣[j+٤];
          ZACCL=sensor٣[j+٥];
          }
          aX٣=((XACCH)<<٨) | XACCL;
          aY٣=((YACCH)<<٨) | YACCL;
          aZ٣=((ZACCH)<<٨) | ZACCL;

/*******************************************************/
     for(j=n*١;j<(n*١+١);j++)
          {
          XACCH=sensor٤[j];
          XACCL=sensor٤[j+١];
          YACCH=sensor٤[j+٢];
          YACCL=sensor٤[j+٣];
          ZACCH=sensor٤[j+٤];
          ZACCL=sensor٤[j+٥];
          }
          aX٤=((XACCH)<<٨) | XACCL;
          aY٤=((YACCH)<<٨) | YACCL;
          aZ٤=((ZACCH)<<٨) | ZACCL;

     fprintf(p,"%d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t
%d\t %d\t %d\t %d\t %d\t %d\t
```

```
%d\n",aX١,aY١,aZ١,aX٢,aY٢,aZ٢,aX٣,aY٣,aZ٣,aX٤,aY٤,aZ٤,gX١[n
],gY١[n],gZ١[n]);

          n=n+١;
           }
          printf("data are successfully written to the
file\n ");
          fclose(p);
          return ٠;
}
```

/***** MPU٦٠٥٠.h ********/**

```
#ifndef _MPU٦٠٥٠_H_
#define _MPU٦٠٥٠_H_
#define ADDRESS_AD٠_LOW     ٠x٦٨ // address pin low (GND),
                  default for InvenSense evaluation board
#define ADDRESS_AD٠_HIGH    ٠x٦٩ // address pin high (VCC)
#define SMPLRT_DIV        ٠x١٩
#define CONFIG            ٠x١A
#define MAG_CTRL          ٠x٠A
#define GYRO_CONFIG       ٠x١B
#define ACCEL_CONFIG      ٠x١C
#define MOT_THR           ٠x١F
#define MOT_DUR           ٠x٢٠
#define FIFO_EN           ٠x٢٣
#define I٢C_MST_STATUS    ٠x٣٦
#define INT_PIN_CFG       ٠x٣٧
#define INT_ENABLE        ٠x٣٨
#define INT_STATUS        ٠x٣A
#define ACCEL_XOUT_H      ٠x٣B
#define ACCEL_XOUT_L      ٠x٣C
#define ACCEL_YOUT_H      ٠x٣D
#define ACCEL_YOUT_L      ٠x٣E
#define ACCEL_ZOUT_H      ٠x٣F
#define ACCEL_ZOUT_L      ٠x٤٠
#define TEMP_OUT_H        ٠x٤١
#define TEMP_OUT_L        ٠x٤٢
#define GYRO_XOUT_H       ٠x٤٣
#define GYRO_XOUT_L       ٠x٤٤
#define GYRO_YOUT_H       ٠x٤٥
#define GYRO_YOUT_L       ٠x٤٦
#define GYRO_ZOUT_H       ٠x٤٧
#define GYRO_ZOUT_L       ٠x٤٨
#define SIGNAL_PATH_RESET     ٠x٦٨
#define MOT_DETECT_CTRL       ٠x٦٩
#define USER_CTRL         ٠x٦A
#define PWR_MGMT_١        ٠x٦B
#define PWR_MGMT_٢        ٠x٦C
#define FIFO_R_W          ٠x٧٤
#define WHO_AM_I          ٠x٧٥
```

```
//giving values to the registers, adding "V" at the end of
the register meaning "value"
#define SMPLRT_DIV_V     ٠x١٣ //so that sample
                              rate=٥٠Hz(ADC)
#define CONFIG_V         ٠x٠٤ //Gyro output =١kHz, filter
                              value=٤, bandwidth=٢٠ Hz
#define GYRO_CONFIG_V    ٠x٠٠ //gyro scale +/-٢٥٠ deg/sec
#define ACCEL_CONFIG_V   ٠x٠٠ //accelerometer range =+/- ٢g
#define FIFO_EN_V        ٠x٠٨ //accel measurements
                                are loaded into FIFO
#define SIGNAL_PATH_RESET_V ٠x٠٧ //reset gyro, acc, and
                    temp sensors to their initial condition
#define USER_CTRL_V      ٠x٤٠ //to enable FIFO buffer
#define PWR_MGMT_١_V     ٠x٠٣ //to wake up the sensors
                         set clock source to Z-axis gyro
#define PWR_MGMT_١_V_R     ٠x٨٠ //reset the device
#define INT_ENABLE_V       ٠x٠١ //enable data ready
interrupt
#define PWR_MGMT_٢_V_R     ٠x٠٠ //reset standby
#define PWR_MGMT_٢_V_X     ٠x٠٤ //set gyro x axis in
standby mode
#define PWR_MGMT_٢_V_Y     ٠x٠٢ //set gyro y axis in
standby mode
#define PWR_MGMT_٢_V_Z     ٠x٠١ //set gyro z axis in
standby mode
#define PWR_MGMT_٢_V_All   ٠x٠٧ //all in standby
#endif /* _MPU٦٠٥٠_H_ */
```

/*******I٢C.h********/

```
#ifndef _i٢c_H_
#define _i٢c_H_
char buf[١٠]={٠};
float data;
int i٢c_file;
char filename[٤٠];
//opening the i٢c
int init_i٢c(int addr,int num) {
/*******************************************************/
       //communicate with bus ٢
     if(num==٢)
     {
          sprintf(filename,"/dev/i٢c-٢");
          if ((i٢c_file = open(filename, O_RDWR)) < ٠) {
          printf("open error!\n");
          exit(١);
          }
          if (ioctl(i٢c_file,I٢C_SLAVE,addr) < ٠) {
          printf("address error!\n");
          exit(١);
```

```
          }
          return ١;
         }
/*****************************************************/
       //communicate with bus ٣
      if(num==٣)
     {
          sprintf(filename,"/dev/i٢c-٣");
          if ((i٢c_file = open(filename, O_RDWR)) < ٠) {
          printf("open error!\n");
          exit(١);
          }
          if (ioctl(i٢c_file,I٢C_SLAVE,addr) < ٠) {
          printf("address error!\n");
          exit(١);
          }
          return ١;
         }
     return ٠;

#endif
```

Appendix II.

MATLAB code

/****Accelerometers calibration******/**

```
load data; %data is the raw data file from the sensors
Ax=data(:,١);
Ay=data(:,٢);
Az=data(:,٣);
Bx=data(:,٤);
By=data(:,٥);
Bz=data(:,٦);
Cx=data(:,٧);
Cy=data(:,٨);
Cz=data(:,٩);
Dx=data(:,١٠);
Dy=data(:,١١);
Dz=data(:,١٢);

g=٩,٨١٠٥;
%create ٤ measurmets
p١=١٠٠;
p٢=٣٠٠٠;
A١=[mean(Ax(p١:p٢)) mean(Ay(p١:p٢)) mean(Az(p١:p٢))];
B١=[mean(Bx(p١:p٢)) mean(By(p١:p٢)) mean(Bz(p١:p٢))];
C١=[mean(Cx(p١:p٢)) mean(Cy(p١:p٢)) mean(Cz(p١:p٢))];
```

```
D١=[mean(Dx(p١:p٢)) mean(Dy(p١:p٢)) mean(Dz(p١:p٢))];

p١=٦٠٠٠;
p٢=٨٠٠٠;
A٢=[mean(Ax(p١:p٢)) mean(Ay(p١:p٢)) mean(Az(p١:p٢))];
B٢=[mean(Bx(p١:p٢)) mean(By(p١:p٢)) mean(Bz(p١:p٢))];
C٢=[mean(Cx(p١:p٢)) mean(Cy(p١:p٢)) mean(Cz(p١:p٢))];
D٢=[mean(Dx(p١:p٢)) mean(Dy(p١:p٢)) mean(Dz(p١:p٢))];

p١=١٠٠٠٠;
p٢=١٢١٠٠;
A٣=[mean(Ax(p١:p٢)) mean(Ay(p١:p٢)) mean(Az(p١:p٢))];
B٣=[mean(Bx(p١:p٢)) mean(By(p١:p٢)) mean(Bz(p١:p٢))];
C٣=[mean(Cx(p١:p٢)) mean(Cy(p١:p٢)) mean(Cz(p١:p٢))];
D٣=[mean(Dx(p١:p٢)) mean(Dy(p١:p٢)) mean(Dz(p١:p٢))];

p١=١٤٦٠٠;
p٢=١٦٦٠٠;
A٤=[mean(Ax(p١:p٢)) mean(Ay(p١:p٢)) mean(Az(p١:p٢))];
B٤=[mean(Bx(p١:p٢)) mean(By(p١:p٢)) mean(Bz(p١:p٢))];
C٤=[mean(Cx(p١:p٢)) mean(Cy(p١:p٢)) mean(Cz(p١:p٢))];
D٤=[mean(Dx(p١:p٢)) mean(Dy(p١:p٢)) mean(Dz(p١:p٢))];
%we have four measurments we find MA MB MC MD ba bb bc bd
ax=٠;
ay=٠;
az=g;
O١=[ax ay az ٠ ٠ ٠ ٠ ٠ ٠ ١ ٠ ٠;
    ٠ ٠ ٠ ax ay az ٠ ٠ ٠ ٠ ١ ٠;
    ٠ ٠ ٠ ٠ ٠ ٠ ax ay az ٠ ٠ ١];

ax=٠;
ay=g;
az=٠;

O٢=[ax ay az ٠ ٠ ٠ ٠ ٠ ٠ ١ ٠ ٠;
    ٠ ٠ ٠ ax ay az ٠ ٠ ٠ ٠ ١ ٠;
    ٠ ٠ ٠ ٠ ٠ ٠ ax ay az ٠ ٠ ١];

ax=g;
ay=٠;
az=٠;

O٣=[ax ay az ٠ ٠ ٠ ٠ ٠ ٠ ١ ٠ ٠;
    ٠ ٠ ٠ ax ay az ٠ ٠ ٠ ٠ ١ ٠;
    ٠ ٠ ٠ ٠ ٠ ٠ ax ay az ٠ ٠ ١];
ax=٠;
ay=٠;
az=-g;
```

```
O٤=[ax ay az ٠ ٠ ٠ ٠ ٠ ٠ ١ ٠ ٠;
    ٠ ٠ ٠ ax ay az ٠ ٠ ٠ ٠ ١ ٠;
    ٠ ٠ ٠ ٠ ٠ ٠ ax ay az ٠ ٠ ١];
O=[O١;O٢;O٣;O٤];
ZA=[A١';A٢';A٣';A٤'];

THETA=inv(O'*O)*O'*ZA;
mA=[THETA(١) THETA(٢) THETA(٣);
    THETA(٤) THETA(٥) THETA(٦);
    THETA(٧) THETA(٨) THETA(٩)];
bA=[THETA(١٠); THETA(١١); THETA(١٢)];

ZB=[B١';B٢';B٣';B٤'];

THETA=inv(O'*O)*O'*ZB;
mB=[THETA(١) THETA(٢) THETA(٣);
    THETA(٤) THETA(٥) THETA(٦);
    THETA(٧) THETA(٨) THETA(٩)];
bB=[THETA(١٠); THETA(١١); THETA(١٢)];

ZC=[C١';C٢';C٣';C٤'];

THETA=inv(O'*O)*O'*ZC;
mC=[THETA(١) THETA(٢) THETA(٣);
    THETA(٤) THETA(٥) THETA(٦);
    THETA(٧) THETA(٨) THETA(٩)];
bC=[THETA(١٠);THETA(١١);THETA(١٢)];

ZD=[D١';D٢';D٣';D٤'];

THETA=inv(O'*O)*O'*ZD;
mD=[THETA(١) THETA(٢) THETA(٣);
    THETA(٤) THETA(٥) THETA(٦);
    THETA(٧) THETA(٨) THETA(٩)];
bD=[THETA(١٠); THETA(١١);THETA(١٢)];
npts=length(Ax);

for i=١:npts
temp=inv(mA)*([Ax(i);Ay(i);Az(i)]- bA);
Axc(i,١)=temp(١);
Ayc(i,١)=temp(٢);
Azc(i,١)=temp(٣);

temp=inv(mB)*([Bx(i);By(i);Bz(i)]- bB);
Bxc(i,١)=temp(١);
Byc(i,١)=temp(٢);
Bzc(i,١)=temp(٣);

temp=inv(mC)*([Cx(i);Cy(i);Cz(i)]- bC);
```

```
Cxc(i,١)=temp(١);
Cyc(i,١)=temp(٢);
Czc(i,١)=temp(٣);


temp=inv(mD)*([Dx(i);Dy(i);Dz(i)]- bD);
Dxc(i,١)=temp(١);
Dyc(i,١)=temp(٢);
Dzc(i,١)=temp(٣);

end

save Cal_data mA mB mC mD bA bB bC bD

/******Gyro calibration******/
clear all;
load gyro_calibration_fifo; % gyro_calibration_fifo is the
file name of raw data form the gyroscope
Gx١=gyro_calibration_fifo(:,١);
Gy١=gyro_calibration_fifo(:,٢);
Gz١=gyro_calibration_fifo(:,٣);

% Y up, cw
p١=١٠;
p٢=٢٥٠٠;
A١=[mean(Gx١(p١:p٢)) mean(Gy١(p١:p٢)) mean(Gz١(p١:p٢))];
% Y up, ccw
p١=٢٦٠٠;
p٢=٥٠٠٠;
A٢=[mean(Gx١(p١:p٢)) mean(Gy١(p١:p٢)) mean(Gz١(p١:p٢))];
% X up, cw
p١=١٤٠٠٠;
p٢=١٦٦٠٠;
A٣=[mean(Gx١(p١:p٢)) mean(Gy١(p١:p٢)) mean(Gz١(p١:p٢))];

% X up, ccw
p١=١٧٠٠٠;
p٢=١٩٥٠٠;
A٤=[mean(Gx١(p١:p٢)) mean(Gy١(p١:p٢)) mean(Gz١(p١:p٢))];

% Z up, cw
p١=٢١٦٠٠;
p٢=٣٠٠٠٠;
A٥=[mean(Gx١(p١:p٢)) mean(Gy١(p١:p٢)) mean(Gz١(p١:p٢))];

% Z up, ccw

p١=٣٠٢٠٠;
p٢=٣٣٠٠٠;
```

```
A٦=[mean(Gx١(p١:p٢)) mean(Gy١(p١:p٢)) mean(Gz١(p١:p٢))];
wconst=١٠*(٢*pi)/٦٠;
%we have ٦ measurments we find MAg MBg MCg MDg bag bbg bcg
bdg
gx=٠;
gy=wconst;
gz=٠;


O١=[gx gy gz  ٠   ٠  ٠   ٠  ٠  ٠  ١ ٠ ٠;
     ٠  ٠  ٠  gx gy gz   ٠  ٠  ٠  ٠ ١ ٠;
     ٠  ٠  ٠  ٠   ٠  ٠   gx gy gz ٠ ٠ ١];
gx=٠;
gy=-wconst;
gz=٠;

O٢=[gx gy gz  ٠   ٠  ٠   ٠  ٠  ٠  ١ ٠ ٠;
     ٠  ٠  ٠  gx gy gz   ٠  ٠  ٠  ٠ ١ ٠;
     ٠  ٠  ٠  ٠   ٠  ٠   gx gy gz ٠ ٠ ١];
gx=wconst;
gy=٠;
gz=٠;

O٣=[gx gy gz  ٠   ٠  ٠   ٠  ٠  ٠  ١ ٠ ٠;
     ٠  ٠  ٠  gx gy gz   ٠  ٠  ٠  ٠ ١ ٠;
     ٠  ٠  ٠  ٠   ٠  ٠   gx gy gz ٠ ٠ ١];

gx=-wconst;
gy=٠;
gz=٠;

O٤=[gx gy gz  ٠   ٠  ٠   ٠  ٠  ٠  ١ ٠ ٠;
     ٠  ٠  ٠  gx gy gz   ٠  ٠  ٠  ٠ ١ ٠;
     ٠  ٠  ٠  ٠   ٠  ٠   gx gy gz ٠ ٠ ١];
gx=٠;
gy=٠;
gz=wconst;
O٥=[gx gy gz  ٠   ٠  ٠   ٠  ٠  ٠  ١ ٠ ٠;
     ٠  ٠  ٠  gx gy gz   ٠  ٠  ٠  ٠ ١ ٠;
     ٠  ٠  ٠  ٠   ٠  ٠   gx gy gz ٠ ٠ ١];
gx=٠;
gy=٠;
gz=-wconst;
O٦=[gx gy gz  ٠   ٠  ٠   ٠  ٠  ٠  ١ ٠ ٠;
     ٠  ٠  ٠  gx gy gz   ٠  ٠  ٠  ٠ ١ ٠;
     ٠  ٠  ٠  ٠   ٠  ٠   gx gy gz ٠ ٠ ١];

O=[O١;O٢;O٣;O٤;O٥;O٦];
ZA=[A١';A٢';A٣';A٤';A٥';A٦'];
```

```
THETA=inv(O'*O)*O'*ZA;
mAg=[THETA(١) THETA(٢) THETA(٣);
    THETA(٤) THETA(٥) THETA(٦);
    THETA(٧) THETA(٨) THETA(٩)];
bAg=[THETA(١٠); THETA(١١); THETA(١٢)];

% first sensor calibrated value, when rotated around z in
clockwise
% direction

for i=١:٢٤٠٠
temp=inv(mAg)*([Gx١(i);Gy١(i);Gz١(i)]- bAg);
Gxc١(i,١)=temp(١);
Gyc١(i,١)=temp(٢);
Gzc١(i,١)=temp(٣);

end

save Cal_data_gyro mAg  bAg
```

List of references

١. Ezzaldeen Edwan, Stefan Knedlik, Otmar Loffeld," Constrained Angular MotionEstimation in a Gyro-Free IMU" IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS VOL. ٤٧, NO. ١ JANUARY ٢٠١١

٢. BeagleBoard System Reference Manual Rev C٥, July ١٥, ٢٠١١.

٣. Fernando Suárez Lainez," Implementation of Attitude and Heading Reference System Using Beagleboard ", ٢٣ Nov ٢٠١١.

٤. MPU-٦٠٠٠ and MPU-٦٠٥٠ Product Specification Revision ٣,٢, ١٦ Nov ٢٠١١.

٥. MPU-٦٠٠٠/MPU-٦٠٥٠ ٩-Axis Evaluation Board User Guide ١ Dec ٢٠١١.

٦. Lucian Ioan Iozan, Martti Kirkko-Jaakkola, Jussi Collin, Jarmo Takala, Corneliu Rusu," North Finding System Using a MEMS Gyroscope", the European Navigation Conference on Global Navigation Satellite Systems, October ١٩-٢١, ٢٠١٠, Braunschweig, Germany.

٧. Oliver J. Woodman, "An introduction to inertial navigation", August ٢٠٠٧.

٨. Yuan Long Wei, Min Cheol Lee, "Mobile Robot Autonomous Navigation Using MEMS Gyro North Finding Method in Global Urban System", ٢٠١١ IEEE International Conference on Mechatronics and Automation August ٧ - ١٠, Beijing, China.

٩. Dr. Walter Stockwell,"" Bias Stability Measurement: Allan Variance".

١٠. Dan Simon, "Kalman Filtering".

١١. Cornell University subject MI٦٣,"Kalman Filter Applications", September ٢٠٠٨.

١٢. JackW Judy, "Microelectromechanical systems (MEMS): fabrication, design and applications",٢٦ November ٢٠٠١.

١٣. Randall K. Curey, Michael E. Ash, Leroy O. Thielman, Cleon H. Barker,"Proposed IEEE Inertial System Terminology Standard and Other Inertial Sensor Standards".

١٤. IEEE Standard Specification Format Guide and Test Procedure for Linear, Single-Axis, Nongyroscopic Accelerometers, ٢٠٠٨.

١٥. Navid Yazdi, Farrohk Ayazi, and Khalil Najafi, "Micromachined Inertial Sensors", Proceedings of the IEEE, VOL. ٨٦, NO. ٨, AUGUST ١٩٩٨.

١٦. Claudia C. Meruane Naranjo, "Analysis and Modeling of MEMS based Inertial Sensors", Stockholm ٢٠٠٨.

١٧. M.A Hopcroft." Calculates standard Allan Deviation (ADEV) of a time domain signals."MATLAB file exchange, ٠٣ Nov ٢٠١٠.WEB.١٨,٠٨,٢٠١٢ <http://www.mathworks.com/matlabcentral/fileexchange/١٣٢٤٦-allan>.