2017-01-01

# Microelectromechanical Systems Inertial Measurement Unit As An Attitude And Heading Reference System

Francisco Javier Salcedo Ortega
*University of Texas at El Paso*, fjsalcedoortega@gmail.com

# MICROELECTROMECHANICAL SYSTEMS INERTIAL MEASUREMENT UNIT AS AN ATTITUDE AND HEADING REFERENCE SYSTEM

FRANCISCO JAVIER SALCEDO ORTEGA

Master's Program in Mechanical Engineering

APPROVED:

Louis Everett, Ph.D., Chair

Virgilio Gonzalez, Ph.D.

Arifur R. Khan, Ph.D.

Angel Flores Abad, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

# Dedication

To my parents.

# MICROELECTROMECHANICAL SYSTEMS INERTIAL MEASUREMENT UNIT AS AN ATTITUDE AND HEADING REFERENCE SYSTEM

By

FRANCISCO JAVIER SALCEDO ORTEGA, B.S IN MECHANICAL ENGINEER

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Mechanical Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 2017

# ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. Everett for the support and guidance he gave me during my studies. He always had insightful information and the resources I needed to complete this project. I would also like to extend my appreciation to Dr. Virgilio Gonzalez for his support, as well as the other members of my evaluation committee Dr. Khan and Dr. Flores.

To my parents who with their love, encouragement, understanding, and support made it possible through all of my years of study. This would not have been possible without them.

Finally I would like to thank all of my friends, family, and everyone in my life who offered help and support during my thesis and career.

# ABSTRACT

Microelectromechanical systems inertial measurement units (MEMS IMU) are an accessible technology that can be used as an attitude and heading reference system (AHRS); they are low cost and functional enough to be used in many Inertial Navigation Systems (INS). However, the gyroscope component on the IMU tends to severely drift over time without the use of filtering or other navigation systems reference such as a Global Positioning System (GPS) device. This thesis is focused on working with a variety of IMUs to describe the accuracy, precision, and error propagation. It also works as an introductory guide to the world of IMUs and filtering, thus, methods for detecting noise components and key factors for selection of IMU's are described. Additionally, educational laboratory workshops that can be implemented on a classroom environment will be presented. Three different models of IMU's with different degrees of freedom (DOF) were used on the testing: MPU-6050(10 DOF), MPU-6050 (6 DOF) and ALTIMU-10 V5 (10 DOF). The tests were focused mainly on measuring the performance of the gyroscope and accelerometer components with the addition of different filters used. In addition to the implementation of several codes for the visualization of the results. The results show conclusive evidence that the filters and algorithms significantly improve the outputs the sensor. Not only reducing the individual noise factors that affect each sensor, but enhancing their abilities as well. With the filters applied, the sensors are able to provide an adequate AHRS.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1: INTRODUCTION

In the past inertial sensors consisted of high precision mechanical gyroscopes and accelerometers, these were expensive devices that were typically only used for high-end space applications. The Gyroscopes and accelerometers cost could go as high as a small vehicle, however with the advances of the electronics technology, the cost of these sensors has dropped substantially. IMUs can now be obtained for as low as $5.00 USD in the form of microelectromechanical systems (MEMS) inertial measurement units (IMUs). The MEMS IMUs are very small devices composed of micro components ranging from 0.001 mm to 0.1 mm, these components are made of silicon, polymers, metals and ceramics. The IMUs also have a CPU (central process unit) typically in the form of a microcontroller [1]. IMUs are widely used for aircraft, submarine, spacecraft technology as navigational systems, and because of their low cost and low weight they are also ideal for aerospace applications.

As mentioned before, the IMUs are composed of several components such as: accelerometers, gyroscopes, magnetometers, and sometimes barometers. This thesis discusses the basics of the IMUs, and continues with the applications and limitations of them. This includes the key components for selection of an appropriate device of a desired application. Information about its functionality in space with the corresponding limitations and additional requirements is also described.

One of the main limitation of the IMUs is their tendency to drift as time passes (because of their gyroscope component), this error propagation is a big factor present in all IMUs and it is imperative to determine the source of this noise. A method to detect the noise components using Allan Variance; which estimates the stability due to noise is explored. Additionally, in order to mitigate this error propagation there are methods that can be used, these are known as filters. These methods and how to implement them in the IMUs are demonstrated as well.

## 1.1 BACKGROUND

The art of navigation has always been pursued by humanity through its history, as it allowed the gathering of resources and commerce. An early example comes from the ancient Greeks with seamanship, which is the art of guiding the vessels on open sea and stablishing its position. Thus as it can be seen navigation has been around long before the MEMS technology concept even existed. While the MEMS technology may be relatively new, its components are not; the gyroscope goes back as far as 1817. Made by the German Johann Bohhenberger, back then it was known only as the "machine". It owes its modern name to the French Leon Foucault in 1852 [2]. Accelerometers on the other hand did not became commercially available until the 1920's, in the form of resistance based accelerometers designed by Burton McCollum and Orville Peters [3].

The first inertial sensors were developed and tested by rocket designers in the 1930's. However the first use of inertial navigations can be attributed to the Germans in 1942 using the V-2 missile. In the United States the development of inertial navigation systems began in the late 1940's and the 1950's by the Michigan Institute of Technology (MIT), Northrop, and Autonetics under Air Force sponsorship. The inertial sensor technology was perfected by institutions such as Drapers Labs, and by the 1960's inertial navigation made possible many of the great spaceflight achievements such as the Apollo program; at the same time inertial guidance systems were used for military and commercial planes.

IMUs can now be found on drones, aircrafts, submarines, cars, and spacecraft applications such as satellites and landers [4]. The Inertial navigation system (INS) market has grown significantly on the last couple of years. According to Markets and Markets a market research company, the INS products are projected to grow from 4.64 billion UDS in 2015 to 8.87 billion USD by 2020 [5]. More relevant, IMU's are expected to be a big contributor to the INS market.

## 1.2 INERTIAL MEASUREMENT UNITS

Currently IMUs are integrated sensor packages composed mainly of accelerometers and gyroscopes, but they can also include other sensors such as magnetometers and barometers. As

previously mentioned IMUs are used in the Inertial Navigation System technology. They do so by employing a technology called Dead Reckoning (DR) navigation. Dead reckoning works by using a previous position to calculate the current position, without the need of an external reference [6]. DR navigation is known for cumulative errors in its position without the use of an external reference such as Global Position System (GPS). As it then follows, IMU's allow for an excellent short term accuracy, however long term accuracy is not very reliable without the use of said external references or some type of filter [7]. The main advantage that IMUs have in addition to their low weight, it's their ability to operate in a wide variety of environments. This is true as long as it is within the limits of the device, which can be found on the datasheet. Datasheet's information and how to properly interpret it will be discussed later on this chapter.

An IMU can measure acceleration rates, rotation rates, earth's magnetic field, and pressure using a combination of its sensors: gyroscope, accelerometer, magnetometer, and barometer. More information on the components of the IMU will be discussed in greater depth in the following sections of this chapter. An IMU sensor also allows the possibility to determine a body's attitude, this can be made possible by computing the rotation in three dimensions: pitch, yaw and roll (shown in figure 1.1).

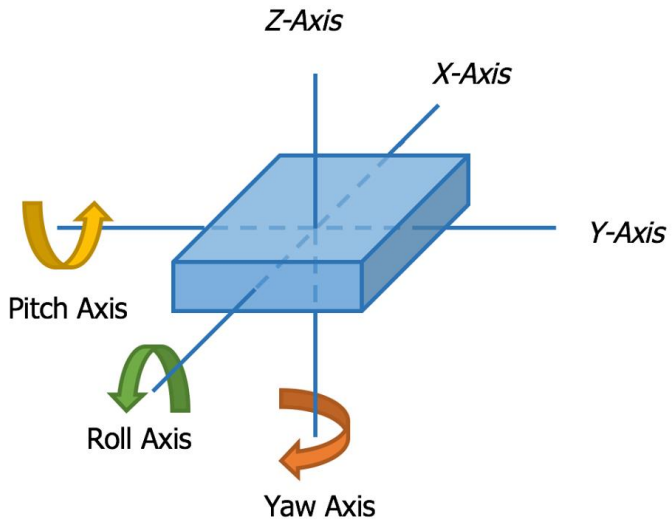



Figure 1.1: Inertial Measurement Unit Yaw, Roll and Pitch example

In this example yaw is left and right movement about the Z-axis, roll is the rotation about the X-axis and finally pitch is up or down about the Y-axis (note that the axes "X, Y and Z" in this thesis are user defined, and can change depending of the convention defined by each user).

The IMU's are typically classified based on their Degrees of Freedom (DOF), which are the number of independent readings they are able to take. Most modern IMU's have at least 6 DOFs, consisting of a 3-axis accelerometer combined with a 3-axis gyroscope [8].

### 1.2.1 Grading performance of IMUs

The first thing to keep in mind when selecting an IMU is the intended use of the application, when selecting a device the following parameters should be taken into consideration:

Table 1.1: Summary of the main sensor characteristics [9]

| | |
|---|---|
| Range | Maximum and minimum value measurable |
| Resolution | Smallest measurable increment |
| Sensing frequency | Maximum frequency measurable |
| Accuracy | Error of measurement, in % full scale deflection |
| Reliability | Life cycle of device |
| Size | Dimensions and mass or sensor |
| Operating environment | Operating temperature and environmental conditions |
| Drift | Deviation of measurement over a time period |
| Cost | Purchase and maintenance cost of sensor |

Besides DOF, IMUs are also divided into performance rating grades according to their bias instability specifications, which spans a huge range in terms of product prices and performance. Typically with the lowest grade being used for consumer products, and the highest for critical strategic applications. This is important because lower bias stability means less error when

integrating the output, which means less drifting over time; naturally the better performance grade equals a higher cost [10].

There are six levels of gyroscope grades, it should be noted however that there is no set parameter for the actual definitions of high, medium, and low end performance. These values are just for reference and since gyroscopes are constantly improving this table could become obsolete later down the line. Nonetheless the following values serve as a basis for selecting an IMU [9].

Table 1.2: Gyroscopes Grades based on Bias Stability [10]

| **Performance Grade** | **Bias Stability** |
|---|---|
| Consumer | 30-1000 °/hr. |
| Industrial | 1-30 °/hr. |
| Tactical | 0.1-30 °/hr. |
| High-end Tactical | 0.1-1 °/hr. |
| Navigation | 0.01-0.1 °/hr. |
| Strategic | 0.0001-0.01 °/hr. |

As with the gyroscopes the accelerometers can also be lumped into different categories based on their range, resolution, and sensing frequency. Accelerometers are measured in 'g' units which is the acceleration due to the gravity of the earth $9.81\frac{\mathrm{m}}{s^2}$, more information on this topic will be given in the following sections. The following figure (Fig. 1.2) provides some insight of the most common types of accelerometers. Accelerometers typically have a resolving power of about $10^6$ positions. The inertial navigation force-balance grade accelerometers are capable of distinguishing up to $10^8$ positions. Because the accelerometers are capable of sensing both the acceleration and deceleration, an accelerometer sensing range is twice the value found in the Fig. 1.2. For example a force-balance (servo) accelerometer would have an output range of up to ± 200g. Force-balance accelerometers tend to have a smaller sensing range because of their closed-loop design, while the Piezoresistive accelerometers tend to have the highest output range which can go up to ± 200,000g. Piezoresistive accelerometers tend to be more rugged and versatile as

they have the widest span on their operating range, they can measure micro-gs up to high impact measurements. The ruggedness and robustness of accelerometers is an important characteristic of the accelerometer. For example it may only be required to measure forces up to 10g, however the accelerometer would be required to survive higher level impacts that are in the scale of 1000g or more [9].

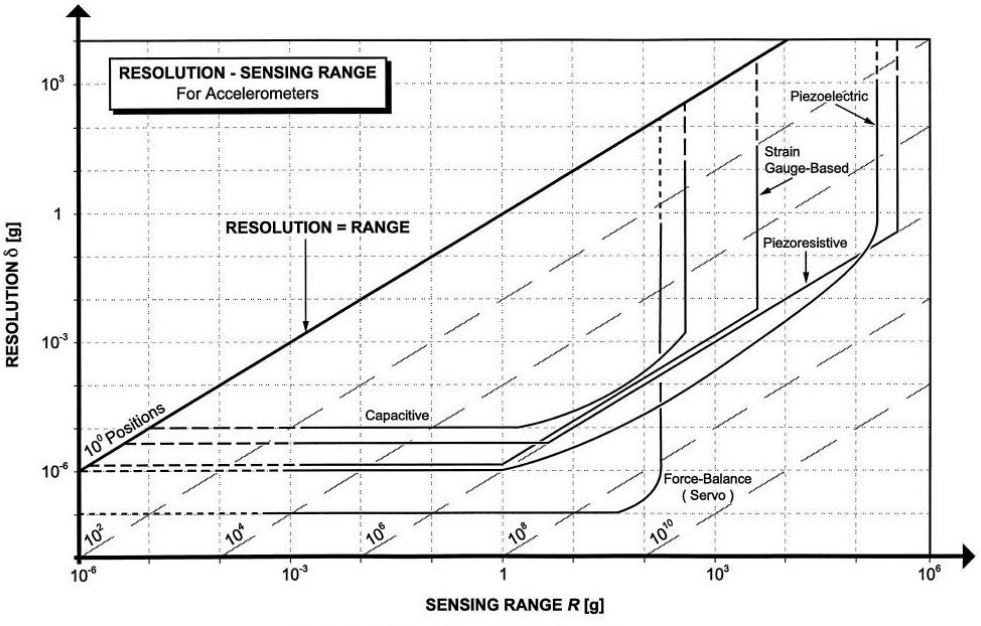



Figure 1.2: Resolution vs. Sensing Range for typical accelerometers [9]

Accelerometers that have a higher sensing range typically have higher natural frequencies, this is because they work with higher vibrations, thus higher frequencies. The higher accelerations also produce higher forces on the inertial mass, thus a higher spring stiffness is also needed to limit displacements. The resolution of the accelerometers is governed by the noise level of the sensor, high-sensitivity accelerometers have a larger signal-to-noise ratios (SNR); which is the level of the signal compared to the level of noise. In turn high-sensitivity accelerometers also have the finest resolutions, they are often designed for micro-g measurements; however they tend to have less

sensing range. As it can be seen it is always a matter of what exactly the application is going require when choosing an accelerometer [11].

As mentioned before accelerometers are used in vibration, therefore sensing frequency is vital when classifying the performance of the accelerometers. Fig. 1.2 shows the comparison of sensing frequency and sensing range. Once again it can be seen that the piezoelectric and piezoresistive accelerometers have the highest sensing frequencies, exceeding 10 kHz. It is worth noting that accelerometers work the best at frequencies that are below the resonant frequency; that is a frequency in which an external force drives another system to oscillate with greater amplitude [12].

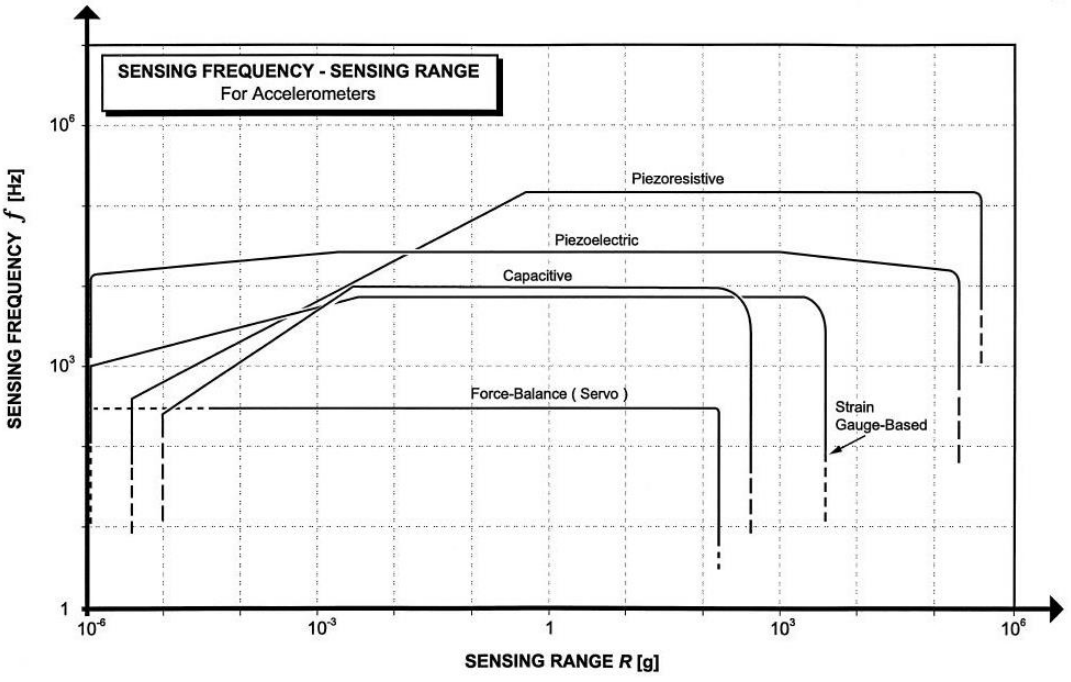



Figure 1.3: Sensing Frequency vs. Sensing Range for typical accelerometers [9]

Additionally, IMUs as a system can be grouped into four levels of magnitude by performance and price. When selecting a sensor the first step will be to determine what grade of precision is required for the application. The IMUs can be divided into: marine and navigation grade, tactical grade, industrial grade, automotive grade, and finally consumer grade.

*Marine and navigation grade*: besides any custom inertial navigation system used on missiles, *marine grade* is the highest grade of sensor that is available in the market. It provides the overall best performance for determining position and orientation. The marine grade IMUs are typically used on ships, submarines and spacecraft technology. They can provide a drift of less than 1.8 km per day without the need of an external reference such as GPS, note that IMU technology is continuously improving so in the future this number could decrease significantly. This drift means that even if you left the device stationary, it would still be 1.8 km away from the actual position of the device. This drift is due to bias, lack of calibration, and other errors in the sensors which will be discussed in more detail in chapter 2.

*Navigation grade:* these IMUs have slightly less performance compared to the marine grade. They are used on commercial airlines and military aircraft with a drift that is less than 1500m per hour. The navigation grade systems are costly, with prices ranging up to $100,000 USD and are about 6in x 6in x 6in in size. When combined with a GPS they provide very accurate positioning systems than can go in a range of centimeters in real time.

*Automotive and consumer grade:* these are the lowest grade of the inertial sensors. The difference between these and *industrial* grade is the quality of the sensor calibration that they received. As the MEMS technology continues to improve, in the future it may be possible that consumer grade IMUs will achieve what is currently defined as navigation or even marine grade [8]. An example of a tactical grade MEMS IMU is the OEM-HG1900 made by Honeywell.

Figure 1.4: Tactical grade MEMS OEM-HG1900 IMU [13]

Determining the right type of inertial sensor for the application can be a difficult task due to confusion and lack on data available. The information found on datasheets for IMUs can be very confusing and make potential buyers unknown of the differences between products. For this reason this thesis will provide information on how to read and interpret the datasheets, as well as specific characteristics to look for when selecting the sensor for your application.

### 1.2.2 Datasheets

Once a sensor has been selected, its functionality must be understand. Most of the information on the particular sensor can be found on its datasheet, which can be found online most of the time. When users are not familiar with datasheets they can be confusing and intimidating when first looking at them. This thesis will go through the steps and information of the key elements to look for when selecting a MEMS IMU. For the purpose of this the MPU-6050 will be chosen. This section will focus on general recommendations that can be applied not just for IMUs, but for any sensors.

The first thing to do is find the datasheet of the sensor, these can be found online by typing the name of the sensor or at the vendor's website. If the datasheet is not found it can be requested to the supplier directly. Most datasheets will be provided in PDF format, with the first pages dedicated to the table of contents for easy access to the particular information the user is looking for. The next section will give a product overview of all the features the sensor has, this information will give a quick impression if it's right for the intended application. It can also work as a basis for comparing multiple sensors for features and components. After that comes arguably the most important section; *Characteristics.* Some datasheets will divide them into *Mechanical characteristics* and *Electrical characteristics,* while others may list them all as *Electrical characteristics;* in the case of the MPU-6050 it is listed as electrical. These include the main characteristics of the two most important components of the IMU: the gyroscope and the accelerometer. They are typically given at room temperature (25 degree Celsius) and at the nominal voltage for the sensor. They include the measurement range, sensitivity, frequencies, low

and high pass filters (if any), output rate, bias error, and the temperature effect on the sensor. All of these will be discussed in more detail during this subchapter.

We will begin with the characteristics of the gyroscope, as this is usually the first table of characteristics that will be available in the datasheet.

**6.1 Gyroscope Specifications**

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, $T_A$ = 25°C

| PARAMETER | CONDITIONS | MIN | TYP | MAX | UNITS | NOTES |
|---|---|---|---|---|---|---|
| **GYROSCOPE SENSITIVITY** | | | | | | |
| Full-Scale Range | FS_SEL=0 | | ±250 | | °/s | |
| | FS_SEL=1 | | ±500 | | °/s | |
| | FS_SEL=2 | | ±1000 | | °/s | |
| | FS_SEL=3 | | ±2000 | | °/s | |
| Gyroscope ADC Word Length | | | 16 | | bits | |
| Sensitivity Scale Factor | FS_SEL=0 | | 131 | | LSB/(°/s) | |
| | FS_SEL=1 | | 65.5 | | LSB/(°/s) | |
| | FS_SEL=2 | | 32.8 | | LSB/(°/s) | |
| | FS_SEL=3 | | 16.4 | | LSB/(°/s) | |
| Sensitivity Scale Factor Tolerance | 25°C | -3 | | +3 | % | |
| Sensitivity Scale Factor Variation Over Temperature | | | ±2 | | % | |
| Nonlinearity | Best fit straight line; 25°C | | 0.2 | | % | |
| Cross-Axis Sensitivity | | | ±2 | | % | |
| **GYROSCOPE ZERO-RATE OUTPUT (ZRO)** | | | | | | |
| Initial ZRO Tolerance | 25°C | | ±20 | | °/s | |
| ZRO Variation Over Temperature | -40°C to +85°C | | ±20 | | °/s | |
| Power-Supply Sensitivity (1-10Hz) | Sine wave, 100mVpp; VDD=2.5V | | 0.2 | | °/s | |
| Power-Supply Sensitivity (10 - 250Hz) | Sine wave, 100mVpp; VDD=2.5V | | 0.2 | | °/s | |
| Power-Supply Sensitivity (250Hz - 100kHz) | Sine wave, 100mVpp; VDD=2.5V | | 4 | | °/s | |
| Linear Acceleration Sensitivity | Static | | 0.1 | | °/s/*g* | |
| **SELF-TEST RESPONSE** | | | | | | |
| Relative | Change from factory trim | -14 | | 14 | % | 1 |
| **GYROSCOPE NOISE PERFORMANCE** | **FS_SEL=0** | | | | | |
| Total RMS Noise | DLPFCFG=2 (100Hz) | | 0.05 | | °/s-rms | |
| Low-frequency RMS noise | Bandwidth 1Hz to10Hz | | 0.033 | | °/s-rms | |
| Rate Noise Spectral Density | At 10Hz | | 0.005 | | °/s/ √ Hz | |
| **GYROSCOPE MECHANICAL FREQUENCIES** | | | | | | |
| X-Axis | | 30 | 33 | 36 | kHz | |
| Y-Axis | | 27 | 30 | 33 | kHz | |
| Z-Axis | | 24 | 27 | 30 | kHz | |
| **LOW PASS FILTER RESPONSE** | | | | | | |
| | Programmable Range | 5 | | 256 | Hz | |
| **OUTPUT DATA RATE** | | | | | | |
| | Programmable | 4 | | 8,000 | Hz | |
| **GYROSCOPE START-UP TIME** | **DLPFCFG=0** | | | | | |
| ZRO Settling (from power-on) | to ±1°/s of Final | | 30 | | ms | |

Figure 1.5: MPU-6050Gyroscope characteristics [14]

Beginning from the top with *the gyroscope sensitivity*. The sensitivity has something called the full-scale range (FS), this can be selected and configured by the user. It is the maximum rate that the gyroscope can generate for the output. In the case of the MPU-6050 there are four scales too choose from: 250, 500, 1000, and 2000 degrees per second. Let us imagine the user selected 250 degrees per second, and the gyroscope was spinning at 360 degrees per second (one revolution per second). The gyroscope would show an output of 250 degrees per second, because the scale

would be saturated. However, if the user selected 2000 degrees per second, the output would have read the 360 degrees. This may cause users to believe that it is always better to select the highest range scale since it would be able to read the widest ranges of values, however this is not the case. In the first column there is something called sensitivity scale factor, for this example let us look at the specifications and sensitivity scale factor from the ALTIMU-10 V5:

### 4.1 Mechanical characteristics

@ Vdd = 1.8 V, T = 25 °C unless otherwise noted.

**Table 3. Mechanical characteristics**

| Symbol | Parameter | Test conditions | Min. | Typ.(1) | Max. | Unit |
|---|---|---|---|---|---|---|
| LA_FS | Linear acceleration measurement range | | | ±2 | | *g* |
| | | | | ±4 | | |
| | | | | ±8 | | |
| | | | | ±16 | | |
| G_FS | Angular rate measurement range | | | ±125 | | dps |
| | | | | ±245 | | |
| | | | | ±500 | | |
| | | | | ±1000 | | |
| | | | | ±2000 | | |
| LA_So | Linear acceleration sensitivity | FS = ±2 | | 0.061 | | mg/LSB |
| | | FS = ±4 | | 0.122 | | |
| | | FS = ±8 | | 0.244 | | |
| | | FS = ±16 | | 0.488 | | |
| G_So | Angular rate sensitivity | FS = ±125 | | 4.375 | | mdps/LSB |
| | | FS = ±245 | | 8.75 | | |
| | | FS = ±500 | | 17.50 | | |
| | | FS = ±1000 | | 35 | | |
| | | FS = ±2000 | | 70 | | |
| LA_SoDr | Linear acceleration sensitivity change vs. temperature(2) | from -40° to +85° delta from T=25° | | ±1 | | % |
| G_SoDr | Angular rate sensitivity change vs. temperature(2) | from -40° to +85° delta from T=25° | | ±1.5 | | % |
| LA_TyOff | Linear acceleration typical zero-*g* level offset accuracy(3) | | | ±40 | | *mg* |
| G_TyOff | Angular rate typical zero-rate level(3) | | | ±10 | | dps |
| LA_OffDr | Linear acceleration zero-*g* level change vs. temperature(2) | | | ±0.5 | | mg/ °C |
| G_OffDr | Angular rate typical zero-rate level change vs. temperature(2) | | | ±0.05 | | dps/°C |
| Rn | Rate noise density | | | 7 | | mdps/√Hz |
| An | Acceleration noise density | FS= ±2 *g* ODR = 104 Hz | | 90 | | µg/√Hz |

Figure 1.6: ALTIMU-10 V5 Mechanical characteristics [15]

It can be seen that increasing the angular rate is correspondingly increasing the mdps/LSB (millidegrees per second at the least significant bit).

So what exactly is this "sensitivity"? This particular IMU has 16 bits of data for the output, the first bit will determine the sign of the reading (positive or negative), while the other 15 bits represent the magnitude of the data. For example: 0000000000000001 would be equal to 0.004375 degrees per second when the measurement range selected is equal to ± 125 degrees per second. Thus, when the measurement range is increased the sensitivity decreases, in other words it can no longer differentiate values that are smaller than the least significant bit.

Going back to the MPU-6050, the sensitivity factor is measured in LSB/°/s (least significant bit in degrees per second). What this means is that it would be required to divide the "raw" output of the gyroscope by 131 (this will be explored in detail later during this chapter) to get the degrees per second that the gyroscope has moved. It can be seen that this number decreases as the measurement range is increased, thus the higher the range the less sensitive the device becomes. This applies to all of the sensors in the IMU, as there will always be a tradeoff between range and sensitivity. As a general rule it is always recommended to select the smallest possible scale that will still be able to meet the operating range of the application in use. In addition, temperature also affects the sensitivity, in Figure 1.4 (MPU-6050) it can be seen that the sensitivity will vary ± 2% from -45°C to 85°C. If the application is not operating in a controlled temperature environment this element will need to be evaluated when selecting the IMU as well. As it can be seen in Figure 1.5 (ALTIMU-10 V5) the sensitivity from temperature is only ± 1% from -45°C to 85°C, in other words this value varies from sensor to sensor. If the sensor has a high sensitivity the earth's rotation should also be taken into account when taking readings from the gyroscope.

Moving on in the table is the *zero rate output level*, this is basically the bias in the readings which this thesis will cover more in depth in the following chapter. This can be explained easily as the difference between the actual rate and the rate that the gyroscope or accelerometer (or any reading for that matter) will display in its output. This means that even if the gyroscope is set at a flat table and readings are taken, the average of that readings will not be 0. This is caused by the

bias that comes from stress during the soldering of the board, this bias is affected by the temperature change as well. The bias can be calculated and subtracted from readings, in addition temperature compensation and calibration will also help with reducing the bias. All of this will be discussed in greater depth in consequent chapters.

The *self-test* section is not common in IMUs, and works as calibration of the device on the MPU-6050, it allows for the testing of mechanical and electrical portions of the sensors, there are self-test registers in this case from 13 to 16 that allows the tests to be activated, when the values of the self-test response are within the minimum and maximum values limits of the production specification, the device has passed the test, otherwise it has failed, these values are included in the datasheets. [14]

After that we continue with the *noise performance* or *rated noise density (Rn),* this noise is not predictable, nor can it be calibrated. Evidently when selecting a sensor this number should be as low as possible, however when comparing sensors you will run into different units for this value. This is because manufacturers can use different methods to define the rated noise density, as it will be shown next. Using the fast Fourier transfer (FFT), which is a method used to calculate the discrete Fourier Transform (DFT) of a sequence of N numbers. FFT is used to obtain the frequency content of a signal [16] (in this case the gyroscope output data). The noise will be given in degrees per second per root hertz ($\frac{^\circ/s}{\sqrt{Hz}}$) or millidegrees per second, however it can also be defined in Power Spectral Density (PSD) in units of degrees per second per hertz ($\frac{^\circ/s^2}{Hz}$) (these can be converted easily). The other method is the Angle Random Walk (ARW), this can be defined using the Allan Variance and will be further discussed in chapter two. In short it is basically the drift over time in the gyroscope readings using the units of degrees per root hour ($\frac{^\circ}{\sqrt{hr}}$). In order to convert between ARW to FFT and DFT some simple formulas can be used: ARW = (60) ($\sqrt{PSD}$) and ARW = (60) (FFT).

Finally the *output data* rate which is very self-explanatory, the datasheets typically list the highest output data rate (Hz). Logically, the faster this value is, the faster the data will be able to be recorded. This rate can be user selectable as well, it should be noted however that the higher

the frequency the higher the noise. The accelerometers datasheet works in the same way and for this reason some datasheets even lumps them together as they follow the same rules, with the key difference being the units of measurement.

After this we will move on to the *Electrical characteristics,* as explained before some datasheets will lump the *Mechanical characteristics* together. This section will be explained using the MPU-6050 datasheet that is shown on Figure 1.7.

It starts with the *Temperature Sensor* which is basically the range the sensor will be able to operate at. While this may be a fairly simple concept to understand, nonetheless it is key when selecting the IMU. Most sensors work at a range of -40 to 85 degrees Celsius, if the intended application will go below or above those limits the necessary modifications should be made.

**6.3 Electrical and Other Common Specifications**

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, $T_A$ = 25°C

| PARAMETER | CONDITIONS | MIN | TYP | MAX | Units | Notes |
|---|---|---|---|---|---|---|
| **TEMPERATURE SENSOR** | | | | | | |
| Range | | | -40 to +85 | | °C | |
| Sensitivity | Untrimmed | | 340 | | LSB/°C | |
| Temperature Offset | 35°C | | -521 | | LSB | |
| Linearity | Best fit straight line (-40°C to +85°C) | | ±1 | | °C | |
| **VDD POWER SUPPLY** | | | | | | |
| Operating Voltages | | 2.375 | | 3.46 | V | |
| Normal Operating Current | Gyroscope + Accelerometer + DMP | | 3.9 | | mA | |
| | Gyroscope + Accelerometer (DMP disabled) | | 3.8 | | mA | |
| | Gyroscope + DMP (Accelerometer disabled) | | 3.7 | | mA | |
| | Gyroscope only (DMP & Accelerometer disabled) | | 3.6 | | mA | |
| | Accelerometer only (DMP & Gyroscope disabled) | | 500 | | µA | |
| Accelerometer Low Power Mode Current | 1.25 Hz update rate | | 10 | | µA | |
| | 5 Hz update rate | | 20 | | µA | |
| | 20 Hz update rate | | 70 | | µA | |
| | 40 Hz update rate | | 140 | | µA | |
| Full-Chip Idle Mode Supply Current | | | 5 | | µA | |
| Power Supply Ramp Rate | Monotonic ramp. Ramp rate is 10% to 90% of the final value | | | 100 | ms | |
| **VLOGIC REFERENCE VOLTAGE** | MPU-6050 only | | | | | |
| Voltage Range | VLOGIC must be ≤VDD at all times | 1.71 | | VDD | V | |
| Power Supply Ramp Rate | Monotonic ramp. Ramp rate is 10% to 90% of the final value | | | 3 | ms | |
| Normal Operating Current | | | 100 | | µA | |
| **TEMPERATURE RANGE** | | | | | | |
| Specified Temperature Range | Performance parameters are not applicable beyond Specified Temperature Range | -40 | | +85 | °C | |

Figure 1.7: MPU-6050 Electrical Specifications [14]

The *VDD Power Supply* is the voltage that must be supplied to the IMU in order for it to function. This table often includes a typical value of operation, along with a minimum and maximum value. It also includes the operating current of the gyroscope, accelerometer, and any other device that the sensor will draw while under operation. These values are important to keep in mind in order to be able to provide all the amperage that is required for the desired application.

The *Vlogic reference voltage* is the voltage needed for the logic levels of its I2C, which is the serial computer bus interface. And finally the *Temperature Range* is the range that the sensor will be able to perform. Other important sections include the *absolute maximum ratings* which are self-explanatory, they are the maximum settings that the sensor can work at. These values are useful when thinking about the environment the sensor will be exposed to.

**6.9 Absolute Maximum Ratings**

Stress above those listed as "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to the absolute maximum ratings conditions for extended periods may affect device reliability.

| Parameter | Rating |
|---|---|
| Supply Voltage, VDD | -0.5V to +6V |
| VLOGIC Input Voltage Level (MPU-6050) | -0.5V to VDD + 0.5V |
| REGOUT | -0.5V to 2V |
| Input Voltage Level (CLKIN, AUX_DA, AD0, FSYNC, INT, SCL, SDA) | -0.5V to VDD + 0.5V |
| CPOUT (2.5V ≤ VDD ≤ 3.6V ) | -0.5V to 30V |
| Acceleration (Any Axis, unpowered) | 10,000g for 0.2ms |
| Operating Temperature Range | -40°C to +105°C |
| Storage Temperature Range | -40°C to +125°C |
| Electrostatic Discharge (ESD) Protection | 2kV (HBM);<br>250V (MM) |
| Latch-up | JEDEC Class II (2),125°C<br>±100mA |

Figure 1.8: MPU-6050 Absolute Maximum Ratings table [14]

When the user is programming the sensor there is another section that needs to be taken in mind, and that is the *FIFO mode* section. FIFO stands for first in first out buffer, most of the

sensors have different modes to this buffer and the user should select the one that is convenient for its application.

When buffering, the output is stored in an area of the memory that is specially created to temporarily store data. The buffer size depends on the sensor, this is useful for several reasons as it reduces processing power and computing time. It also allows the sensor to take and store additional readings, however when using an application that is heavily dependable on real time data buffering may not be useful. While there are other features in the datasheet, keeping these key characteristics in mind will allow the user to select the most optimal sensor for the intended application.

### 1.2.3 Space applications and limitations of device

The conditions in outer space are harsh, it is in fact one of the most extreme environments imaginable. Space-crafts are subjected to extreme temperatures both hot and cold, but it is important to understand that since outer space is a vacuum (very close to a perfect vacuum in fact), space by itself has no temperature. However there is temperature change due to radiation, be it the background radiation left from the Big Bang or the radiation exerted by the sun [17]. Temperatures can range from the extreme cold (hundreds below freezing), to millions of Kelvins when close to the sun. The extreme temperatures are not the only condition that affects a spacecraft, the space conditions begin even before leaving the earth. The spacecraft has to deal with the lunch, which results in violently shaking vibrations with several g's of force. With this in mind, engineers need to integrate a cooling system and insulators to be able to withstand the temperature, vibration, and harmful radiation from outer space. As mentioned previously, the IMUs datasheets provide the operating range of the sensor and operating environment, this will give the base-line for creating the appropriate enclosure that will be required. Engineers are able to simulate some of the outer space conditions by using vibration tables, acoustic chambers, temperature chambers, and vacuum chambers. These tests are typically known as validation testing.

## 1.3 IMU COMPONENTS

As it has been previously stated trough this thesis, an inertial measurement unit can consist of several components. However the two most important ones are the accelerometer and gyroscope as combining these sensors allows for attitude measurement. As it will be seen in chapter 2 combining additional sensors allows for superior readings and reduction of noise using filtering techniques. In the subsequent subchapters the components of the IMU will be described, their functionality explained, and finally useful equations for the accelerometers and gyroscope components will be explored.

### 1.3.1 Accelerometers

An accelerometer as its name implies is a device that reads linear acceleration or the rate of change of velocity of a body in its own rest frame [18]. Accelerometers are relative to free-fall or to an inertial frame, this is called proper acceleration [19]. For this reason gravity will always be measured by accelerometers (unless the accelerometer is in orbit or perfect free-fall). While this may seem like a noise factor, in reality it is extremely useful as it helps the accelerometer by providing a vertical reference. Accelerometers will always output accelerations in the body-fixed coordinate frame (X, Y and Z), which makes it possible for an accelerometer to calculate roll or pitch angles. Accelerometers are used in several applications in today's technology to sense orientation, vibration, and shock. They are an integral part of the inertial navigation systems, tablets, cameras, video games, cellphones, and several more equipment. In the MEMS technology accelerometers work by measuring the change in capacitance, an accelerometer can be easily imagined as a mass on a spring that is confined to move along one direction and fixed outer plates. Thus when an acceleration in a particular direction occurs, the mass will move though that direction and the capacitance (fixed outer plates) will be able to measure the change it senses, this will be processed and will correspond to a particular acceleration value [20]. Figure 1.9 visually shows the effect of an applied acceleration and how the MEMS accelerometer would react.

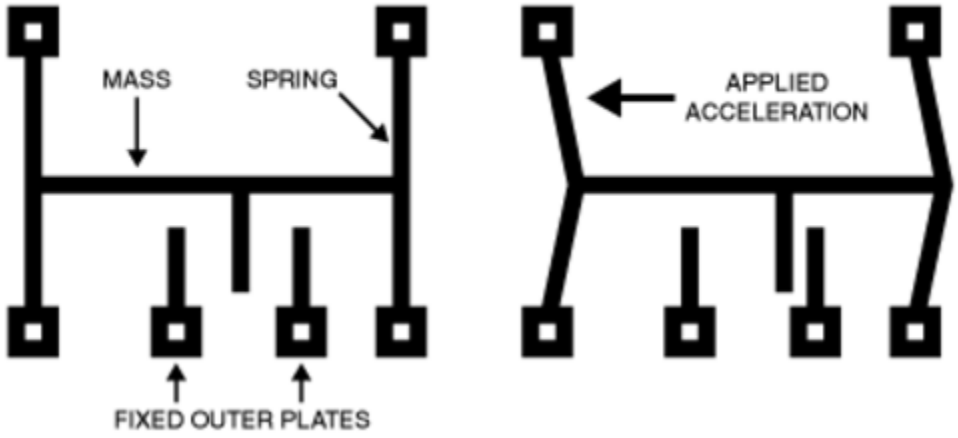



Figure 1.9: Effect of applied acceleration on MEMS accelerometer [20]

When a body such as a car is accelerating, the proper acceleration can be felt by the occupants, this acceleration is called "g-force". While the name implies that this is a force, this is not correct, as it is really an acceleration (that will be measured by the accelerometer). The unit of measure of the g-force (g) is the acceleration due to gravity at the earth's surface (the standard gravity) $9.81\frac{m}{s^2}$. These forces can be felt both horizontally and vertically, however it is important to understand that gravity alone does not produce any actual g-force. The g-force is the result of the resistance of the gravitational acceleration by mechanical forces, and these mechanical forces produce the g-force acceleration on a mass. The g-force results in stress and strain on objects because they are transmitted to its surface, thus large g-forces are dangerous to humans, structures, and machines [21].

So how exactly do accelerometers deliver this information? There are two main categories for accelerometers: digital and analog. The digital accelerometers use serial protocols such as I2C, SPI, and USART. While analog accelerometers output a voltage within a predefined range that will have to be converted to a digital value using an analog to digital converter (ADC). Microcontrollers typically have a built-in ADC module, this will be explained in more detail later on, but first lets focus on how the accelerometer works.

To make it easier to understand how an accelerometer works, let us imagine a ball inside a box, figure 1.10 will be used to demonstrate this.

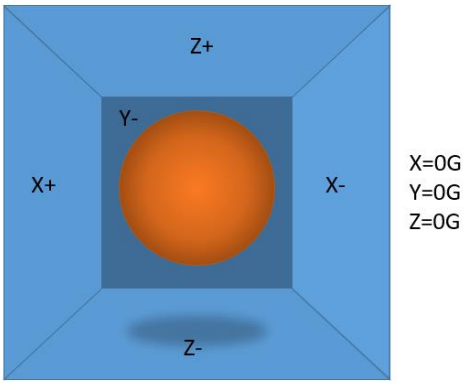



Figure 1.10: Single-axis accelerometer in a gravitational free environment [22]

As it can be seen the box is placed somewhere in the universe where there are no gravitational fields or any other fields that could affect the ball's position (the Y+ axis is removed in order to observe the inside of the box), thus the ball will float in the middle of the box permanently. Suddenly, the box is moved to the right and accelerates 1g, what happens next is that the ball will hit the left wall (X+). The accelerometer would then measure the pressure force that the ball applies to the wall, it would output a value of 1g on the X axis (this can be seen on figure 1.11).

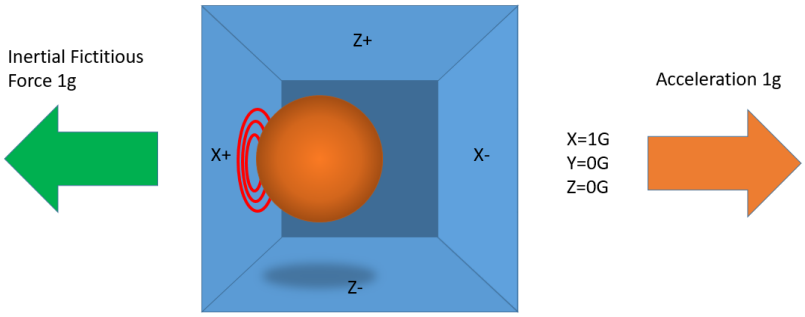



Figure 1.11: Single-axis accelerometer with a force applied [22]

As it can be noted the accelerometer detects a force that is directed in the opposite direction from the acceleration vector. This force is called Inertial Force or Fictitious Force, which is the resultant force from an acceleration of the non-inertial reference frame, in other words the acceleration the balls will have when the box is moved. This explains why accelerometer measures acceleration indirectly through a force applied to one of its wall, it might be easy to think that this force is only cause by an acceleration, but this is not always the case.

If this model is moved away from the gravitational free environment to earth, the ball will fall on the Z-wall and apply a force of 1g on the bottom wall (figure 1.12). The box will not be moving but it will still get a reading of -1g on the Z axis. This pressure force was caused by a gravitation force not an acceleration, therefore while the accelerometers may measure proper acceleration, accelerometers in essence also can measure forces.

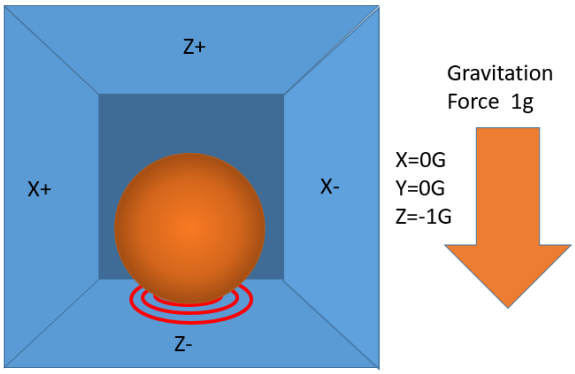



Figure 1.12: Gravitation Force effect on single-axis accelerometer [22]

This box example gives the output of an accelerometer on a single axis, however most accelerometers (and the ones this thesis is covering) are tri-axial, in other words they can detect forces on all three axes. Figure 1.13 shows the effect of gravitation on a tri-axial accelerometer.

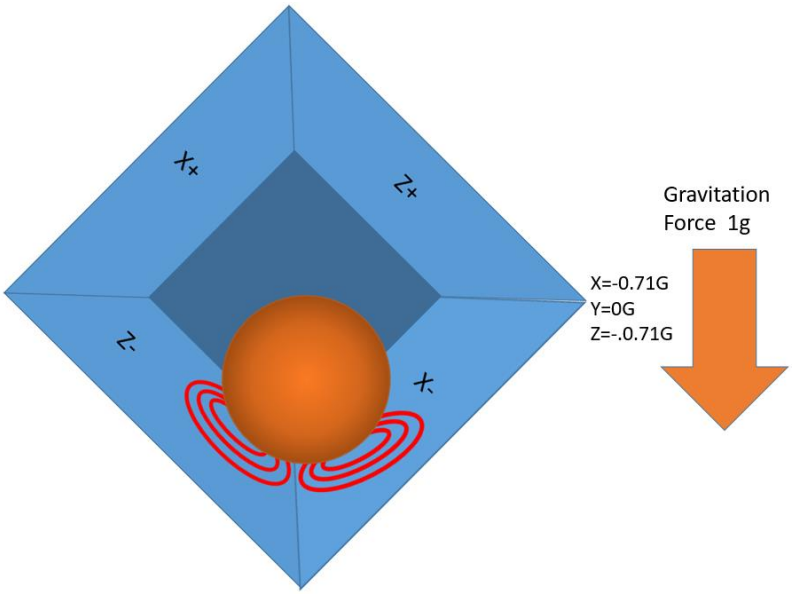



Figure 1.13: Gravitation Force effect on tri-axial accelerometer [22]

This figure gives a more realistic version of how an accelerometer would work in the MEMS IMU. The values 0.71 are an approximation of the equations the accelerometer uses to calculate the accelerations. To understand more clearly how these formulas work, a new model will be used shown on Figure 1.14, it displays a fixed coordinate system to the accelerometer axis.

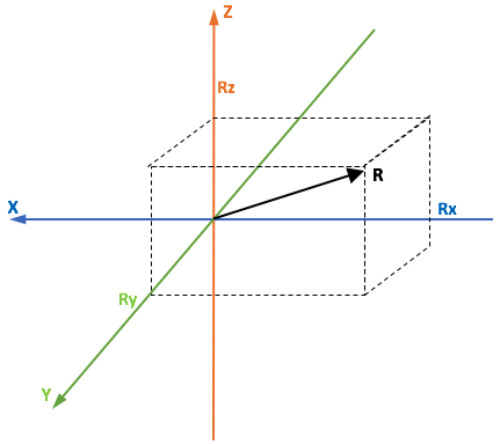



Figure 1.14: Accelerometer in a fixed coordinate system [22]

On this model each of the axes is perpendicular to each of the boxes face, and the R vector is the force the accelerometer will be reading. The force vector value can come from the gravitation force, the inertial force, or a combination of both. While the Rx, Ry, and Rz are projections of the R vector on the X, Y, and Z axes respectability. Using the following equation, we have a relation between the vectors:

$$R^2 = Rx^2 + Ry^2 + Rz^2$$

As it can be seen this equation is very similar to the Pythagorean Theorem, it is clearer now how the number 0.71 came to be. Going back to figure 1.13 we have:

$$(1)^2 = (-\sqrt{1/2})^2 + 0^2 + (-\sqrt{1/2})^2$$

Then by substituting R=1 :

$1 = \sqrt{(-\sqrt{1/2})^2 + 0^2 + (-\sqrt{1/2})^2}$ , thus we have Rx = $-\sqrt{1/2}$ , Ry =0, and Rz = $-\sqrt{1/2}$

This is closer to how accelerometers work in real life. When using an analog to digital converter module the accelerometer will output a value in a certain range corresponding to the acceleration calculated. For example taking a 10-bit ADC module it will output values in the range of 0-1023. The reasoning can be easily explain with how binary numbers works: 1023 = $(2^{10}) - 1$ , which is 1024, however you subtract 1 because in binary code 0 counts as a number. Likewise, a 14-bit ADC module will output values in the range of 0 – 16383 (16383 =$(2^{14}) - 1$).

To further explain this, let us imagine that a 10-bit ADC module accelerometer was attached to a small object, and our module gave us the following values. In this example we will be using a tri-axial accelerometer.

AdcRx = 335

AdcRy = 478

AdcRz = 328

ADC modules have their own reference voltage, for this example we can assume it is 5 V. In order to convert the values the accelerometer read, the following formula will be used: VoltsRx = AdcRx *Vref / 1023, likewise for the 14-bit the divider would be 16383, and for an 8-bit this number would be 255. Applying the formula to all 3 readings we get the following:

$$\text{VoltsRx} = \frac{(335)(5V)}{1023} = 1.64\ V$$

$$\text{VoltsRy} = \frac{(478)(5V)}{1023} = 2.33\ V$$

$$\text{VoltsRz} = \frac{(328)(5V)}{1023} = 1.60\ V$$

Going back to the zero-g level error in the data-sheets, which is the value that corresponds to 0 g. In our example accelerometer the 0 g voltage level will be Vzerog = 1.50 V, thus the voltage shifts from zero-g voltage as follows:

ΔVoltsRx = 1.64 V – 1.50 V = 0.14 V

ΔVoltsRy = 2.33 V – 1.50 V = 0.83 V

ΔVoltsRz = 1.60 V – 1.50 V = 0.10 V

This gives us our accelerometer data in volts, however the output is still not in g's (g-force). For this we need to do a final conversion applying the accelerometer sensitivity, as mentioned before this can be found on the data-sheets. In our example accelerometer the sensitivity will be 450 mV/g, because our current accelerometer data is in volts we will do a simply conversion. Dividing our obtained value by 1000; $\frac{450\ mV/g}{1000}$ = 0.450 V/g.

With the sensitivity value in the correct units, the final force values expressed in g's can be obtained with the following formula: Rx = ΔVoltsRx / Sensitivity

Rx = 0.14 V / 0.450 V/g = 0.31 g

Ry = 0.83 V / 0.450 V/g = 1.84 g

Rz = 0.10 V / 0.450 V/g = 0.22 g

If we combine everything into one equation we can get the following formulas for a tri-axial accelerometer:

$$\text{Rx} = \frac{(\frac{(\text{AdcRx})(\text{Vref})}{1023} - \text{Vzerog})}{Sensitivity}$$

$$\text{Ry} = \frac{(\frac{(\text{AdcRy})(\text{Vref})}{1023} - \text{Vzerog})}{Sensitivity}$$

$$\text{Rz} = \frac{(\frac{(\text{AdcRz})(\text{Vref})}{1023} - \text{Vzerog})}{Sensitivity}$$

These 3 equations define the inertial force vector (assuming the device is not subject to other forces other than gravitational force), these equations then calculate the gravitational force vector. The angles between X, Y, Z axes and the force vector R can also be calculated as follows:

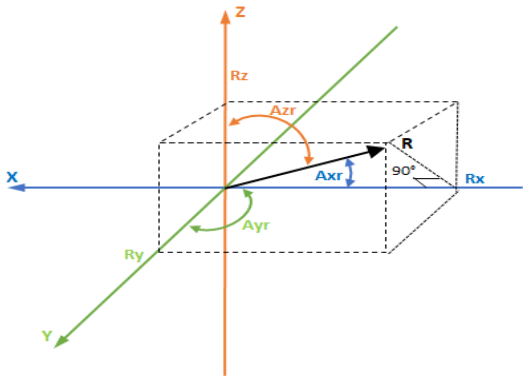



Figure 1.15: Accelerometers angles between axes and force vector [22]

As Figure 1.15 shows, we will define the angles as Axr, Ayr, and Azr. Additionally, there is a right-angle triangle formed by R and Rx, which gives the following formulas:

Cos X = Cos (Axr) = $\frac{Rx}{R}$

Cos Y = Cos (Ayr) = $\frac{Ry}{R}$

Cos Z =Cos (Azr) = $\frac{Rz}{R}$

Deriving this formulas gives us the angles:

Axr = $arcCos(\frac{Rx}{R})$

Ayr = $arcCos(\frac{Ry}{R})$

Azr = $arcCos(\frac{Rz}{R})$

The angles in relation to the ground can also be calculated using the following formulas:

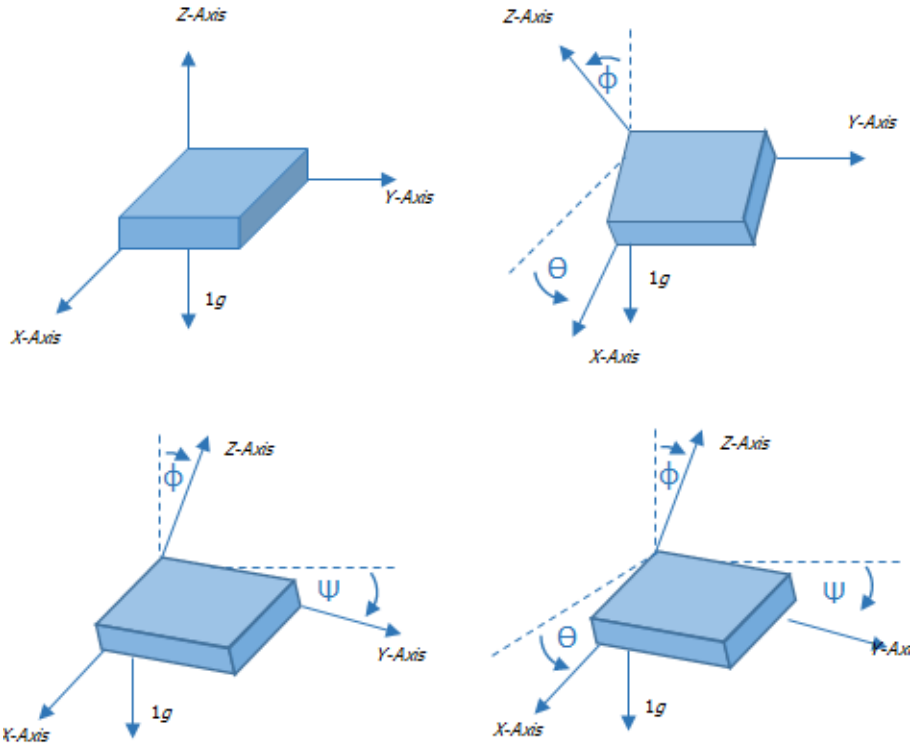



Figure 1.16: Accelerometers angles relative to the ground [22]

We can find pitch (Θ), yaw (Ψ), and roll (ϕ) with basic trigonometry, the equations show the angles of inclination (note that these equations are assuming a perfect accelerometer meaning no sensitivity):

$$\Theta = tan^{-1}\left(\frac{Ax}{\sqrt{Ay^2+Az^2}}\right)$$

$$\Psi = tan^{-1}\left(\frac{Ay}{\sqrt{Ax^2+Az^2}}\right)$$

$$\phi = tan^{-1}\left(\frac{\sqrt{Ax^2+Ay^2}}{Az}\right)$$

There are more intricacies to the accelerometers, including bias errors, sensitivity mismatch, and calibration techniques. All of these will be discussed in more detail in chapter 3.

### 1.3.2 Gyroscopes

Gyroscopes or gyro for short is a device that measures angular rate about a particular axis, this means the rate of change of an angle with time. Gyroscopes were invented in 1817, and just

like the IMUs (they are after all a vital component of them) these sensors are used in vehicle-control, aviation, aerospace, navigation, robotic, and military applications. The MEMS gyros are also used for camera stabilization, cellphones and video games [23]. Gyroscopes are able to measure angular rates using the Coriolis Effect. Similar to the g-force, the Coriolis force (named for Gaspard-Gustave Coriolis) is not a real force [20] [23]. To explain the Coriolis force, let us think of a moving body from an engineering standpoint, there are many forces that must be taken into account. We have physical-contact forces such as friction, forces that act at a distance like gravitational and electromagnetic forces, and then there are the "fictitious" forces. The fictitious forces include the inertial force (which we already covered in the accelerometers), centrifugal force, and the Coriolis force. As mentioned before, the inertial force (mass times absolute acceleration) is equal to the physical force that must be applied to a mass to cause it to accelerate. Centrifugal force ($\frac{mv^2}{r}$), is equal to the force that must be applied perpendicularly to the trajectory of the mass to move along a circular path of radius r at a constant speed. As usual, this is more easily explained visually. In Figure 1.17 we will imagine a fixed body-reference frame that will experience all the "fictitious" forces. We will pretend we already know the inertial force, therefore we also know absolute acceleration during the whole motion with respect to the ground reference frame R.

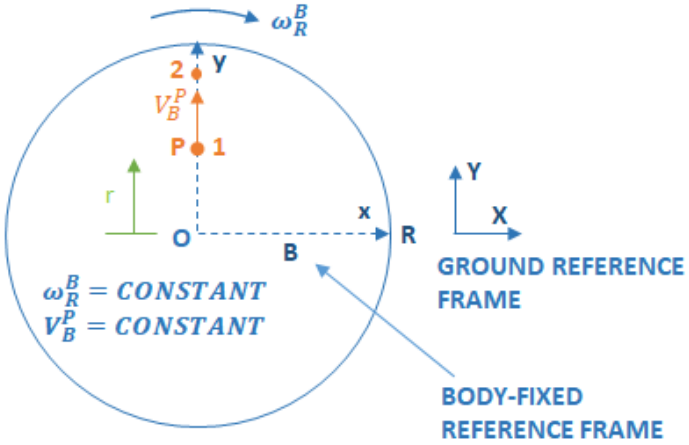



Figure 1.17: "Fictitious" forces experience by body [23]

If the absolute force is known, we can calculate all the forces that a body can feel.

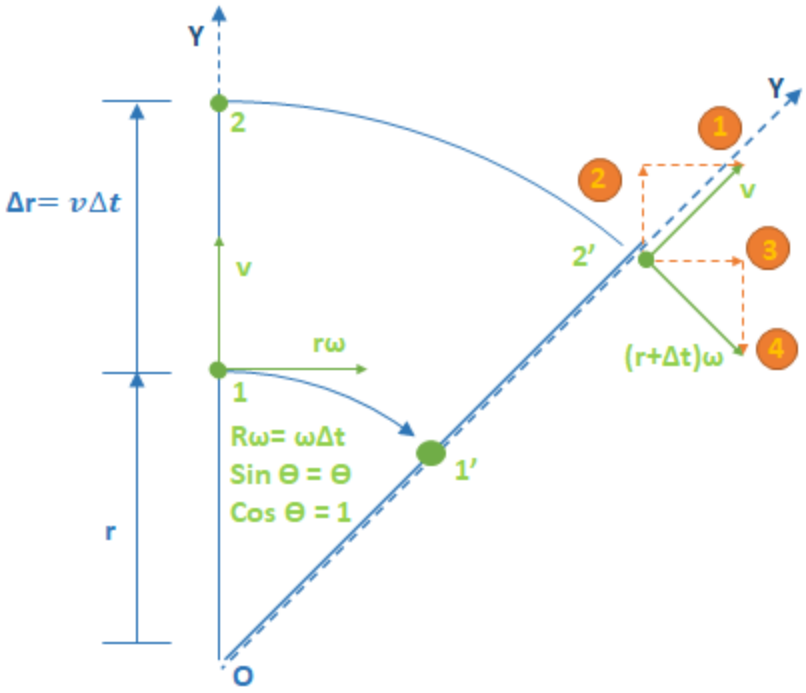



Figure 1.18: Free Body Diagram of "fictitious" forces [23]

The absolute acceleration is the difference between the absolute velocity at points 2' and 1 divided by Δt, and the limit as Δt → 0. If we consider the Y-direction acceleration: $\Delta V_y =$ [component 2 + component 4] – v, diving by Δt → 0. The resultant is $a_y = -r\omega^2$, this is called the centripetal acceleration and is due solely to component 4. As it can be seen the velocity (v) has no effect on the centripetal acceleration, as it only depends on position (r) and angular speed (ω). Next, if we consider the X-direction acceleration: $\Delta V_x =$ [component 1 + component 3] $-r\omega$, diving by Δt → 0; the resultant is $a_x = 2\omega v$. This is called the Coriolis acceleration, and is independent of position. This 2 forces represent the sum of two identical contributions: The effect of ω changing the orientation of v, is exactly the same as the effect of v carrying rω to a different radius changing its magnitude [23].

MEMS gyroscopes rely on vibrating elements, similar to the accelerometers. As shown in Figure 1.19; A mass is moving in a particular direction with a particular velocity (yellow arrow), when suddenly an external angular rate is applied (green arrow), causing a force to occur (red arrow). This will create a perpendicular displacement of the mass, and this displacement will cause a change in capacitance. When this change in capacitance is measured and processed it will correspond to a particular angular rate [24].

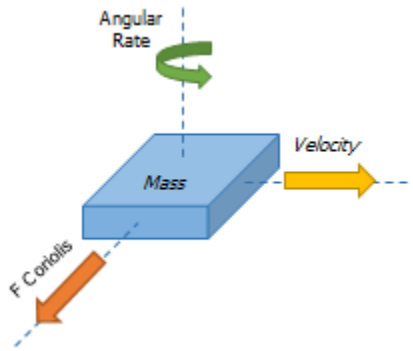



Figure 1.19: Coriolis effect on gyroscope [24]

Figure 1.20 shows the micro-structure of the gyroscope; a mass will be constantly oscillating (in the driving direction), and when an external angular rate is applied to the gyro the flexible part of the mass will move causing the perpendicular displacement (which will cause the change in capacitance) [24].

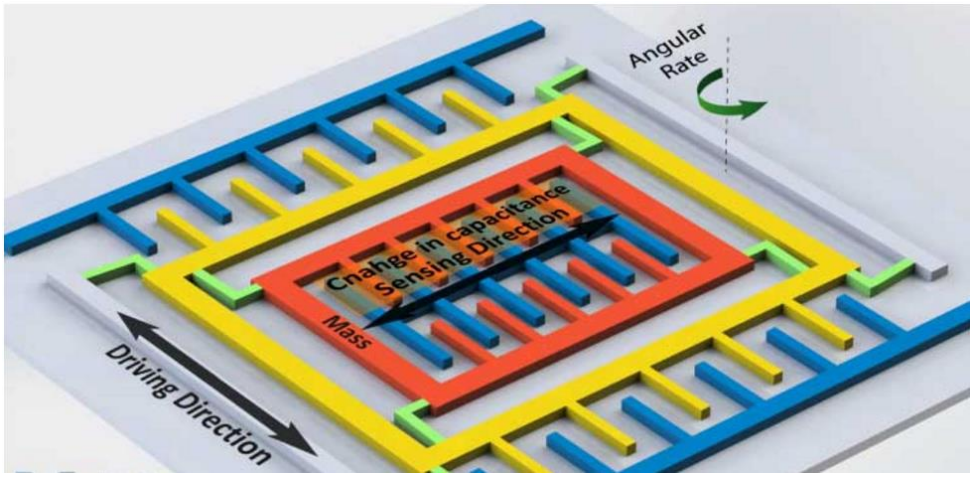



Figure 1.20: Micro-structure of MEMS gyroscope [24]

Now that we have a better understanding of how gyroscopes function, let us go back to the model used in the accelerometer example in order to understand how gyroscope measure in real life. On Figure 1.21 we will have a 2-axes gyroscope that will measure rotation about the X and Y axes.

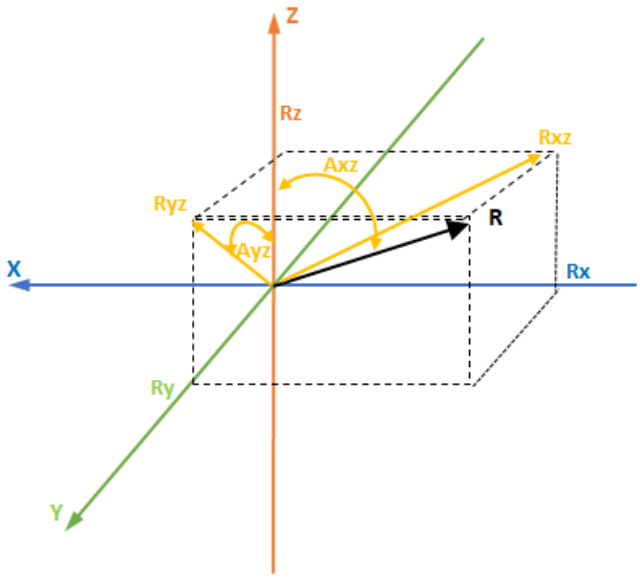



Figure 1.21: Gyroscope angles between axes and force vector [22]

Each of the gyroscope channels will measure the rotation around its respective axis, let us define this projections more clearly:

Rxz: The projection of the inertial force vector R on the XZ plane.

Ryz: The projection of the inertial force vector R on the YZ plane.

There is a right-angle triangle formed by Rxz and Rz, using Pythagorean Theorem we get:

$Rxz^2 = Rx^2 + Rz^2$, similarly:

$Ryz^2 = Ry^2 + Rz^2$ , also note that:

$R^2 = Rxz^2 + Ry^2 = Ryz^2 + Rx^2$

While these equations will not be used for the calculations, it is none the less useful to note the relation between all values in the model.

Now let us go back the angles between the Z axis; Rxz and Ryz. Axz: Is the angle between the Rxz (projection of R on the XZ plane) and Z axis. Ayz: Is the angle between the Ryz (projection of R on the YZ plane) and Z axis.

As explained before gyroscopes measure the rate of change of the angles (defined above as Axz and Ayz), therefore this is a close to what a real gyroscope will measure. Gyroscope will output a value that is linearly related to the rate of change of these angles. This can be easily explained with an example; assuming that we have measured the rotation angle around the Y axis (Axz angle) at time T0 and it will be defined as Axz0. After some time has passed we will measure this angle again at time T1, the angle will now be defined as Axz1. In order to calculate the rate of change the following formula will be used:

RateAxz $= \frac{(Axz0 - Axz0)}{(t1 - t0)}$ , if Axz is in degrees and time in seconds the value obtained will be expressed in degrees/seconds (°/s). This is what a gyroscope will measure in real life.

However, as with the accelerometers, gyroscopes will rarely give a value output expressed in degrees/seconds. An ADC value will have to be converted using a similar formula for the accelerometers. For this example a 10-bit ADC module will be assumed (for 8-bit ADC replace 1023 with 255, 12-bit ADC 4095 for 1023, and so on).

$$\text{RateAxz} = \frac{(\frac{(\text{AdcGyroXZ})(\text{Vref})}{1023} - \text{VzeroRate})}{Sensitivity}$$

$$\text{RateAyz} = \frac{(\frac{(\text{AdcGyroYZ})(\text{Vref})}{1023} - \text{VzeroRate})}{Sensitivity}$$

*AdcGyroXZ, AdcGyroYZ*; similar to the accelerometer example, values obtained from the ADC module, and they represent the channels that measure the rotation of projection of R vector in the XZ and YZ planes. In other words the rotation that was done around Y and Z axes respectively.

*Vref*; is the ADC reference voltage, 5V will be used for this example (this value can be found on the data sheet information).

*VzeroRate*; is the zero-rate voltage found on the data sheet, in other words is the voltage that the gyroscope will output when it is not subjected to any rotation. In this example this value

will be 1.50V (this value can be found on the data sheet,) most gyroscopes will suffer a slight offset after being soldered and these values can be measured using a voltmeter. If this value varies, a calibration routine can be used to correct this. The user must keep the device in a still position to allow the gyroscope to calibrate, this will be touched in more detail in chapter 2.

*Sensitivity*; this is the sensitivity of the gyroscope and it is found on the datasheet (the accelerometer's sensitivity is also found on the datasheet), it is expressed in $\frac{mV}{degrees/seconds}$. In this example the sensitivity will be $\frac{2mV}{degrees/seconds}$.

For this example we will pretend we have a gyroscope attached to an object, that object is then rotated and the ADC module will output the following values:

AdcGyroXZ = 350

AdcGyroYZ = 230

Using the formula stated above, we can find the rate of the gyroscope angles:

RateAxz = $\frac{(\frac{(590)(5V)}{1023} - 1.50\ V)}{\frac{0.002V}{degrees/seconds}} =$ 105 deg/s

RateAyz = $\frac{(\frac{(230)(5V)}{1023} - 1.50\ V)}{\frac{0.002V}{degrees/seconds}} =$ -188 deg/s

This means that the device rotates around the Y axis (or XZ plane) with a speed of 105 deg/s and around the X axis (or YZ plane) with a speed of -188 deg/s. As it can be seen the rotation about the X axis is negative, this means that the device rotates in the opposite direction from the conventional positive direction (clockwise and counterclockwise). The datasheet typically will specify which direction is positive, if this is not shown it will have to be found by experimentation with the device. This experimentation can be best done by using an oscilloscope because as soon as the rotation is stopped the voltage will drop back to the zero-rate level. If the value measured is greater than the zero-rate voltage it means that the direction of rotation is positive.

Now that the gyroscope and accelerometers are more clearly explained, the next step will be to combine them; after all the IMU combines several sensors into one package. The combination of these sensors will be explored in chapter 2, where it will be shown that by combining the outputs

from the sensors a more accurate output will be delivered than each of them individually. This is commonly called filtering. As of now we will continue exploring other possible sensors that can be implemented into the inertial measurement unit.

### 1.3.3 Magnetometers and barometers: The other sensors in IMUs

As mentioned before an IMU is primarily composed of an accelerometer and gyroscope; however there are several more sensors that an inertial measurement unit can be composed of. The numbers of sensors directly impacts its DOF (remember that every additional sensor and its corresponding axis of measurements counts as an additional degree of freedom). This allows for the possibility of measuring additional signals like pressure or magnetism. More notably it allows for a combination of several measurements, thus allowing for more complex and robust filtering opportunities.

Let us start with the magnetometer, this is an instrument that measures magnetism, be it from a magnetic material or from a magnetic field. A known example of the magnetometer is a compass. Magnetometers measure the earth magnetic field by using the Hall Effect (invented by Carl Friedrich Gauss) or Magneto Resistive Effect [24]. Almost 90% of the sensors on the market uses the Hall Effect to measure the magnetic field. Figure 1.22 will help us understand the Hall Effect mechanism.

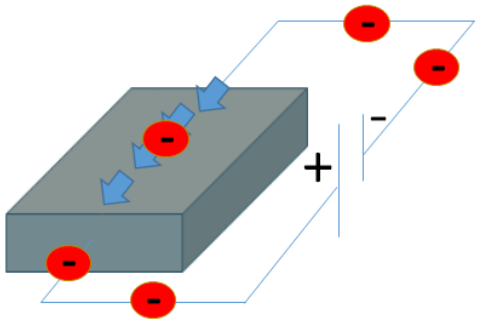


Figure 1.22: Hall Effect on magnetometer [24]

If we have a conductive plate like in Figure 1.22 and a current is flowing through it, the electrons would flow through from one side of the plate to the other. If we were then to introduce a magnetic field (Figure 1.23) near that plate the flow would be disturbed and the electrons would deflect to one side of the plate and the protons to the other side of the plate.

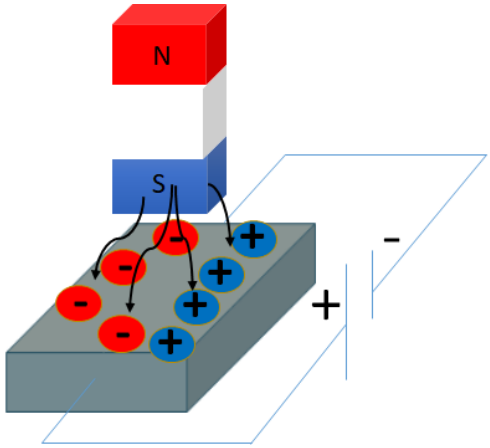



Figure 1.23: Hall Effect on magnetometer with magnetic field [24]

This would allow us to measure the voltage, which in turn would correspond to the magnetic field strength and its direction.

The other 10% of the sensors on the market utilize the Magneto-resistive Effect. These sensors use a material that is sensitive to magnetic field, such as Iron and Nickel, thus when these materials are exposed to magnetic fields they change their resistance [24]. Like the accelerometer and gyroscope the magnetometer can be calibrated, more on this will be explained in chapter 2.

The barometer is an instrument used to measure the atmospheric pressure [25]. They are mostly used on inertial measurement units as pressure altimeters. Pressure altimeters are barometers that can be transported to heights in order to match the atmospheric pressure to a corresponding altitude. Barometers allow the IMU an additional DOF and are essential for aircraft navigation. In addition, barometers are more often more accurate than a GPS for measuring altitude. While the barometer is not as significant as the other 3 previously mentioned sensors, it can also add a frame of reference that can help improve data measurement via filtering.

## 1.4 OBJECTIVES

There is a lot of data and literature available on MEMS IMUs, however this data is often dispersed or written in an overly complicated manner. The objectives of this thesis are:

1. To write a clear beginner's guide to the world of IMUs. This guide includes the means to identify the important factors of the inertial measurement units, in order to pick the right IMU for the application with its uses and limitations. Explains the functionality of the sensors that compose the IMU. Includes useful equations of the sensors and implementation to Arduino. And finally, it contains calibration methods to improve the functionality of the sensors.
2. To analyze and explain noise factors, methods to identify them and decrease them. An overview of the errors that compose the IMUS, methods to detect them, and finally filtering to mitigate the propagation of the errors will be assessed.
3. To create simulation models of different filters available for the MEMS IMU. Different filter models will be created to simulate the increased performance of the sensors.
4. To assess the performance improvement on the MEMS IMU with the different filters. Data set will be collected in order to assess the performance of the filters using the simulation models.
5. To create educational laboratory workshops. Workshops that can be implemented on a classroom environment will be presented, these will challenge students on the concepts of the IMUs.

## 1.5 RESEARCH METHODOLOGY SUMMARY

The following methodology has been carried out to meet the objectives outlined in the previous section:

1. Error characterization of MEMS IMU: This is focused on examining the error sources (bias and scale factor) and obtaining the noise factors present in the MEMS sensors.

2. Implementation of (error compensation) filters to reduce errors and bias: This step involved the development and implementation of different algorithms to error propagation of the MEMS output data.
3. Software development: All the integration strategies, filters and error compensation algorithms are implemented in MATLAB. The software is a post-processing package, although the implemented algorithms are adaptable to real-time applications.
4. Implementation of educational workshops: This step is dedicating to in the development of several workshops that will work as a base for students.
5. Field test and data analysis: The final step was to undertake field tests and data processing. Field testing was conducted with different IMUs to provide several performance results of the error compensation filters.

## 1.6 OUTLINE

Chapter Two provides an overview of error characterization. This includes a description of filtering, identification of noise factors, and methods to locate the source of these errors. Finally, it provides a brief overview of calibration techniques for several of the sensors that compose the IMU.

Chapter Three provides an overview of Arduino implementation, and discusses the combination of multiple sensors for filtering. It also provides an overview of different integration strategies and approaches using the combination of sensors for error compensation.

Chapter Four provides a brief overview on graphical display options for the sensors. It then delivers the simulation models of the different filtering options presented in chapter three.

Chapter Five provides the description of five educational workshops, and recommendations for future workshops that could be implemented in a classroom environment.

Chapter Six summarizes the work presented in this thesis, and draws conclusions from the test results and analysis. Finally, recommendations for future work are outlined.

# CHAPTER 2: NOISE FACTOR AND BIAS

The first thing that can be noticed when connecting an accelerometer and taking readings is the amount of noise present. In signal processing, noise is an undesired deviation of the actual output of a sensor. Figure 2.1 will show this more clearly.

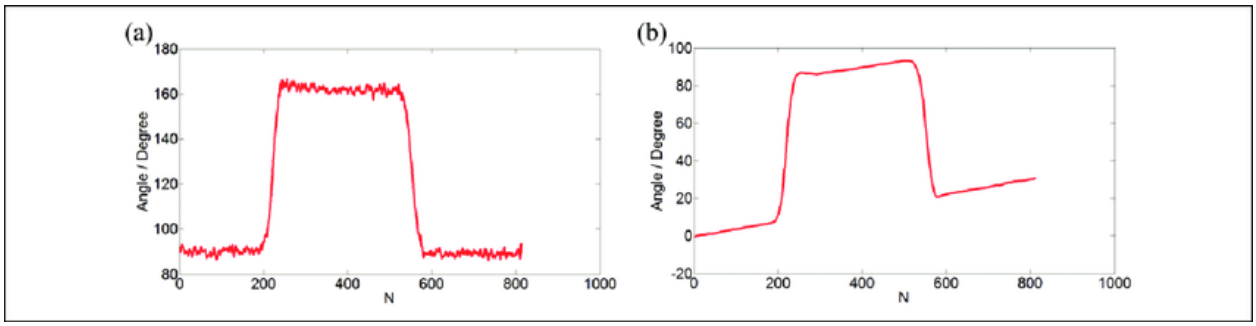



Figure 2.1: Noise in a signal output from gyroscope [26]

On the left size of the figure we can see the noise as spikes on the output data, while the right size shows a clean signal output in a straight pattern. Going back to chapter 1, it was noted that an accelerometer can provide accurate orientation angles as long as gravity is the only force acting on the sensor. However when moving and rotating the sensor, forces will be applied to it causing measurements to fluctuate. In other words accelerometer data tends to be extremely noisy, with brief but significant perturbations. If these perturbations can be averaged out, the accelerometer provides accurate results over timescales longer than the perturbations. In order to demonstrate this, data will be recorded on the accelerometer. The accelerometer will remain stable on a table and a gently tap will be applied. Figure 2.2 will show the perturbations caused by tapping the table. These perturbations are caused by a change in force, which are created due to introducing an external force besides gravitational to the accelerometer and can also be referred to as white noise.

Computing the orientation from the gyroscope is different, as explained in chapter 1 gyroscopes measure angular velocity; that is the rate of change in orientation angle. Thus when the sensor is first initialized a known value for the position must be given. From then it measures

the angular velocity around the X,Y, and Z axes at measured intervals Δt. With the formula stated previously (ω) (Δt) = change in angle, we can get our value. The new orientation angle will be the original angle plus the change measured.

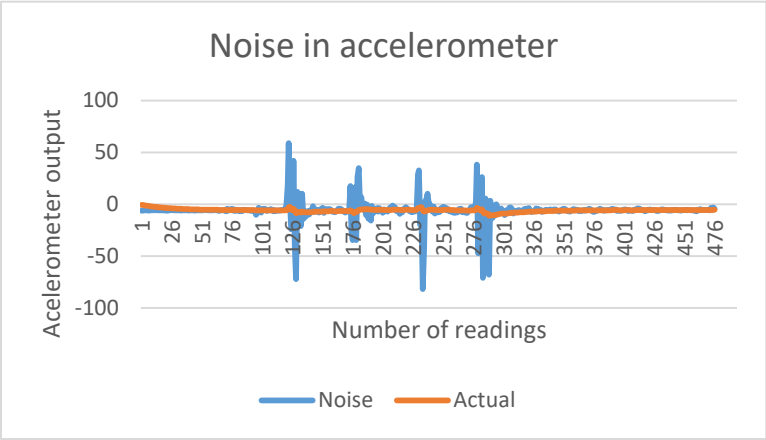



Figure 2.2: Perturbation on accelerometer

The problem then becomes obvious, the orientation angle is being integrated; thus small errors are being added. These small errors will be added continuously, which will become magnified over time, this is known as gyroscope drift. The gyroscope drift will cause gyroscope data to become increasingly inaccurate.

In order to demonstrate gyroscope drift, data will be recorded by leaving the sensor completely still on a table for a period of time. As time passes, the data will begin to drift further and further without moving the sensor at all.

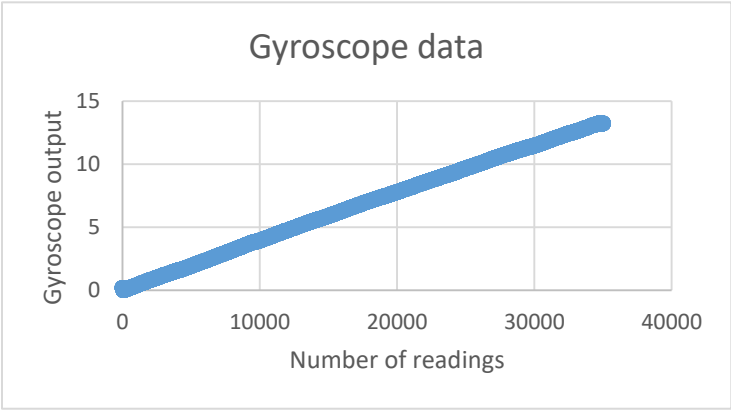



Figure 2.3: Drift on gyroscope

Since the gyroscope is left still on a table any value that deviates from the true position is called bias. Bias is the difference between a measured value of a quantity and its true value. These errors and bias are common in all of the components of the IMU, for this reason it is very important to be able to correctly identify them. Additionally it is essential to use the proper filter to limit their effect on the output signals [27].

## 2.1 IDENTIFYING NOISE FACTORS

Figure 2.1 and 2.2 show a clear example of the short-comings of these sensors; its propensity to noise and errors. Of course the error and bias acceptance (as always), depends on the application. IMUs (and its sensors) can typically be separated into three main classes depending on their performance requirements. The first category is *motion detection*; this is usually hand gestures and pedometers. A clear example of motion detection is the Motorola cellphones with its hand gestures.

The next category is *attitude measurement*; usual examples are flight control, stability, and robotic arms. The final and more demanding category is *inertial navigation;* which we have already covered in chapter 1. Each one requires a full order of magnitude (or better) of performance out of the sensor.

In the previous example the sensor was left static on a table, which shows of one of the easiest applications these sensors can perform; motion detection. On this example all we care about is whether the sensor is moving or not. This means that since we know that actual position of the sensor it would be a simple matter of filtering out all of the bias (incorrect data) from the output. In this case all we care about is to look for deviations from a steady-state value of the sensor. This certainly could be achieved with either a high or low pass filter (this will be further discussed in the next section).

The difficulty comes when the sensors are used for non-static applications such as inertial navigation. The reason is that the acceleration would have to be integrated twice to get an estimate position. Therefore any bias will quadratically increase error with time and these noises will turn

into a double integrated error. The main problem is that in a non-static application the ability to average data is not possible like in a static one, additionally there is more than just white noise acting on the devices.

It is important to understand that most of these other noise factors are only important when the application requires high precision, like for attitude measurement or inertial navigation. As previously mentioned in non-static applications, in addition to white noise, there are several other types of noises such as: Turn-on bias, Temperature bias, Bias Instability (Gyro Drift), Bias Random Walk, and Scale Factor error. All of these noises will be further explored in the upcoming subchapter. *Turn-on bias;* it is a constant offset that changes every time that the device is turned on.

*Temperature bias;* it is a change in the bias of these sensors as a function of temperature. This is caused because MEMS components are silicon based, and such temperature will expand or contract the structures inside the devices. *Bias instability;* it is basically the gyro drift previously explained. *Random walk bias*; it is the random fluctuations to the bias, but this follows a random process. The result is that after averaging for a period of time, the standard deviation of its average increases because of the slow fluctuations in the bias. In order to detect this biases a tool known as Allan variance chart is used to analyze the bias performance of the sensors, which we will explore in the next subchapter. As mentioned in chapter 1 some datasheets have information regarding the standard errors, however in some cases there will need to be a lot of experimenting before working out if a device is suitable for the required application [28].

### 2.1.1 Allan Variance

As mentioned before Allan variance is a method of analyzing the bias and error. It can analyze a sequence of data in the time domain, in this case the output data. Allan variance is a method that is simple to compute and understand, for this reasons it is one of the most popular methods for identifying and quantifying the noise in the inertial sensor data. The results from using

this method are related to the noises described earlier, these are quantization noise, angle random walk, bias instability, rate random walk, and rate ramp [29].

Allan variance uses a time domain signal Ω (t). The process for the Allan variance consists of computing the root Allan variance (also called Allan deviation) as a function of different averaging times (τ). It then analyzes the characteristic regions and log scale slopes of the Allan deviation curves in order to identify the different noise modes. One of the most useful uses for the Allan deviation plot is for noise identification in a gyroscope. We will first walk through the steps that need to be followed in order to create an Allan deviation plot.

The overlapping Allan variance method is used for computing the Allan variance and creating the Allan deviation plot for the noise analysis [30]. The steps go as follows [29]:

1. The first step is to acquire a time history Ω (t) of the gyroscope's output, it's important to do this while the gyroscope is static using an experimental setup. When a certain amount of time has passed, take the data and let the number of samples be N and the sample period be $\tau_0$ [29].
2. The next step is to set the averaging time as $\tau = m\tau_0$, where m is the averaging factor. This value can be chosen arbitrarily as long as it meets the following: $m < \frac{(N-1)}{2}$ [29].
3. The following step is to divide the time history of the signal, this is the sequence of gyroscope output data over time into clusters of finite time duration of $\tau = m\tau_0$ .

After these steps of overlapped Allan variance the time stride between two consecutive data clusters is always equal to the sample period $\tau_0$. This allows to form all possible overlapping

sample clusters with the averaging time τ, thus making maximum use of the dataset. This can be seen on Figure 2.4 [29].

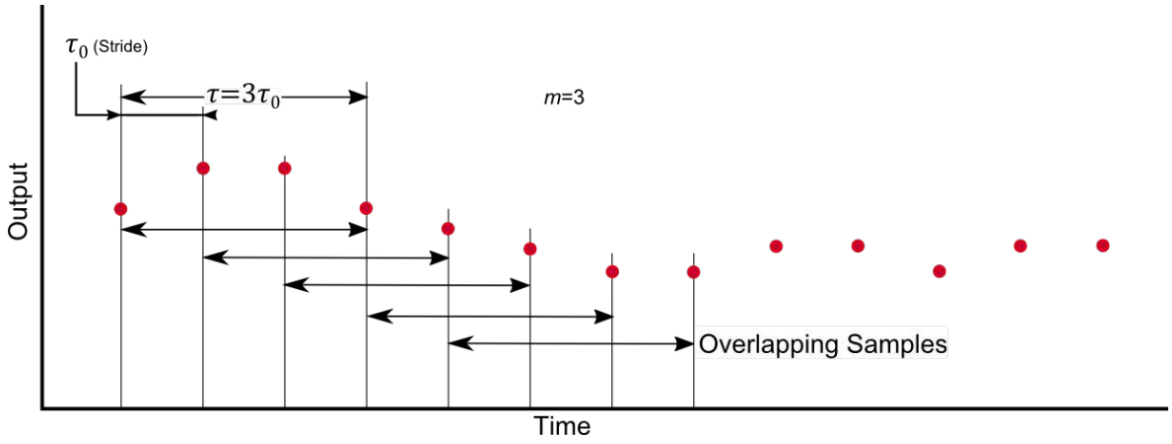



Figure 2.4: Overlapping samples using overlapped Allan variance [29]

As it can be seen on Figure 2.4 the overlapping Averaging factor m is 3. The m value follows the rule: $(m < \frac{(N-1)}{2})$ where N = 8, thus m = 3 <3.5, which is true. It can also be observed that the clusters have overlapping samples (i.e. they share samples between them). Additionally the cluster has a duration equal to $\tau = 3\tau_0$. And each cluster is separated by the sample period $\tau_0$ which is the Stride [30].

4. After the clusters are formed, the Allan variance can be computed (using overlapping method):
    - Averages of the output rate samples (over each sample).
    - Output angles Θ corresponding to each gyro sample.
5. The last step is to calculate the Allan deviation value for a particular value of τ, and then obtain the Allan deviation plot by repeating the steps for multiple values of τ.

Now, different ways to calculate the Allan variance will be shown [29].

<u>Method 1: Calculating Allan variance using output rate angles:</u>

1. First calculate the Θ corresponding to each of the gyroscope's output sample. This is done by using the following equation: $\Theta(t) = \int^t \Omega(t')\, dt'$.

The angle measurements are made at times t = $k_{\tau_0}(\tau_0, 2\tau_0, 3\,\tau_0 \ldots)$ where k varies from 1 to N. For a discrete set of samples, a cumulative sum can also be used to give N values of Θ. In this case the cumulative sum of the gyro output samples at each $k_{\tau_0}$ is taken and each sum obtained is then multiplied by a sample period $\tau_0$ in order to give N values of Θ. This may look a little bit confusing so an example will be used to better explain the concept [29].

**Example:** We are asked to calculate three values of Θ, assuming k to vary from one to three samples. We will then assume the observed values of $\Omega_k$ as 20, 22, and 25. If the cumulative sum is used the following values are obtained: 20, 20 + 22 =42 and 20 + 22 + 25 = 67. After this it will be multiplied by $\tau_0$, the corresponding values of $\Theta_k$ can also be obtained: $\Theta_1 = 20_{\tau_0}$ , $\Theta_2 = 42_{\tau_0}$ , $\Theta_3 = 67_{\tau_0}$ [29].

2. After the N values of Θ have been computed, the Allan variance can be calculated using the following equation:

   $\sigma^2\,(\tau) = \frac{1}{2\tau^2} < (\Theta_{k+2m} - 2\Theta_{k+m} + \Theta_k)^2 >$

   $\sigma^2\,(\tau)$ represents the Allan variance as a function of $\tau$, and < > is the ensemble average. If the ensemble average is expanded the following equation can be obtained:

   $\sigma^2\,(\tau) = \frac{1}{2\tau^2(N-2m)} \sum_{k=1}^{N-2m} (\Theta_{k+2m} - 2\Theta_{k+m} + \Theta_k)^2$

   N is the total number of samples, m is the averaging factor, $\tau = m\tau_0$ is the averaging time, and finally k is a set of discrete values varying from 1 to N [29].

This equation computes the final rate Allan variance (by overlapping method) value using the output angle Θ for a gyroscope, for one particular value of τ. Once the output angles values of Θ are known, along with the sample period and the value of m, we can use this equation to compute Allan variance [30].

Method 2: Calculating Allan variance using averages output rate samples:

1. The first step is to average the output rate samples across each cluster between times $k_{\tau_0}$ and $k_{\tau_0} + \tau$. The following equation is obtained:
$$\bar{\Omega}_k(\tau) = \frac{1}{\tau}\int_{k_{\tau_0}}^{k_{\tau_0+\tau}} \Omega(t)\, dt$$
Using the previous method to calculate the N as values of $\Theta$, the average output of $\bar{\Omega}_k(\tau)$ can be calculated using the output angle. The average output rate in terms of the output angle $\Theta_k$ between the times $k_{\tau_0}$ and $k_{\tau_0+\tau}$ can be given as the following:
$$\bar{\Omega}_k(\tau) = \frac{\Theta_{k+m} - \Theta_k}{\tau}$$
An example will be shown to make this easier to comprehend [29].

**Example:** Calculate $\bar{\Omega}_k(\tau)$ with k = 2 and m = 3, assuming $\tau = 3_{\tau_0}$. Denote the average output of the gyroscope over the cluster between times $2_{\tau_0}$ and $2_{\tau_0} + 3_{\tau_0} = 5_{\tau_0}$, using the previous equation this would be equal to:
$$\bar{\Omega}_2(\tau) = \frac{\Theta_5 - \Theta_2}{3_{\tau_0}}$$

Using the equation above it is possible to obtain the values of $\bar{\Omega}(\tau)$ (over each cluster) and calculate them for a particular value of $\tau$. Since there are N = m possible clusters formed, we can find all possible values of $\bar{\Omega}(\tau)$. Thus since all possible values are now known, the Allan variance can be computed using the following formula:
$$\sigma^2(\tau) = \frac{1}{2} < (\bar{\Omega}_{k+m}(\tau) - \bar{\Omega}_k(\tau))^2 >$$
This is the definition of Allan variance. $\sigma^2(\tau)$ represents Allan variance as a function of $\tau$ and $< >$ is the ensemble average. Expanding this equation we can get the following:
$$\sigma^2(\tau) = \frac{1}{2m^2(N-2m)} \sum_{j=1}^{N-2m}\left\{\sum_{i=k}^{j+m-1}(\bar{\Omega}_{k+m}(\tau) - \bar{\Omega}_k(\tau))^2\right\}$$
This equation represents the final Allan variance of a gyroscope for a particular value of $\tau$ by using the averages of output rate value, $\bar{\Omega}(\tau)$ across each cluster. When using the overlapping method for computing the Allan variance any of these 2 methods can be used in order to calculate N – 2m Allan variances. The next step is to calculate the Allan deviation and create an Allan deviation plot to be able to identify the noise factors [29].

Calculate Allan deviation and create Allan deviation plot:

The final step is to take the square root of the result obtained from method 1 or method 2 in order to obtain the value of the root Allan variance or the Allan deviation for a particular value of $\tau$. The result will be used to characterize the noise in a gyroscope. The following formula will be used: AD $(\tau) = \sqrt{AVAR(\tau)}$. The AVAR term is commonly used for the overlapping method of Allan variance obtained from the equations previously discussed. As mentioned previously the range of $\tau$ values to obtain a complete Allan variance curve can be chosen arbitrarily. The Allan deviation plot is normally plotted as values of Allan deviation over $\tau$ on a log vs log plot [29].

Noise identification:

The different noise factors that were mentioned in the previous sub chapter will cause different slopes with different gradients to appear on the Allan deviation plot. The different processes usually appear in different regions of $\tau$, this allows their presence to be easily identified. Once the process has been identified it is possible to read its numerical parameters directly from the plot. In the case of the gyroscope the random walk and bias instability can be identified and read [29] [30].

*White noise or Random Walk*; this type of noise factor appears on the Allan variance plot as a slope with a gradient of -0.5 and is contributed by random fluctuations in signal with correlation time much shorter than the sample rate. The random walk measurement for this noise is called Angle Random Walk (ARW) for a gyroscope and Velocity Random Walk (VRW) for an accelerometer. It is obtained by fitting a straight line through the slope and reading its value at $\tau$=1. ARW is given in units of dps/rt (Hz), and for VRW $\frac{m}{\frac{s^2}{\sqrt{Hz}}}$ [29] [31].

*Bias instability*; this type of noise factor appears on the plot as a flat region around the minimum. Although graphically similar, it should not be confused with bias repeatability or turn-on bias. The bias instability indicates the minimum bias that cannot be estimated and it measures

how much the bias changes over a specified period of time at a constant temperature. The units for this type of bias are dps/sec for a gyroscope and $\frac{m}{s^2}$ for the accelerometer [29] [31].

*Rate random walk*; this type of noise factor is characterized by a power spectral density that falls off as $\frac{1}{frequency^2}$ and represents bias fluctuations caused in the long term due to temperature effects. As it can be seen on Figure 2.5 the rate random walk is basically the inverse of the angle random walk and has a slope with a gradient of 0.5 [31].

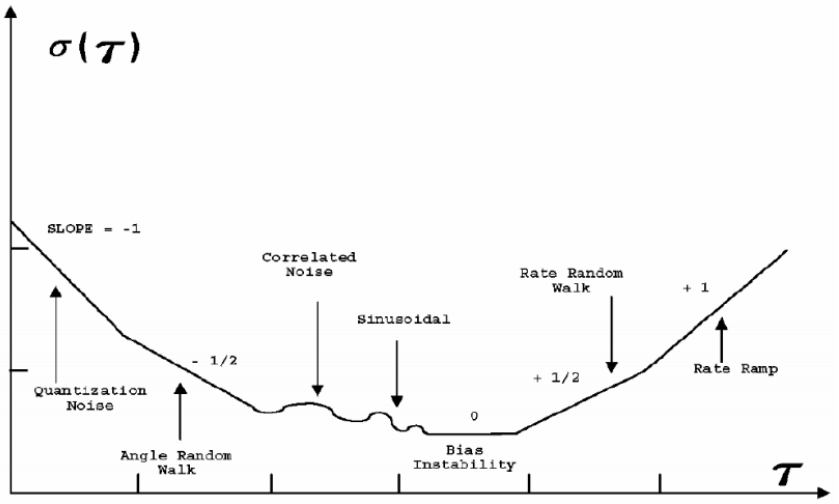



Figure 2.5: Overlapping samples using overlapped Allan variance [29]

In this figure all of the noise factors can be seen and identified, this will allow the end user to understand the size, the frequency, and the impact a certain bias is affecting the output. There is another method that is very useful in the identification of noise factors it is called interval computation which will be discussed on the next chapter.

## 2.2 FILTERING

So now that we have identified the errors and bias we can get rid of all of them right? Well not exactly, with the currently technology available it is impossible to completely eliminate errors and bias from the sensors. So why filter then? The reason is simple; filters make a vast difference compared to raw output data because they deliver a much cleaner and error reduced signal (more on this will be touched in chapter 3).

We have continuously talked about filters through this and the previous chapter, but how exactly do they work? In order to demonstrate this we will use Figure 2.1 to use an example of 2 very typical an easy to implement filters.

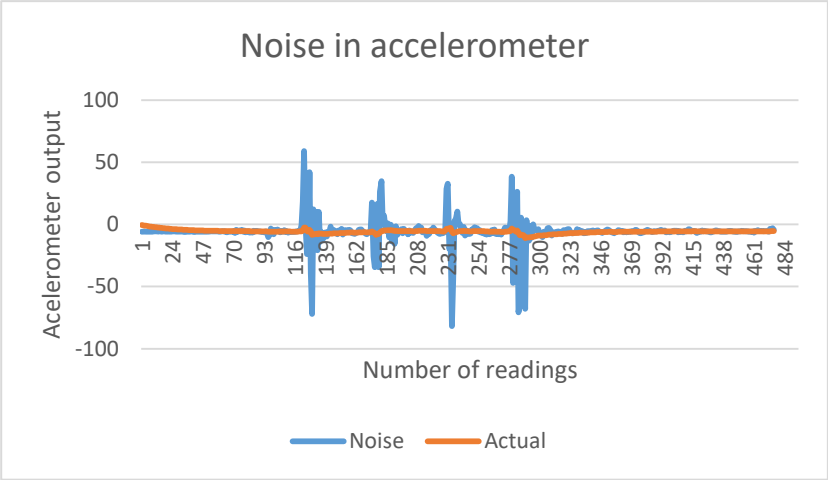



Figure 2.6: Perturbation on accelerometer

As it can be seen on Figure 2.6 there is a lot of noise on the accelerometer data. The accelerometer is standing still on a table and it is receiving small vibrations. For the application that is being used those vibrations are not important so they can be filtered out using a low-pass filter. A low- pass filter is a filter that only allows signals with lower frequencies to pass (this is set by the user using a cutoff frequency). Any frequency that is above the value of the cutoff frequency will be eliminated [32].

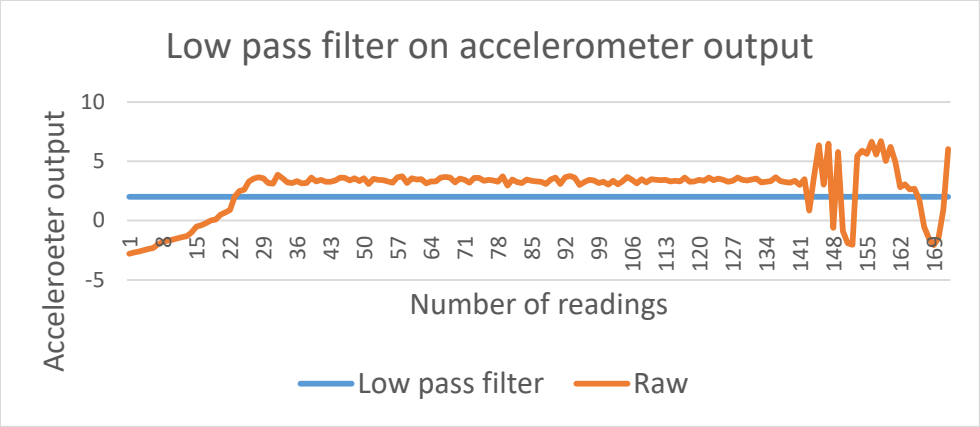



Figure 2.7: Low pass filter

Figure 2.7 shows accelerometer data using a low pass filter at 2g, in this case all of the values above 2g will be eliminated from the output. A low pass filter is useful in applications where high frequencies are not important or when they happen in a very limited capacity that they do not affect the system in a significant matter, thus they can be ignored.

On the other hand, there is the high pass filter. As the name implies this filter works the opposite way that a low pass filter does, it only allows signals with a higher value than its cutoff frequency and eliminates all frequencies that fall below it [32]. Figure 2.8 will show an example of a high pass filter.

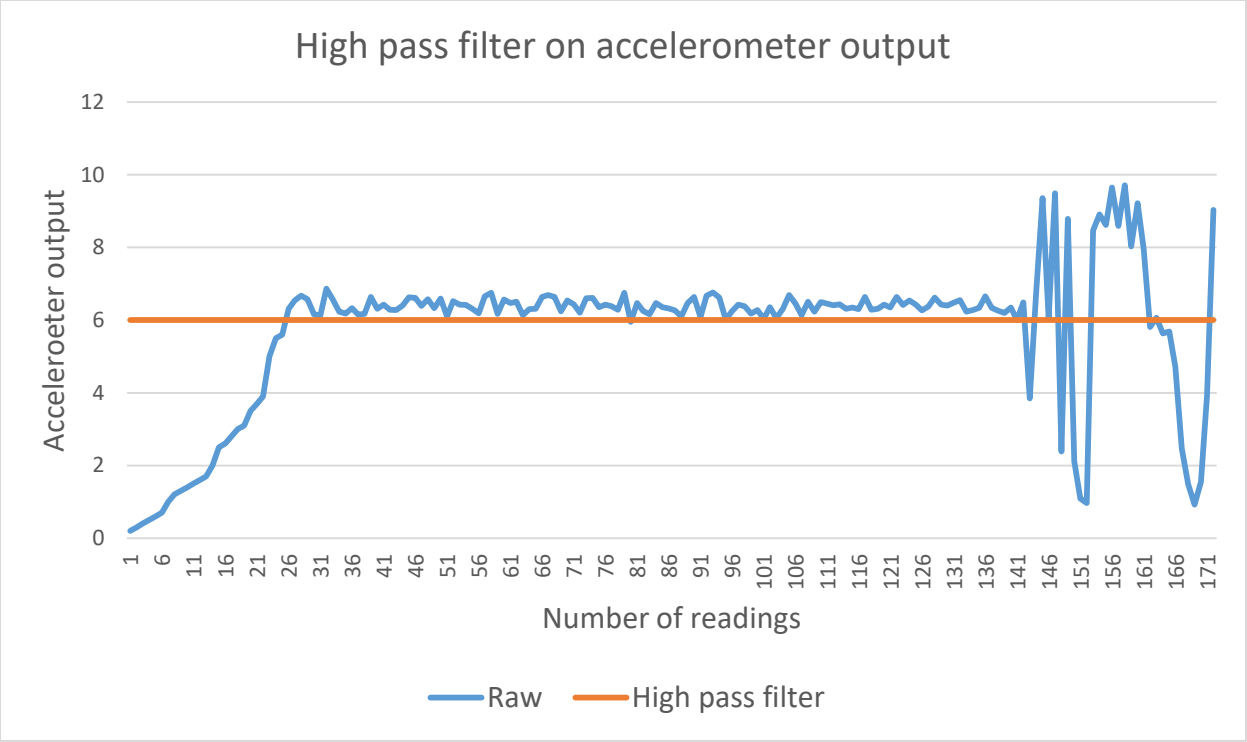



Figure 2.8: High pass filter

Figure 2.8 shows accelerometer data using a high pass filter at 6g, in this case all of the values below 6g will be eliminated from the output. A high pass filter is useful in applications where high frequencies are so frequent that low frequencies will not affect the system, thus they do not need to be taken into account. An example of this application is when the system is subjected to a high vibration profile, thus the system is required to survive the highest frequency. As it can be seen from this cases, filtering is used to eliminate signal outputs that are considered an error, bias, or data that is not relevant to the application.

As mentioned before, turn-on bias is a constant offset that changes every time that the device is turned off and on. A type of simple "filter" that can be used to mitigate this is to use a calibration routine. A calibration routine allows the sensor to account for this turn-on bias and provides a better offset of the sensor. In the following section some calibration methods and information will be mentioned for the accelerometer and gyroscope components of the IMU.

### 2.2.1 Calibration techniques

Error and bias reduction can be vital in certain applications. These errors directly influence heading estimates in navigation systems, along with the accuracy of a platform control system that uses them for feedback. An ideal sensor will produce a predictable output when subjected to a known force or rotation. As mentioned before, calibrations are simple techniques that narrow and reduce the error distribution of the sensor. They allow for a more predictable output and in a more reliable sensor. While these techniques can be applied to most of the sensors on the inertial measurement unit, this thesis will focus on its two main components the gyroscope and accelerometer.

### *2.2.2 Calibration for gyroscopes*

Although an off-the-shelf gyroscope may satisfy some of the requirements of a desired application, it may end up being too inaccurate. A gyroscope calibration offers an easy and affordable solution for a degree of precision required. The purpose of doing a calibration is to translate transducer behaviors into valuable bits at the system level, as a good calibration gives predictable outputs over the important conditions for the system. When translating MEMS-gyroscopes behaviors into predictable behavior, it requires the user to have an advance understanding of the transducer's performance in order to impact the critical system performance and allow for corrective actions. Although this calibration will not replace the need for more advance, precise, and more expensive devices; it will however speed development cycles and lower up-front investment requirements. An ideal MEMS gyroscope produces a predictable output when it is subjected to a known rate of rotation. An idealized model has no noise, perfect linearity, and no offsets. While perfection is not obtainable, creating a deeper understanding of gyroscope errors can significantly improve their performance [33].

The following equation provides a simple behavior model for a MEMS gyroscope: $\omega_{Gyroscope}=(K)(\omega_{Rate})\ (\varepsilon) + (\omega_{Bias})\ (\omega_{Noise}) + K_2\ (\omega_{Rate})\ (\omega^2 Rate)(K_3)\ (\omega^3 Rate)+\cdots$

Data sheets typically provide estimates for each of the error terms in this equation, otherwise they can be estimated using the method mentioned in the previous chapter. This equation allows the estimation of the system-level impact and helps establish the desired performance goals [33].

After an understanding of the MEMS gyroscope's behaviors and error patterns, the impact of the system operation must be determined. It is important to set up performance goals as well, as it is essential to successful sensor integration. If the desired application uses the gyroscope output to determine relative angle displacement integrating, bias errors will add fixed errors to the sensor's output response. These errors can make the device look like it is rotating even when it is stationary. The net result is a constant accumulation of angle-measurement error, this accumulation is equal to the bias error and time accumulation. The following equation shows the relationship of this drift factor: $\Psi BIAS = (\int_0^{t^1} \omega_{BIAS})\,(d_t) = (\omega_{BIAS})\,(t^1)$

Scale-factor errors contribute to displacement measurement error only where there is motion. The following equation shows the scale factor error:

$$\Psi Scale = \int_0^{t^1} (\omega_{Rate})\,(K)\,(\mathcal{E})\,(d_t) = (\mathcal{E})\,(\omega_{Rate})\,(t^1)\,(K)\,(d_t)$$

Noise that is inversely proportional to the integration time associated with the measurement contributes error to the measurement output of the gyroscope. As explained before Allan variance can be used for analyzing the impact of the bias estimate with respect to an integration time. For example, the accuracy for a 10 - second integration time can be given as approximately $0.006\ \frac{^\circ}{s}$. As it is expected the bias, noise, scale factors, and linearity have a notable impact on the MEMS gyroscope angle-displacement measurement outputs. These errors directly influence heading estimates in navigation systems, along with accuracy platform control system applications. Figure 2.9 shows the combined errors in a gyroscope output reading [33].

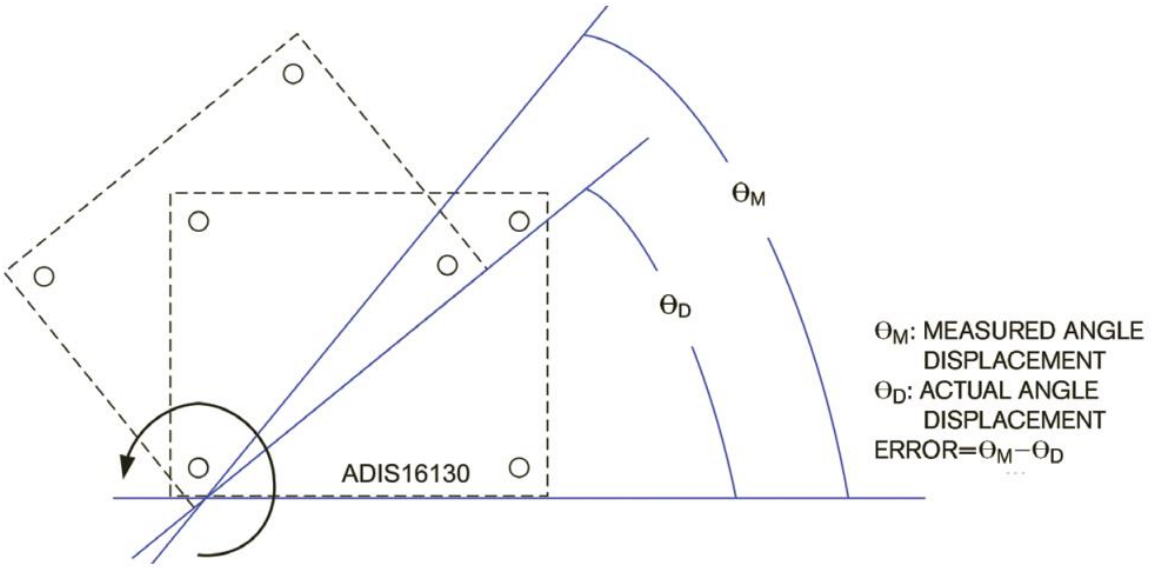



Figure 2.9: Combined error from bias, noise, and scale factors in gyroscope output [33]

As mentioned, the calibration goal is to narrow the error distribution of the transducer population in order to transform the output into a predictable condition. This procedure involves characterizing the transducers under known conditions, this provides the necessary data for the specific correction formulas. This process offers a simple calibration that it's easy to implement and has powerful results. A linear-compensation approach addresses the first-order bias and scale factor errors, it will suffice for reaching less than 1% composite errors. An easy method to estimate bias errors is to average the output of a MEMS gyroscope while holding the device in a constant positon. Using the previously explained Allan variance curve it is possible to analyze the trade-off between test time and bias accuracy. Gyroscopes are typically characterized for scale-factor errors using a servo motor stage with the addition of an optical encoder for precise rate control. This method is very useful because the MEMS gyroscope will rotate at known rates while providing output measurements, however it is expensive and requires a higher investment in equipment. There is a simpler approach for observing scale factor, in this approach the scale factor error is the ratio of the measured angle to the actual angle displacement [33].

The following is a calibration example that is practical to use with a gyroscope, when beginning the calibration the following physical set up is needed [33]:

1. Select a pivot point and secure it with a machine screw assembly. The torque provided should secure the attachment while allowing for smooth rotation on the surface at the same time.
2. Install mechanical stops for two angular positions. The recommendation is to set 2 screws in positions that provide approximately 90° of rotation motion.
3. Pivot the gyroscope between each stop point to make sure the rotation is clear and smooth.
4. Measure the displacement angle using an independent sensor. If the inclination accuracy is at least 0.25° and the rotation span is 90°, the error will be less than 0.3%. An Anglecube was used for this thesis, as it was an accuracy of ± 0.2° (see Figure 2.10).


Figure 2.10: iGaging AngleCube

Once the setup and angle measurement are complete, the following procedure is used to measure the gyroscope's behaviors [33]:

1. Power the gyroscope and allow for thermal stability, if possible implement temperature as an output and refer to the datasheet for more information.

2. Start measuring the output of the gyroscope while holding it against the first stop position.
3. Wait 5 seconds and then turn the gyroscope towards the second stop. This movement should be done using a smooth motion that takes 3 to 4 seconds to move to the 90° span.
4. Wait an additional 5 seconds and then rotate the gyroscope back to the first stop using a similar motion.
5. Wait 5 more seconds and then stop measuring output data, this data should be saved (more on this on chapter 4).

Using the obtained data, use the following steps to calculate the correction factors for the MEMS gyroscope [33]:

1. Calculate the bias offset correction factor by averaging the first 3 seconds of data. The bias correction will be the opposite polarity of this average. For example the bias correction factor will be $\frac{-8.6\,^{\circ}}{sec}$ . This correction yields an accuracy of greater than $\frac{0.1\,^{\circ}}{sec}$ for the bias estimate.
2. Subtract the bias estimate from the time record. Then integrate output data from the 4 second time stamp to the 9 second time stamp. In this case, the measured angle displacement will be assumed to 95.1°. Since we know that the hard stop is 90° we can calculate the scale factor, the scale factor from this step is 90° divided by 95.1 ° resulting in 0.946.
3. Next, using the bias corrected response from step 2, integrate output data from the 12 second time stamp to the 16 second time stamp. In this case we will assume a measured angle displacement of -95.3° (since we are going back to the first hard stop). The scale factor from this step is 90° divided by 95.3 ° resulting in 0.944.

4. The average of the results from step and 3 is calculated in order to obtain the scale factor correction, for our data it will be 0.945. Figure 2.11 will show an example of a gyroscope output using this method.

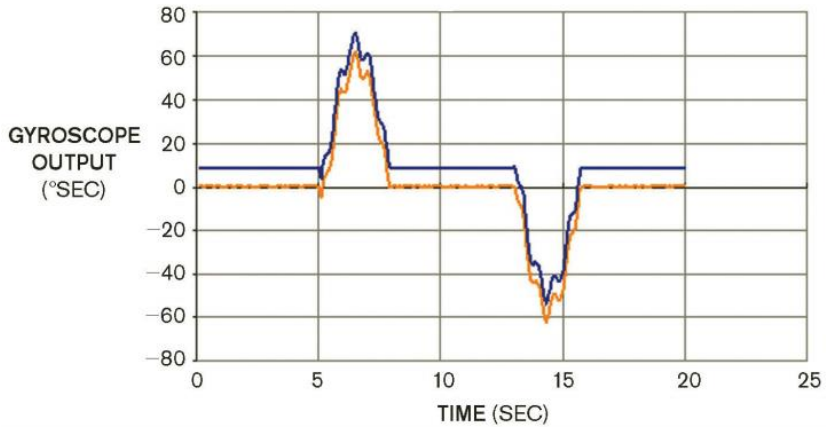



Figure 2.11: Time record of MEMS gyroscope output [33]

This process will obviously require practice and experimentation in order to refine and improve. Experimentation will help refine the speed of rotation, it is important however to make sure the surface is smooth and that the gyroscope does not jump up and down during the rotation. Another area of sensitivity is in the mechanical stops, it is important to make sure the gyroscope does not bounces at a stop as this may introduce errors. Using adequate transition times eases the burden of having accurate time stamps, the important factor is to start integrating before the motion begins and to stop integrating after the motion stops (mechanical stops).

As mentioned in the previous chapters, MEMS gyroscopes are sensitive to acceleration and gravitational force. Therefore, orientation with respect to gravity is worth taking into consideration. Gravity will have the least effect on the gyroscope when the sensor axis of rotation is oriented to be aligned with gravity. In certain applications power-supply and thermal influences will need management as well. For certain applications repeating this process before taking critical measurements will suffice, if that is not the case repeating this process at two power supply levels and two temperatures will help mitigate the first order effects.

Using this gyroscope calibration procedure offers a relatively simple solution and requires little investment in equipment. For some applications this process will provide the necessary accuracy that is required. For more advanced applications that require a higher degree of accuracy, a higher investment will be required in order to have the correct calibration (such as encoders). All of this will have to be evaluated by the engineers in charge of the application and project [33].

### *2.2.3 Calibration for accelerometers*

In the case of accelerometers the calibration concept is very similar. The digital sensor output of the accelerometer is converted to g-force using the following linear equations for high and low gain. These are idealized equations that assume no offset or scaling error. The equations are in the format of y = mx + b, were y is the calculated g-force, m is the scaling factor ($\frac{g}{counts}$), x is the sensor output (counts), and finally b is the y-intercept (or offset) [34]. Figure 2.12 will show the ideal output at high gain (blue line). The red line represents typical output without calibration, which may have both an offset and a slope error. Calibration of the accelerometer will determine the correct scale (m) and the offset (b) factors that makes the red line equal to the blue line [34].

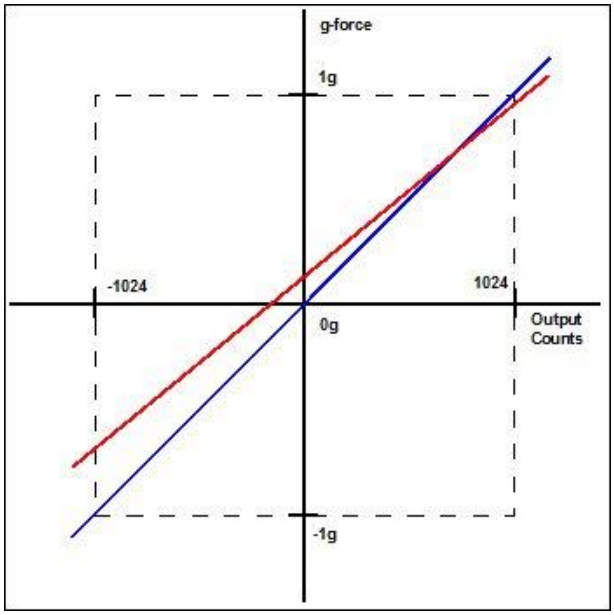



Figure 2.12: Ideal versus typical sensor output [34]

As mentioned before these 2 idealized equations will represent the output of the accelerometer. High gain: $g - force = \frac{1}{1024}\ (counts) + 0$ and Low Gain: $-force = \frac{1}{324}\ (counts) + 0$. Increasing the accuracy of the accelerometers sensor is achieved using a "Tumble" calibration method. This calibration method uses gravity as a reference acceleration. The next step is to place each of the axis of rotation in and out of the gravity vector, this provides a +1g, -1g, and 0g reference points that help establish a scale factor and an offset factor that will tune the linear relationship of the sensor. The following instructions outline the tools and process to perform a 6 point tumble test for a 3-axis accelerometer, which is typical for most modern applications [34].

A flat level must be used in order to properly align the accelerometer sensor. Marble tile is recommended since it is typically machined and polished with a very smooth flat surface (this can be verified with the angle cube). Furthermore, marble is very easy to drill mounting holes through. Additionally, a triangular shape is recommended because it reduces wobbling significantly [34].

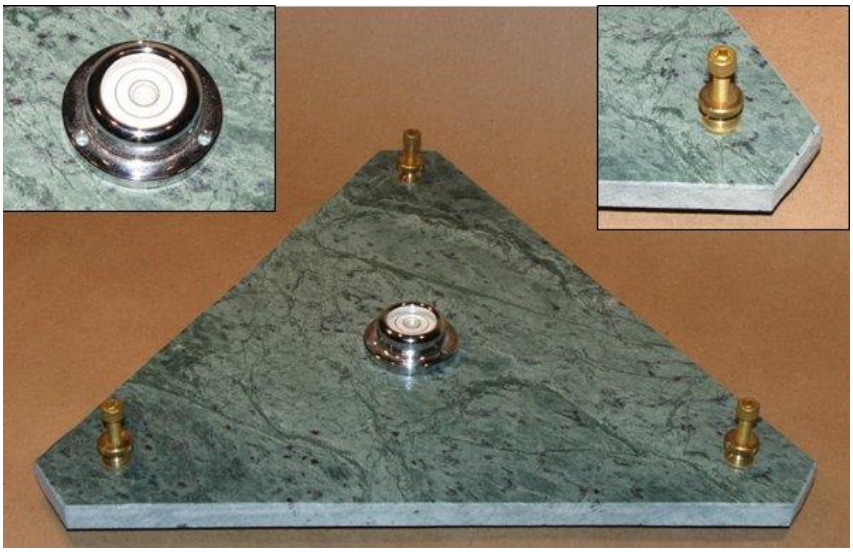


Figure 2.13: Leveling table [34]

The six positions of the tumble test must be orthogonal to each other. Placing the accelerometer on the inside of the cubes allows each side to sit flat on the table without the

interference of the accelerometer. Aluminum is the recommended material for the cube, as aluminum extrusions are very straight and square.

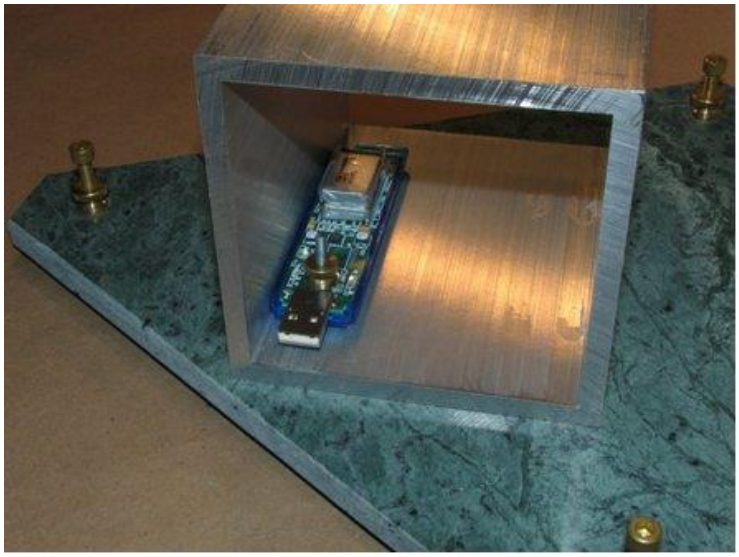


Figure 2.14: Aluminum cube extrusion [34]

For the tumble test place the cube on the six faces of the cube and wait 10 seconds at each position. Rotating the cube to a different face every 10 seconds, and record data.

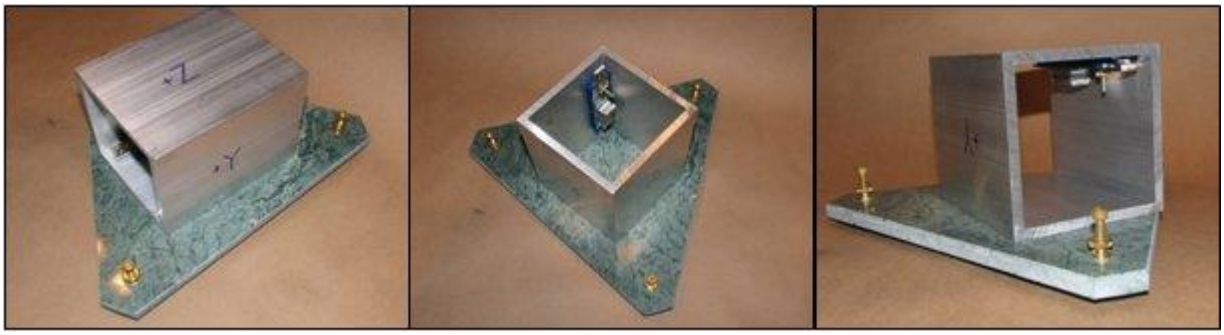


Figure 2.15: Rotating cube through the different positions [34]

After the data has been recorded is time for the data analysis, the six positions of the tumble test place each axis in the line of positive and negative gravity. While one axis is registering gravity, the other two axes are perpendicular to gravity and therefore register 0g. The ideal output relationship of the sensor is a line. The slope of the line is determined by using the measured values for positive and negative gravity. The offset can then be calculated using the values measured that

should read 0g (any deviation from that value is therefore the offset) [34]. The data is analyzed using the "counts" of the accelerometer (before converting to g-force). The new slope and offset values are substituted in the g-force equation to provide the calibrated g-force output. The formulas for the scale factor and offset will be as following:

Scale factor = $\frac{(average\ counts\ at{+}1g){-}(average\ counts\ at{-}1g)}{2g}$

Offset = (average counts when axis is perpendicular to g)

Like the gyroscope calibration procedure, this methods offers a relatively simple solution and inexpensive solution for accelerometer calibration. However, the sensitivity and precision required will have to be evaluated by the engineers in charge of the application and project. The more difficult the task, the more sophisticated the filter needs to be in its basic signal response.

There exists more advanced solutions on the software side, chapter 3 offers a more in depth review of these software solutions, in the form of filters. Several of these different filters will be discussed. In addition, chapter 3 also introduces Arduino implementation (which was used for this thesis), and talks about interval computation which is important to understand the Kalman filter and others.

# CHAPTER 3: COMBINATION OF SENSORS FOR ERROR COMPENSATION

## 3.1 IMU ARDUINO IMPLEMENTATION

The most affordable and simple implementation for an IMU is Arduino. Arduino is a user friendly and open source microcontroller board that is used for building electronic projects [35]. Figure 3.0 shows an Arduino UNO, the UNO is one of the more popular boards in the Arduino family and a great choice for beginners due to its simplicity and price.

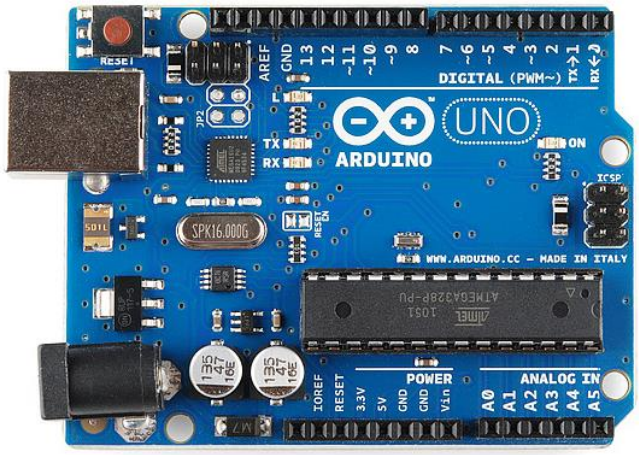



Figure 3.0: Arduino UNO [35]

The Arduino consists of both a physical programmable circuit board (the microcontroller), and its software called the Arduino Integrated Development Environment (IDE). The IDE can be run on virtually any computer/laptop, as it has very few requirements. The IDE writes and uploads code to the physical board. Due to its simplicity, the Arduino platform is the go to choice for people new to electronics (and students). This is because the Arduino does not use a separate piece of hardware to upload a new code onto the board, a simple Type B-USB can be used to get the job done. Figure 3.1 shows some of the different types of USBs cables available in the market (there are of course 1.0 versions of these USB cables as well).


Figure 3.1: USB types [36]

Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program. Finally, Arduino provides a standard form factor that breaks out the functions of the micro-controller into a more accessible package [35].


Figure 3.2: Arduino code example [35]

As it can be seen (on Figure 3.2), the Arduino code is very simple, these 10 lines of code can blink the LED on the board. The Arduino software and hardware is designed for artists, designers, hobbyist, students, and basically anyone that wants to create interactive objects or environments. Arduino can interact with LEDs, buttons, motors, speakers, GPS units, cameras, the internet, smart-phones, TVs, and of course IMUs. The hardware boards (microcontroller) are cheap, and they are intuitive and simple to learn. In addition to this, the Arduino is open source,

which means that the software is completely free for everyone. Being open source means that there is a large community of users that can contribute to code and release instructions for a variety of Arduino projects, these can be found all over the web for free [35].

So what exactly is on this microcontroller board? While there are a variety of Arduino boards that can be used for different purposes, for the scope of this thesis the UNO should be more than enough to get students started on the IMU world. Let us go over the basic components of an Arduino, which will be shown on Figure 3.3.

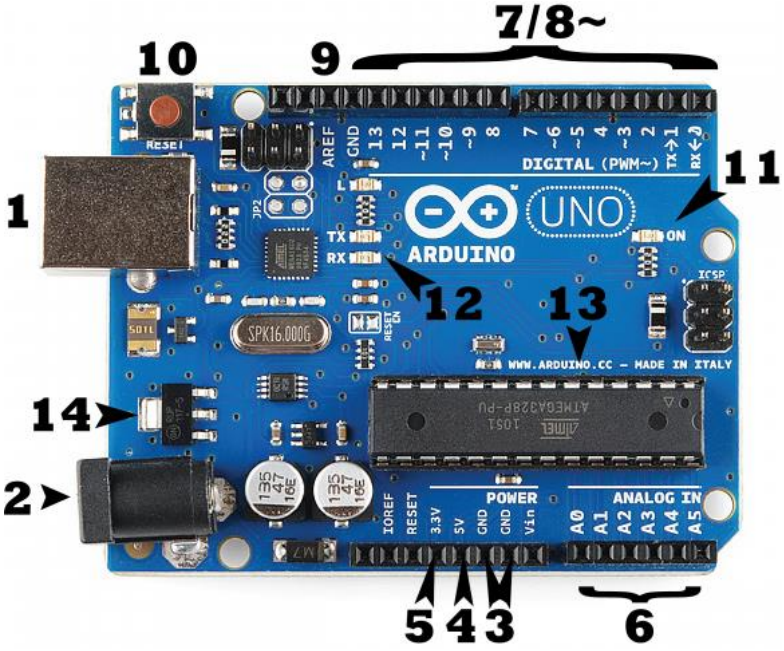



Figure 3.3: Arduino components [35]

The first components (1 & 2) are *Power*; the Arduino UNO can be powered from either a computer, wall power supply, or an external battery. The USB connection is labeled as 1 and the barrel jack is labeled as 2. Figure 3.4, 3.5, and 3.6 will show the configurations available to power the Arduino. On figure 3.4 the UNO will be power by a Type B-USB, which is also required to load code into the Arduino Board. On Figure 3.5 the UNO will be powered by a wall power supply with the adapter, and finally on Figure 3.6 the Arduino will be powered by a 9-V battery (with the required adapter). It is worth mentioning that the Arduino power supply uses a recommended

voltage value of between 6 and 12 volts, if a power supply greater than 20 V is used it will overpower and destroy the Arduino [35].

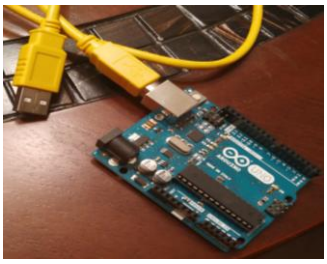


Figure 3.4: Type B-USB power configuration

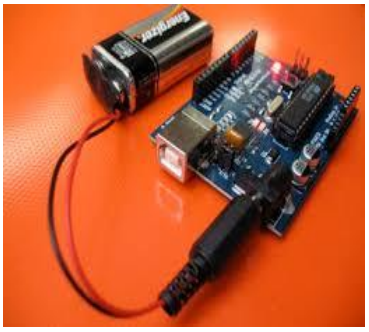


Figure 3.5: Wall adapter power configuration [37]

Figure 3.6: 9-V adapter power configuration [37]

*Pins (5V, 3.3V, GND, Analog, Digital, PWM, and AREF);* the pins on the Arduino board are the places where the wires are connected in order to construct a circuit. A breadboard will also be required, these wires usually have plastic headers that allow them to plug a wire right into the board. Figure 3.7 will show a breadboard and wires.

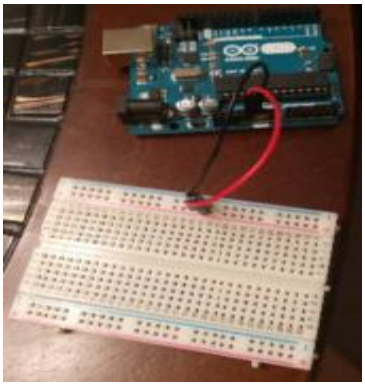


Figure 3.7: Breadboard and wires connected to Arduino

The Arduino has different pin kinds, each of them is labeled on the board and are used for different functions. *GND (3)*; short for ground, this pin is used to ground the circuit. *5V (4) & 3.3V (5)*; the 5V pin supplies 5 voltage of power, and the 3.3V pin supplies 3.3 volts of power. Since different components require a different power supply, selection of power is component dependent. *Analog (6);* this type of pin (A0 to A5) are Analog pins. This means that the pins can read a signal from an analog sensor (like the gyroscope) and convert it into a digital value using the ADC converter. *Digital (7);* next we can find the digital pins (0 through 13). This type of pin can be used for both digital input (pushing a button) and digital output (like the gyroscope). *PWM (8);* some of the digital pins have a *tilde* (~) next to them (3, 5, 6, 9, 10, and 11). These pins can act as regular digital pins, however, they can also be used for Pulse Width Modulation (PWM). This allows control over the duty cycle of a part, which in turns allow for control of motors, RGB LEDs, and a variety of applications. The PWM controls the voltage that the application is receiving, thus allowing for control of the duty cycle. Figure 3.8 will show different duty cycles.

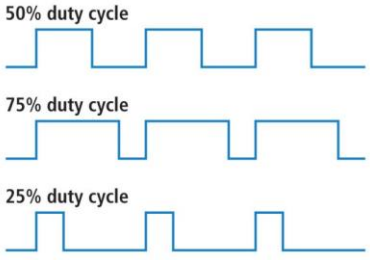



Figure 3.8: 50%, 75%, and 25% duty cycle [38]

In this example 100% duty cycle would be the same as setting the voltage to 5 Volts and 0% would be the same as grounding the signal [38]. PWM allows the part (sensor, motor, or something else) to receive lower voltage at the desired time frame. This can dictate RPMs, position, illumination intensity, and many other applications. *AREF (9);* it stands for Analog Reference, and it is used to set an external reference voltage (in the range of 0 and 5 volts) as the upper limit for the analog input pins. This can be used to avoid components to overheat and/or burn [35].

*Reset Button (10);* the Arduino has a button that when pushed it will temporarily connect the reset pin to ground, this will cause any code that is loaded on the Arduino to restart. This can be useful to repeat a code multiple time without the need to program it to do so[35].

*Power LED indicator (11);* the LED that is next to the word "ON" will light up when the Arduino is plugged into a power source. This light should always turn on when it is connected, if it doesn't the circuit should be checked, as this would be an indicator of something wrong with it [35].

*TX & RX LEDs (12);* the TX short for transmit and RX short for receive, they indicate the pins responsible for serial communication. TX and RX appear on digital pins 0 and 1, and on the LEDs indicators TX and RX, the LEDs give a feedback to the user that the Arduino is either receiving or transmitting data [35].

*Main IC (13);* is the integrated circuit of the Arduino, this can be thought of as the brains of the UNO. The main IC can differ from board type to board type, but it is usually part of the ATmega line of IC's from the ATMEL company. This is of significance because the IC type may be needed when loading a new program from the Arduino software. The IC serial number and information can be found written on the top side of the IC, if more data on the IC is required the datasheets can provide further information [35].

*Voltage Regulator (14);* the voltage regulator, just like its name implies, controls the amount of voltage that can be let into the Arduino board. It will reject extra voltage that can potentially harm the circuit, but it is not without its limits. If more than 20 voltage are used it will burn the Arduino [35]. There is more complex information available on all of the components, as

well additional peripherals that can be added to the Arduino, however this is beyond the scope of this thesis.

Now that the Arduino and IMU have been explained, the set up that was used from this thesis will be shown. Three different IMUs sensor were chosen: MPU-6050(6 DOF) Figure 3.9, MPU-6050 (10 DOF) Figure 3.10 and ALTIMU-10 V5 (10 DOF) Figure 3.11.

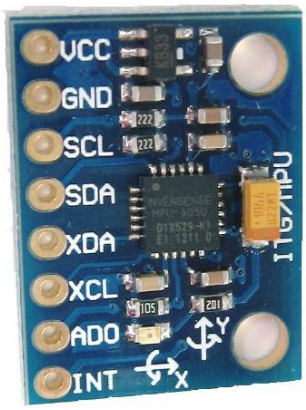



Figure 3.9: MPU-6050 (6 DOF) [39]

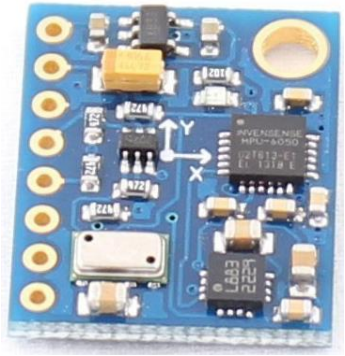


Figure 3.10: MPU-6050 (10 DOF) [39]

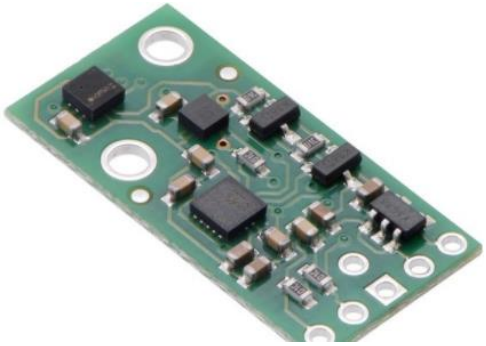


Figure 3.11: ALTIMU-10 v5 [40]

*MPU-6050(6 DOF);* the first sensor is the MPU-6050, manufactured by a company called InvenSense. It contains a MEMS triaxial accelerometer (ADXL345) and a three-axis MEMS gyroscope (L3G4200D) in a single chip (6 DOF). For the price (can be found as cheap as $ 5.00 USD), it as a very accurate sensor that can capture x, y, and z channel at the same time. It contains a 16-bit analog to digital conversion hardware for each of its channels, and the I2C-bus can interface with the Arduino [39].

*MPU-6050 (10 DOF);* while there exists a different sensor from InvenSense called the MPU-9150, which combines adds a magnetometer to its sensors, it uses a different library on Arduino. For this reason a MPU-6050 with a magnetometer and an altimeter was used, as it allows the same library to be used, thus simplifying the process. This particular MPU-6050 has the three axis gyroscope (L3G4200D), the triaxial accelerometer (ADXL345), a three-axis magnetometer (HMC5883L), and an altimeter pressure sensor (MS5611). Together they add 10 independent readings (or 10 DOF), while allowing the same configuration used on the 6 DOF MPU, which makes it very convenient for simplification purposes. More information on both models can be found on the MPU-6050, HMC5883L, and MS5611 datasheet [39].

*ALTIMU-10 V5 (10 DOF);* the Pololu AltIMU-10 v5 contains a LSM6DS33 gyroscope and accelerometer, LIS3MDL magnetometer, and a LPS25H barometer. Combined they add 10 independent readings (or 10 DOF) that can be used to calculate the sensor's altitude and absolute orientation. It has an I2C-bus interference for Arduino. More information on each of the IMU's components can be found on their respective datasheet [40].

Besides the datasheet information, each individual sensor has an Arduino library that allows interaction with the Arduino. These can be found all over the internet free of cost, it is worth noting that three of the main library locations for Arduino are the official Arduino website, github, and sparkfun. Most of the necessary libraries can be found on these 3 websites. These libraries can include examples, all the instructions needed to interact with the sensors, and even custom applications. Since the Arduino is open-source the user can use all of them legally, and even add their own codes to the websites.

In addition to the mentioned IMUs, there are other sensors that were used, as well as some recommended programs that can help with the visualization of the output. These programs will be discussed on chapter 4 in detail, for now let us explore the other sensors that were used to aid in this thesis. First off all, we have the HC06 module (shown in Figure 3.12), this sensor allows for Bluetooth connectivity from the Arduino (more information on this sensor can be found on its datasheet). It requires only four wires; one for power, one for ground, and the final two for TX and RX in order to connect with the Arduino.

Figure 3.12: HC-06 Bluetooth Serial Module [41]

Combining this Bluetooth sensor with the 9-V volt power option allows wireless capability for the IMU. There are even apps for the smartphone that can act as a serial monitor (via Bluetooth), allowing for more flexibility. There are more robust options available, such as the HC-12 which allows for up to 1.8 km communication, however this requires more than one Arduino and other special configurations that are beyond the scope of this thesis (and beyond the scope of people new to Arduino).

While the Bluetooth sensor may allow wireless connectivity, most simple Bluetooth devices only allow a maximum range of 100 meters (depending on the class) [42].

| Device Class | Transmit Power | Intended Range |
|---|---|---|
| Class 3 | 1 mW | less than 10 meters |
| Class 2 | 2.5 mW | 10 meters, 33 feet |
| Class 1 | 100 mW | 100 meters, 328 feet |

Figure 3.13: Device class and range for Bluetooth [42]

If the application requires more than that distance, an easy solution would be to use a data logger sensor. For this thesis the OpenLog (Figure 3.14) from sparkfun was used.

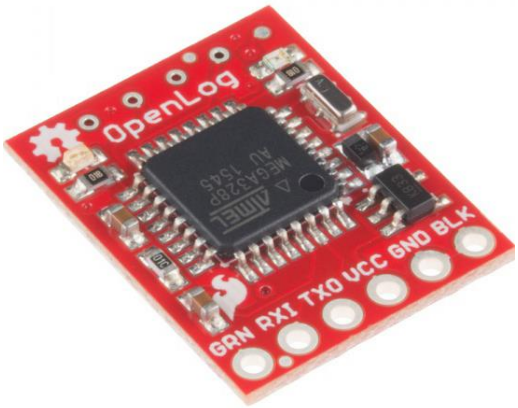



Figure 3.14: Sparkfun OpenLog data logger [43]

This data logger uses a microSD card to record data, and it allows for 64MB – 64GB size and both FAT16 and FAT32 format.

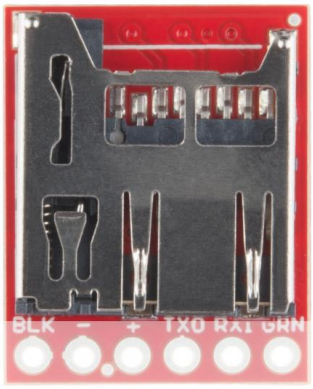



Figure 3.15: MicroSD slot [43]

The data is directly stored in the microSD as a .txt file that can be opened with notepad or any simple text editor program. More information can be found on the OpenLog's datasheet and on the Sparkfun sensor information site.

Now that we have all of the necessary components it's time to take some data. Connecting the IMU to the Arduino is trivial, the MPU-6050 only requires four jumper cables (Figure 3.16). There are a lot of libraries available for the MPU that allows the user to read the raw data output, the accelerometer's output will need to be divided by the FS in order to obtain results in g-force (more on this in the upcoming section). In order to obtain roll, pitch, and yaw angles it is necessary to combine the gyroscope and accelerometer data, this will be explained in the next section.

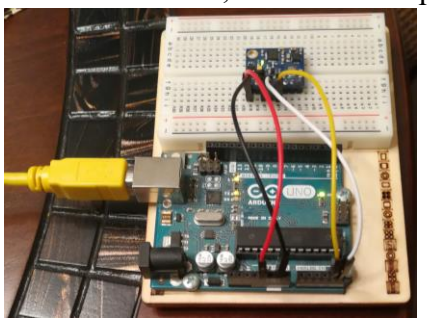


Figure 3.16: MPU-6050 configuration

## 3.2 COMBINATION OF MULTIPLE SENSORS

To demonstrate how to combine multiple sensors of the IMU device we will use the accelerometer and gyroscope. The first step is to align the coordinate systems. This is done by choosing the coordinate system of the accelerometer as the reference coordinate system. The accelerometer datasheet should display the direction of X, Y, and Z axes relative to the IMU [22].

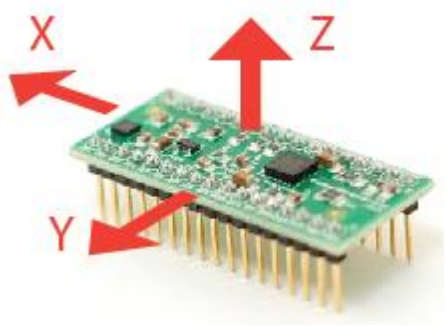



Figure 3.17: Coordinate system of accelerometer with respect to IMU [22]

The next step is to identify the gyroscope outputs that correspond to (using the terminology found on chapter 1) RateAxz and RateAyz. Additionally, determine if these outputs will need to be inverted due to the position of the gyroscope in relation to the accelerometer. The direction of the Gyroscope should be tested to confirm it. The following sequence can be used to determine which output of the gyroscope correspond to the RateAxz value discussed [22]:

1. Start by placing the device in a horizontal position. Both X and Y outputs of the accelerometer should output the zero-g voltage (for example 1.5V)
2. Rotate the device around the Y axis, or in other words, the device is being rotated around the XZ plane. The X and Z accelerometer output will change but the Y output will remain constant.
3. At the same time while rotating the device at a constant speed make a note of the gyroscope output changes and which ones remain constant.
4. The gyroscope output that changed during the rotation around the Y axis will provide the input value for AdcGyroXZ, in order to calculate RateAxz.
5. In order ensure that the rotation direction corresponds to the model it may be necessary to invert the rate RateAxz value due to the physical position of the gyroscope in respect to the accelerometer.
6. Repeat steps 2 – 4 while monitoring the X output of the accelerometer (AdcRx in the model). If AdcRx grows (during the first 90 degrees of rotation from the horizontal position), then AdcGyroXZ should in turn decrease. This happens because the gravitation vector rotates in one direction, and when the device rotates the vector will rotate in the opposite direction. If this is the case RateAxz can be inverted using the following equation: $\text{RateAxz} = InvertAxz * \frac{(AdcGyroXZ * \frac{Vref}{1023} - \text{VzeroRate})}{Sensitivity}$, where InvertAxz is either 1 or -1.
7. The same test can be repeated for the RateAyz by rotating the device around the X axis. This will allow the user to identify the gyroscope output that corresponds to the

RateAyz, and if it requires to be inverted. The following formula allows RateAyz to be inverted, after the appropriate value for InvertAyz has been determined using the steps previously mentioned:

$$\text{RateAyz} = InvertAyz * \frac{(AdcGyroYZ * \frac{Vref}{1023} - \text{VzeroRate})}{Sensitivity}$$

The assumption after this point is that the IMU will be have a setup that allows the calculation of the values Axr, Ayr, and Azr (as defined on section 1.3.1) as well as RateAxz and RateAyz (as defined on section 1.3.2). These relate to each other in such a way that it allows a more accurate estimation of the inclination angles of the device relative to the ground plane [22]. As previously mentioned, the accelerometer data is very sensitive to vibration and mechanical noise in general. This is the main reason that the IMU uses a gyroscope to smooth and reduce the accelerometer errors. However, the gyroscope while less sensitive to linear mechanical movements, has a problem with drift (not returning back to its zero-rate value when the rotation stops). For this reason, the average of data that comes from both the accelerometer and gyroscope outputs combined, are relatively a better estimation of the device inclination that could be obtained by using either of the output data by itself [22].

The following will introduce an algorithm that is based on a Kalman filter (the KF will be explained on section 3.3.2), however this algorithm is far simpler and easier to implement on embedded devices such as the Arduino. This algorithm should calculate the direction of the gravitation force vector R = [Rx, Ry, and Rz], which will be used to derive other values like Axr, Ayr, and Azr or cosX, cosY, and cosZ. Let us go back to the equations used in section 1.3.1:

$$\text{Rx} = \frac{(\frac{(\text{AdcRx})(\text{Vref})}{1023} - \text{Vzerog})}{Sensitivity} = \text{RxAcc}$$

$$\text{Ry} = \frac{(\frac{(\text{AdcRy})(\text{Vref})}{1023} - \text{Vzerog})}{Sensitivity} = \text{RyAcc}$$

$$\text{Rz} = \frac{(\frac{(\text{AdcRz})(\text{Vref})}{1023} - \text{Vzerog})}{Sensitivity} = \text{RzAcc}$$

This may lead to believe that the values Rx, Ry, and Rz are already known, however, these values are derived from the accelerometer data only. Thus, these values are noise heavy, the accelerometer

values will then be redefined as the following vector: Racc = [RxAcc, RyAcc, RzAcc]. Since these values can be obtained from the accelerometer data, this will be considered an input in the algorithm. The length of this vector will be close to 1g. This vector can be redefined as the following: $|Racc|=\sqrt{(RxAcc^2 + RyAcc^2 + RzAcc^2}$

Additionally, this equation can be normalized to ensure the length will always be equal to 1. This equation will be as following:

Racc (normalized) = $[\frac{RxAcc}{|Racc|}, \frac{RyAcc}{|Racc|}, \frac{RzAcc}{|Racc|}]$ [22]

A new vector will now be introduced as the corrected values that are based on gyroscope data and past estimated data, it will be called *Restimate*. And the equation is as follows: Restimate = [RxEst, RyEst, RzEst] [22]

So what exactly will this algorithm do? It will first use the accelerometer output data to tell the position, however it will "confirm" this data and correct it using the gyroscope data as well as the past *Restimate* data (this is called feedback control). Finally, it will output a new estimated vector *Restimate*. As it can be seen, this will use both accelerometer data and gyroscope data in order to give the best estimate of the current position of the device [22]. The first step is to take an initial reading, we will have to trust either the accelerometer or gyroscope data as the first reading. In this case we will use the accelerometer data, the following equation is a simplified version of the full equation:

Restimate (0) = Racc (0)

Because the Racc and Restimate are both vectors, the full equations would be defined as [22]:

RxEst (0) = RxAcc (0)

RyEst (0) = RyAcc (0)

RzEst (0) = RzAcc (0)

The sensor will then take regular measurements at equal time intervals of T seconds (this can be anything the user wants to), this will allow the user to obtain new measurements that will be defined as Racc (1), Racc (2), Racc (3)… Racc (n). In tangent, it will also add new estimates at

the same time intervals Restimate (1), Restimate (2), Restimate (3)….. Restimate (n) [22]. Let us suppose we are measuring value *n*. We then will have two known set of values: Restimate (n-1); which is the previous estimate, Restimate (0) = Racc (0), and Racc (n); which is the current accelerometer measurement. In order to calculate Restimate (n), we will introduce a new value that can be obtained from the gyroscope output and a previous estimate. It will be named *Rgyro*, it will also be a vector consisting of 3 components:

Rgyro = [RxGyro, RyGyro, RzGyro] [22]

Let's use the model introduced in section 1.3.2 to calculate the vector components:

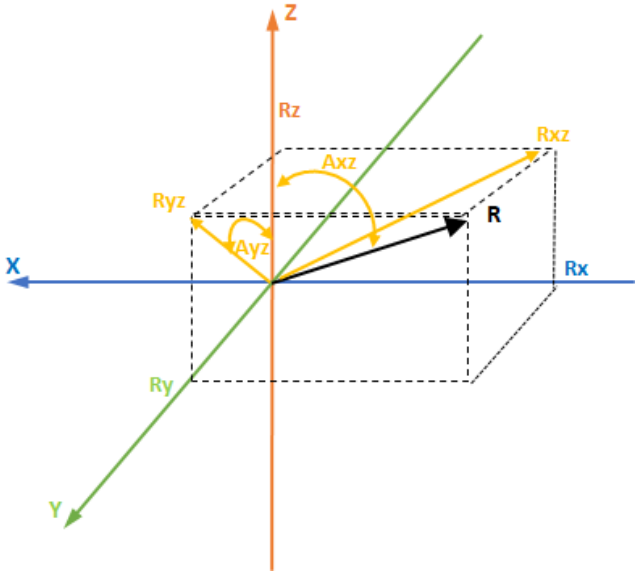



Figure 3.18: Gyroscope angles between axes and force vector [22]

The right-angle triangle that is formed by Rz and Rxz will give the following equation:

Tan (Axz) = $\frac{Rx}{Rz}$ = Axz = $atan2(Rx, Rz)$

Atan2 is a function very similar to atan (arc tangent), except it returns values in the range ($\frac{-\pi}{\pi}$), in addition it takes 2 arguments instead of one. Thus, it allows the conversion of Rx and Rz to angles in the full range of 360 degrees ($\frac{-\pi}{\pi}$). Now that RxEst (n-1) and RzEst (n-1) are known we can find:

Axz (n-1) = atan2 (RxEst (n-1), RzEst (n-1)) [22]

Going back to section 1.3.2, it is known that the gyroscope measures the rage of change of the Axz angle. Thus, it will be possible to estimate the new angle Axz (n) using the following formula:

Axz (n) = Axz (n-1) + RateAxz (n) *T

As reminder, the RateAxz is obtained from the gyroscope ADC readings using the following:

$\text{RateAxz} = \frac{(\frac{(\text{AdcGyroXZ})(\text{Vref})}{1023} - \text{VzeroRate})}{Sensitivity}$ [22]

In order to obtain a more precise formula an average of the rotation rate can be use:

$\text{RateAxzAvg} = \frac{RateAxz\ (n)+RateAxz\ (n-1)}{2}$ [22]

Now we can calculate both angles:

Axz (n) = Axz (n-1) + RateAxzAvg * T

Ayz (n) = Ayz (n-1) + RateAyzAvg * T [22]

Since the angles Axz (n) and Ayz (n) are known, it is time to obtain RxGyro and RyGyro using the following formula:

$|Rgyro|=\sqrt{(RxGyro^2 + RyGyro^2 + RzGyro^2}$ [22]

Because the Racc vector has been normalized, it is possible to assume that the length (1) has not changed after rotation, thus it allows for the following simplification:

$|Rgyro| = 1$

Using the relations above, it is possible to use the following formulas:

$\text{RxGyro} = \frac{\text{RxGyro}}{1}$, thus : $\text{RxGyro} = \frac{\text{RxGyro}}{\sqrt{(RxGyro^2+RyGyro^2+RzGyro^2}}$

The next step is to divide the numerator and denominator of fraction by the following value: $\sqrt{(RxGyro^2 + RzGyro^2}$ , thus we get:

$\text{RxGyro} = \frac{\text{RxGyro}/\sqrt{(RxGyro^2+RzGyro^2}}{\sqrt{(RxGyro^2+RyGyro^2+RzGyro^2}/\sqrt{(RxGyro^2+RzGyro^2}}$

It is known that the denominator $\text{RxGyro}/\sqrt{(RxGyro^2 + RzGyro^2}$ = sin (Axz), simplifying:

$\text{RxGyro} = \frac{\sin\ (Axz)}{\sqrt{(1+RyGyro^2}/\sqrt{(RxGyro^2+RzGyro^2}}$ [22]

The next step is to multiply the numerator and denominator inside the square root by $RzGyro^2$, we will obtain the following simplified equation:

$$\text{RxGyro} = \frac{\sin(Axz)}{\sqrt{(1+RyGyro^2 * RzGyro^2}/\sqrt{((RxGyro^2+RzGyro^2)(RzGyro^2)}}$$

Likewise $\frac{RzGyroZ}{\sqrt{(RxGyro^2+RzGyro^2}} = \cos(\text{Axz})$ and $\frac{y}{z} = \tan Ayz$, simplifying:

$$\text{RxGyro} = \frac{\sin(Axz)}{\sqrt{(1+\cos(Axz)^2 * \tan(Ayz)^2}}$$

Going back the algorithm we can use the same notation:

$$\text{RxGyro} = \frac{\sin(Axz\ (n))}{\sqrt{(1+\cos(Axz(n))^2 * \tan(Ayz(n))^2}}$$

$$\text{RyGyro} = \frac{\sin(Ayz\ (n))}{\sqrt{(1+\cos(Ayz(n))^2 * \tan(Axz(n))^2}}$$ [22]

This equation allows us to find the required gyroscope angles:

$\text{RzGyro} = \text{Sign (RzGyro)} * \sqrt{1 - RxGyro^2 - RxGyro^2}$

Sign (RzGyro) = 1, if RzGyro ≥ 0, and conversely, Sign (RzGyro) = -1, if RzGyro < 0

Simplifying, Sign (RzGyro) = Sign (RzEst (n-1)) [22]

As a note, it is important to be careful when RzEst (n-1) is very close to 0. This is because Rz is used as a reference for calculating Axz and Ayz angles, when the value is close to 0 it may overflow and trigger erroneous results. It may be convenient to skip this gyro phase and instead assign: Rgyro = Rest (n-1) [22].

In order to calculate the updated estimate of Rest (n) the following formula will be used:

$$\text{Rest (n)} = \frac{(\text{Racc} * \text{w1} + \text{Rgyro} * \text{w2})}{(w1+w2)}$$ [22]

Simplifying this formula by dividing the numerator and denominator of the fraction by w1 the following is obtained:

$$\text{Rest (n)} = \frac{(\text{Racc} * \frac{\text{w1}}{\text{w1}} + \text{Rgyro} * \frac{\text{w2}}{\text{w1}})}{(\frac{w1}{w1}+\frac{w2}{w1})}$$

This equation can be further simplified by $\frac{w2}{w1}$ = wGyro as:

$$\text{Rest (n)} = \frac{(\text{Racc} + \text{Rgyro} * \text{wGyro})}{(1+wGyro)}$$ [22]

In the formula above wGyro is a value that can be chosen experimentally, typical values include numbers between 5-20 (which usually trigger good results). The main difference between this algorithm from a Kalman filter (which will be explored in more detail in section 3.3.2) is that the weight is relatively fixed, however in a Kalman filter the weight is permanently updated based on

the measured noise in the accelerometer readings (and a very interesting component called the Kalman gain). However, this simplified algorithm will be good enough for most simple applications. In addition wGyro can be modified to be adjusted depending on some noise factors, but fixed values will work for simple applications. [22]

The final step in this algorithm is to get the updated values from the following equation:

$$\text{RxEst (n)} = \left(\frac{RxAcc + RxGyro * wGyro}{1 + wGyro}\right)$$

$$\text{RyEst (n)} = \left(\frac{RyAcc + RyGyro * wGyro}{1 + wGyro}\right)$$

$$\text{RzEst (n)} = \left(\frac{RzAcc + RzGyro * wGyro}{1 + wGyro}\right)$$

These equations can be further simplified by normalizing the vector:

$R = \sqrt{RxEst\ (n)^2 + RyEst\ (n)^2 + RzEst\ (n)^2}$, giving the final equations:

$$\text{RxEst (n)} = \frac{RxEst\ (n)}{R}$$

$$\text{RyEst (n)} = \frac{RyEst\ (n)}{R}$$

$$\text{RzEst (n)} = \frac{RzEst\ (n)}{R}$$ [22]

This algorithm will allow an acceptable amount of error for most applications. In the following sections different integration strategies will be explored, as well as the Interval computation which provides more detail in the backbone behind this type of algorithm.

### 3.3 INTEGRATION STRATEGIES FOR ERROR COMPENSATION

There are several more options for filtering data, however one of the main limitations of using Arduino, is the Arduino itself (specially the UNO). The Arduino does not have a lot of computing power nor does memory, thus, this restricts the data and the usage of computationally expensive equations. To mend this, a simplified algorithm that can be used to improve the quality of the output with minimum memory taxing implications will be explored. This algorithm is called the complementary filter (section 3.3.1)

### *3.3.1 Complementary filter*

The complementary filter is a simple, yet powerful technique used in the flight control industry for estimation of position or velocity. It achieves this by combining measurements of several components into one output. This filter is usually designed without reference to the Kalman filters, although it is related to it (as it will be seen). The main difference, as mentioned earlier, is that the complementary filter does not consider any statistical description for the noise corrupting the signals. In other words, this filter is obtained by a simple analysis in the frequency domain. A basic complementary filter is composed of two elements, they will be called $x$ and $y$ for simplification purposes. These two elements are noisy measurement of some signal $z$, and the $\hat{z}$ element is the estimate of $z$ resulting after applying the filter [44]. Figure 3.19 will show this filter example.

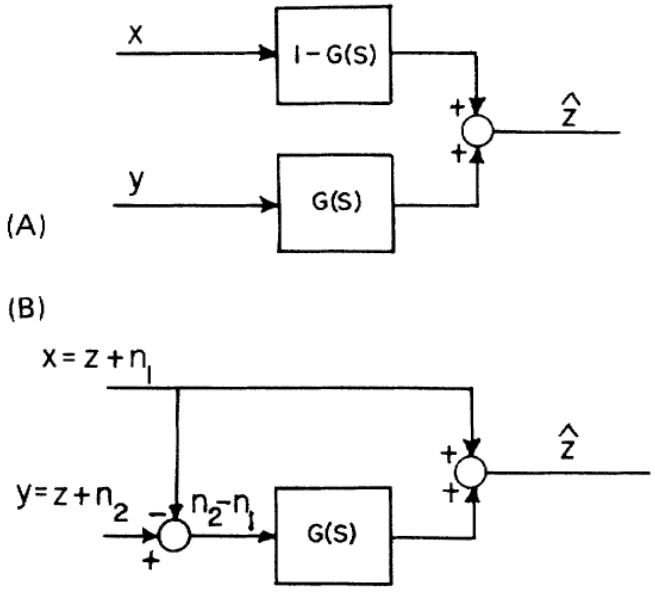



Figure 3.19: A) Basic complementary filter example B) Alternative version [44]

For further simplification, it will be assumed that the noise in the $y$ element is mostly composed of high frequencies, and that the noise in x is mostly low frequency (all of this is on example A). With these assumptions G (s) will be a low-pass filter in order to filter out the high-frequency noise in $y$. [1- G (s)] is the complement, in other words a high-pass filter that will filter

the low-frequency noise in *x*. As it can be seen, no detailed description of the noise processes are considered in this simplified complementary filter [44].

A different method for the same filter can be seen in example B, in this case the input for *G (s)* is $y - x = n_2 - n_1$. As it can be seen, the filter just operates on the noise or error in measurements of the *x* and *y* components. It can be noted that in the cases of error free measurements ($\hat{z}$ = z [1-G(s)] + zG (s) = *z*) the signal will then be estimated perfectly [44].

Another application of the complementary filter is to combine measurements of vertical acceleration and barometer data (vertical velocity), in order to obtain an estimate of vertical velocity. The acceleration measurement will be integrated to produce a velocity signal that will be called $\dot{h_a}$ (this will be shown on Figure 3.20).

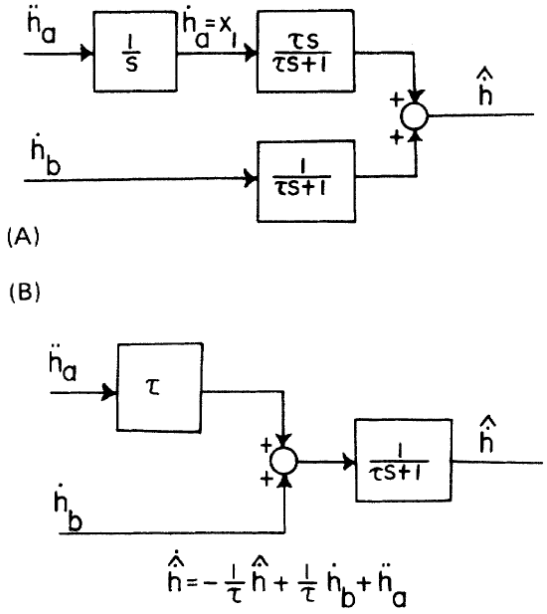



Figure 3.20: A) Basic complementary filter B) Actual realization of filter [44]

The integration of the velocity attenuates the high frequency noise in the acceleration measurement, however, the noise in $\dot{h_b}$ has not changed. Therefore if $h_b$ is filtered by a low pass filter, then $\dot{h_a}$ is filtered by a high pass filter. This will give the following equation:

$1 - G\,(s) = 1 - \frac{1}{\tau s+1} = \frac{\tau s}{\tau s+1}$ [44]

The time constant τ is usually in a range of 2-6 seconds and will be adjusted during simulation or flight testing. Figure 3.20 A shows a simplified filter, while B shows the actual realization of the filter. In the filter the measurement, $\ddot{h}_a$ has went through a low pass filter, while $\dot{h}_a$ has gone through a high pass filter; the reason is the integration [44].

The following example will resemble more closely a use for an IMU sensor. In this example an acceleration measurement is combined with a position measurement, this allows position and velocity to be estimated. Figure 3.21 will show the approach a complementary filter could take in order to improve to output results. Figure 3.21 (A) estimates the velocity from position an acceleration measurements, while Figure 3.21 (B) estimates the position using acceleration and position measurements [44].

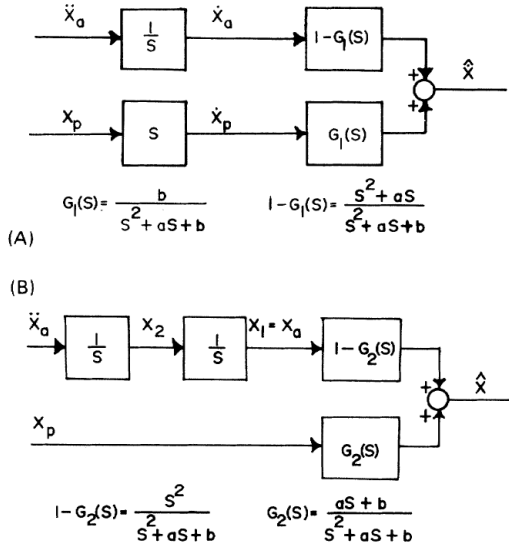



Figure 3.21: A) Complementary filter for Velocity, B) Complementary filter for position [44]

Now, let us focus on the complementary filter that was used on this thesis. A very simplified method was used, which combines the accelerometer and gyroscope data giving the following formula:

Filtered Angle = $\alpha\ (Gyroscope\ angle) + (1-\alpha)\ (Accelerometer\ angle)$, where
$\alpha = \frac{\tau}{\tau+\Delta_t}$, Gyroscope angle = (Last measured filtered angle) + (ω) (Δ $\tau$) [45]

For the most part, the accelerometer and gyroscope outputs follow the same filtering method mentioned on section 3.2, where the accelerometer data is trusted initially and then it is averaged. That data is used along with the raw gyroscope data to create the gyroscope angle. $\Delta_t$ is the sampling rate, while τ is the time constant greater than the timescale of a typical accelerometer noise [45]. Let us walk through a reduced step guide onto how this particular complementary filter is implemented on the Arduino software:

1. The first step is to "initialize" and define all classes and variables that will be used in the Arduino code.
2. Define a variable that will store the last read gyroscope angles and accelerometer data.
3. Define a variable for the "raw" gyroscope and accelerometer output data.
4. Define a variable that will be "the base acceleration and gyroscope output data", this will be used for filtering purposes.
5. The acceleration data will be combine with the gyroscope data in a variable for filtering purposes. The accelerometer data will be "trusted" more than the gyroscope output on this step.
6. Read and record several (in this case I chose 10) gyroscope and accelerometer outputs. Average the results to create the "base" acceleration and gyroscope output data, it will be used for the filtered outputs in a future step. This step is known as a calibration.
7. Convert the combined gyroscope output (combined with accelerometer data) to degrees/ sec. Using the following formula, and the conversion factor FS_SEL: $Gyro_x = \frac{(accel.gyro.value)\ (gyro) - base_x gyro)}{\text{FS\_SEL}}$ (same formula will be used for all 3 angles x, y, and z).
8. Convert the accelerometer output to get angle values, using the following formula: $Accel_{angley} = \arctan(\frac{(-1)(accel_x)}{\sqrt{{accel_y}^2 + {accel_z}^2}}) * (\frac{180}{3.14159})$ ; for the x angle:

$Accel_{anglex} = \arctan(\frac{(accel_y)}{\sqrt{accel_x{}^2 + accel_z{}^2}}) * (\frac{180}{3.14159})$. Note that z angle cannot be calculated because these computations rely on gravity pointing in the Z direction, and are invariant to rotation around the Z- axis (Yaw).

9. Compute the filtered gyroscope angles using the following formula: $gyro_{angle_x} = gyro_x\,(dt) + last_{gyro_x}$ ; where dt = $\frac{current\ time - previous\ time}{1000}$ (same formula will be used for x, y, and z angles).
10. Apply the complementary filter formula using the "filtered gyroscope angles" and "filtered accelerometer angles". The following formula will be used: $ed\_Angle_x = (\alpha)\left(gyro_{angle_x}\right) + (1-\alpha)\,(Accel_{anglex})$ ; note that the same formula will be used for x and y angle, as the z angle cannot be obtained for the previously explained reason.
11. Additionally, it should be noted that $\alpha = 0.96$ (this is user defined, and this number should be experimented to find the appropriate value).

The results from this complementary filter will be showed on chapter 4. This filter provides powerful results in a relatively simple to implement and compute manner. In the next section interval computation will be explored, this provides a more in depth overview of the way Kalman filter and other similar algorithms function.

### *3.3.2 Interval Computations*

This section will introduce the concept of interval computations, this concept will be used to help understand the need for the Kalman filter (see next section) and the fundamentals behind its theory. This section will not only describe the techniques used, but will also explain the shortcomings and issues from using these techniques (briefly). It will first explain why these computations are required in the first place, continuing with the uncertainty related to them (interval uncertainty). Finally, it will list some typical applications of these techniques [46].

On theory, all problems can be separated into three main classes, which can be classified as the following "requirements":

- Learn what is currently happening in the world; these are numerical values of different quantities such as: distances, masses, coordinates, etc.
- Based on the values obtained, predict how the state of the world will change over time.
- Finally, what changes would need to be made in the world so that these changes will lead to the desired results.

As it can be seen, these steps are essentially the steps that were needed for the complementary filter, and the algorithm mentioned in the previous section [46].

Let's examine these situations in more detail, starting with the current state of the world. When trying to learn the current state of the world there are several cases that are possible. We know that for these kind of problems numerical values of different quantities "*y"* characterize the states. Sometimes these quantities can be directly measured, for example measuring a patient's temperature, weight, etc. In other cases we can simply ask an expert, and the expert can provide an approximate value $\tilde{y}$ of the *y* quantity. At times this is a requirement, because some measurements cannot be directly measured, thus the only way to learn information about them is to either measure (or ask an expert to estimate) an easier to measure quantity $x_1 \ldots . x_n$. Taking this measurement will allow us to estimate *y* based on the obtained values $\tilde{x}_i$(estimate) of the quantities $x_i$ [46].

In order to estimate the value of *y,* it is first required to know the relation between *y* and the easier to measure quantities $x_1 \ldots . x_n$. The estimates of $x_i$ will be used to come up with an estimate for *y*. For the relation between y and $x_i$ we will use an algorithm $\mathrm{F}\ (x_1 \ldots . x_n)$, which will transform the values of $x_i$ into an estimate for *y*. Once the algorithm is known and the *x* quantities are measured, *y* can be estimated as $\tilde{y} = \mathrm{F}\ (\tilde{x}_1 \ldots . \tilde{x}_n)$. Figure 3.22 will show this algorithm.

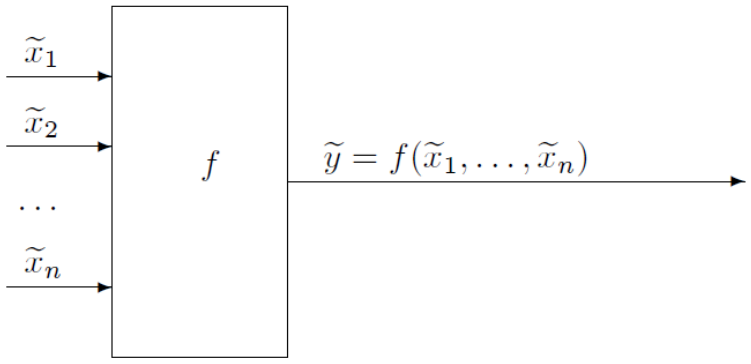



Figure 3.22: Approximation algorithm example [46]

Depending on the situation there will be different algorithms, however, as it has been seen before, if the algorithm is very complex it will require a lot of computations [46].

Once the values $y_1 \ldots . y_m$ are known (this is known as characterization of the state), it is possible to start predicting the future state, or in other words the future values of these quantities. For simplification purposes we will call the future state '*z'*, the *z* state value will depend on the current values $y_1 \ldots . y_m$. More specifically, known estimates $\tilde{y}_i$ for $y_i$ will be used in order to come up with an estimate for *z*. In other words, the relation between *z* and $y_i$ must be given in the form of an algorithm g $(y_1 \ldots . y_n)$, the algorithm (g) could be very complicated, time consuming, and computational taxing. The computational part of applying these algorithms is called data processing [46].

Computations from these types of problems, as mentioned before, are only estimates that are approximately equal to the (unknown) actual values $(\tilde{x}_i)$ of the corresponding quantities. This estimate will never be exact for 2 reasons; first, the actual value is a real number, thus infinitely many bits would be needed to describe the exact value. However, only a finite number of bits can be made after every measurement. The other reason has been explored heavily, there will be always noise mixed with the results. In addition to the *approximation error* of this estimation, there is also a *measurement error* on the value obtained by the measurement. Interval computations enable the estimation of the uncertainty in *y* caused by the uncertainty of the inputs [46].

In some cases it is important to not only estimate the quantity *y,* but also to understand how accurate that estimate is. This is known as a *tolerance,* and just like in manufacturing, the limit is dependent on the application. For example, an estimate of 100 ± 10 would have a value in the range of 110-90, the *actual value* would be in that range and depending on the application this may or not be acceptable.

What's worse about these noise issues is that they keep adding up. For example, we start with estimates $\tilde{y}_i$ of the current values, after that the algorithm $\tilde{z} = \mathrm{g}(\tilde{y}_1 \ldots . \tilde{y}_m)$ will give the value for the desired future value of *z.* It is known that the estimates $\tilde{y}_i$ are different from the (unknown) actual values of $y_i$. Therefore, even if the prediction algorithm is absolutely exact, in other words $z = \mathrm{g}(y_1 \ldots . y_m)$, the prediction result $\tilde{z}$ will be different from the actual value *z.* It is worth noting that in most practical situations, the prediction algorithm is only approximately known, this means that there is also model uncertainty added to the other sum of uncertainty. We can begin to see the limitations of these algorithms [46].

Let us dwell more in these uncertainties, we will begin with the direct measurement uncertainty. In order to estimate the uncertainty $\Delta_y$ that is caused by the measurement uncertainties of $\Delta_{x_i}$, we must first understand that there are several possible values of $\Delta_{x_i}$ (since we do not know the exact value of $x_i$). It is not all bad, manufacturers of a measuring device (typically) provide an upper bound $\Delta_i$ for the (absolute value of) possible measurement errors, in other words it is guaranteed that $|\Delta_{x_i}| \leq \Delta_i$. This allows the following formula to be true. This formula guarantees that the actual (unknown) value must be within the following interval:

$x_i = [\tilde{x}_i - \Delta_i, \tilde{x}_i + \Delta_i]$ [46].

In most practical applications, it is assumed that the corresponding measurement errors are normally distributed with 0 means and known standard deviation [46].

If possible, it is also useful to compare the values of a much more accurate measuring instrument, as this will allows us to get a better understanding of the measurement error of the instrument. It should be noted that in some cases there is no better measurement equipment, or

said instrument is too expensive. When these cases arise, there will be no comparative measurements for the desired application [46].

Since the function $f(x_1, \ldots\ldots, x_n)$ is usually continuous, this range is also an interval. In other words $y = [y, \bar{y}]$ for some $y$ and $\bar{y}$. Thus, in order to find this range it is sufficient to find the end points $y$ and $\bar{y}$ of this interval. On traditional data processing, the estimates of $\tilde{x}_i$ are the input values, and these will be used to estimate $\tilde{y}$ for the desired quantity $y$. As mentioned, these algorithms often require a lot of computing power, the alternative is to use *intervals.* Instead of numerical estimates, we will use both intervals for input and output ($[y, \bar{y}]$) for the possible values of $y$, these computations will be known as *interval computations [46]*.

While the interval computations may require less computing power it has some limitations, the main one is that it relies on estimations. These estimations be it from measurement results or expert estimates, will always have some error to it. The other issue is that depending on the application, the range required could be smaller than possible with the data available. Interval computations are a useful techniques for processing fuzzy data as well, which is also referred as a nested or family of intervals [46].

There are a lot more uses, limitations, and more intrications to the intervals computations, however this is beyond the scope of this thesis, as they were only intended to give an introduction to the Kalman filter. Interval computations gave us more knowledge to the algorithms and the theory behind estimations and filtering techniques. In the next section we will examine one of the more useful tools for improving output data; the Kalman filter.

### *3.3.2 Kalman filter*

Now that we have stronger understanding of what a filter is capable of (and its limitations), we can explore another powerful option; the Kalman filter. First and foremost, just like the complementary filter and our algorithm example, the Kalman filter is not a filter in the sense that it explicitly blocks certain frequencies and passes others (like the high and low pass filter). The Kalman filter is an estimator that is designated to make a "best-guess" output in the presence of

noise or uncertainty, rather than for a specific frequency response [47]. The Kalman filter does this just like the Complementary filter, by combining information from several outputs (sensors in our case).

The Kalman filter can be used for estimation of dynamics, for example a vehicle's position, velocity, or the IMU output data. It is typically used when there are noisy measurements (accelerometers, gyroscopes). The Kalman filter makes "educated guesses" about what the system is going to do next, thus Kalman filters are ideal for systems which are continuously changing. While the Kalman filter is more taxing than the complementary filter (computationally speaking), they only keep data of the previous state [48].

Let us walk through a quick overview of how the Kalman filter works. We already covered most of this in the algorithm and in the Interval computations, but there are some additional components used in the KF that we have not explored in the previous chapters. There are mainly two things that the Kalman filter will keep track of during its use:

- The estimate of the current position (or velocity), also called the state vector.
- The uncertainty of the estimate the position, called the *state covariance* [47].

Now, let us do an overview of the process of the Kalman filter:

- Starting from the current estimate of the position, the next step is to guess where you will be at the next state using a model. However, this state is not perfectly known, thus the uncertainty will grow. This is called predicting the *next state,* and the uncertainty is named the *next state covariance.*
- Next, take a *measurement* or *observation* (depending on the application), of the position from the sensor at the time that you will be predicting. Since there is noise on the sensor, this will only be an estimate of the next state. The noise will form the *measurement covariance*, which is a description of how accurate you "think" the sensors are.
- Work out the difference between the measurement and the predicted next position. This is called *measurement residual*.

- The next step is to work out the uncertainty on the difference between the measurement and the predicted position (or velocity), called *innovation covariance.* This will allow us to know which one to "trust" more (measurement or motion model).
- The next step is to calculate the *Kalman Gain,* it is the factor of how much we weight the motion model against the measurements. This will come from the residual and the innovation covariance calculated before. This *changing* factor is one of the most important parts of the Kalman filter.
- Correct the prediction of the next state depending how much of the measurement is "trusted". This is achieved my multiplying the Kalman Gain by the residual and adding it to the next predicted state.
- Finally, adjust the state covariance by taking a measurement. If done right, the uncertainty in the position (or velocity) should reduce more and more.
- Repeat all of the steps until you are done taking data.

While the operations are simple (it is just some matrix equations), the main problem is that matrix inverses are computer taxing. Thus the AVR (the microcontroller Arduino uses) can only handle low-sample rate implementations of a Kalman filter [47].

Now, let us go through a more detailed description of all the steps in the Kalman filter. As always, everything is easier to understand with an example. Let's imagine a little autonomous robot that can move around, the robot will then need to know exactly where it is in order to navigate without falling or crashing [48].

Figure 3.23: Autonomous robot [48]

The current state of the robot is $\overrightarrow{x_k} = (\vec{p}, \vec{v})$. This list of variables $(\vec{p}, \vec{v})$ could be anything the user chose, it is all depending on the application of the system that the user will be using. Going back to the problem, it is known that the robot has a GPS sensor, this sensor is accurate to about 10 meters. Since there are a lot of cliffs in the area the robot needs to be very precise, or it could fall off a cliff. In other words, the GPS is not good enough for this application [48].

Figure 3.24: Autonomous robot falling off a cliff [48]

Additionally, we know the commands sent to the wheels motors, and we know that if it's headed in one direction (and nothing interferes), at the next instant it will likely be further along the same direction; this is called a *prediction*. Of course, a lot of noise factors could affect our little robot. It could be buffeted by wind, or if the ground is wet, it could slip, or roll over bumpy terrain. In other words, the amount that the wheels have turned do not exactly represent how far the robot has traveled, and the prediction won't be perfect [48].

Thus, the GPS sensor give us some information about the state, but only indirectly, and with some uncertainty or inaccuracy added to its measurements. The wheels movement also give us some information about the robot, but only indirectly, and with added uncertainty. Finally, the prediction tells us something about how the robot is moving, but only indirectly, and once again, with added uncertainty or inaccuracy (starting to notice a pattern?). Combining all of this information together forms the Kalman filter [48].

Since the Kalman filter uses matrices for modeling, we will change our state to be in this form: $\vec{x} = \begin{bmatrix} p \\ v \end{bmatrix}$. As explained before, we don't know what the actual position and velocity are.

However, there is range of possible combinations of position and velocity that could be true, some more likely than others (remember interval computations). Figure 3.25 provides an example of likeness of these events. The Kalman assumes that both of the variables (position and velocity) are random and Gaussian distributed [48].

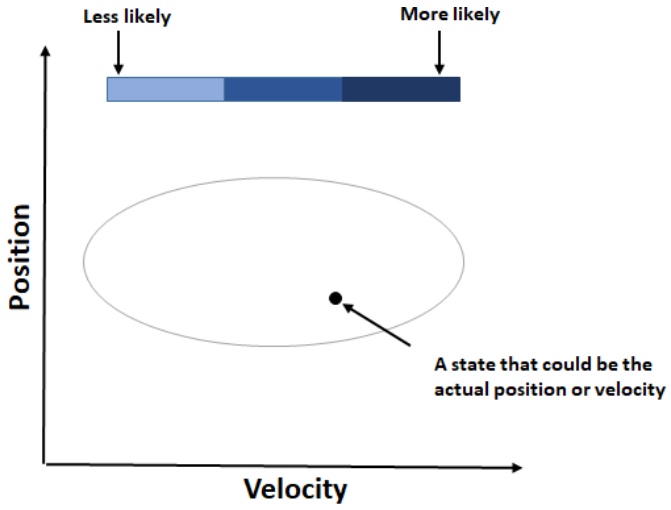



Figure 3.25: Likeness of events [48]

Each of the variables has a mean value with a symbol **μ**, this is the center of the random distributions and it is also the most likely state. It also has a variance with a symbol $\boldsymbol{\sigma^2}$, this is the uncertainty of both components (as it will be shown on Figure 3.26) [48].

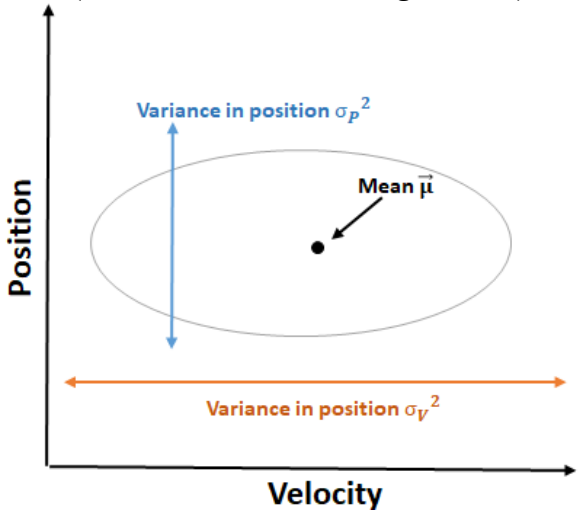



Figure 3.26: Elements of variance uncorrelated [48]

On this particular case, it can be seen that velocity and position are *uncorrelated*, this means that the state of one of the variables does not provide information about what the other variable might be. However, this is not always the case, as it can be seen on Figure 3.27 position and velocity are *correlated*. In other words, the likelihood of observing a particular position depends on the velocity the robot might have. This makes sense when estimating a new position based on an older one, if the velocity was high, the robot *probably* moved farther, thus the new position will be more distant. On the other hand, if the robot is moving slowly it most likely won't get as far [48].

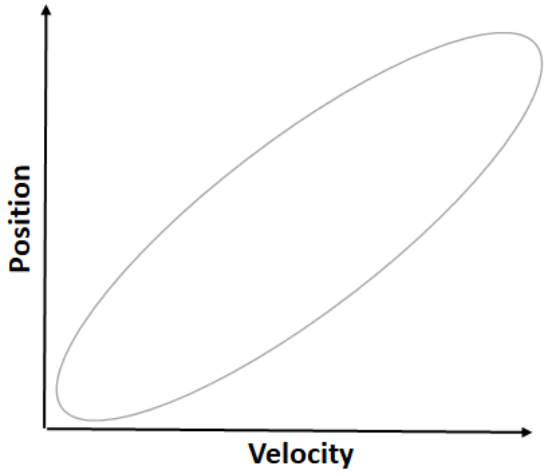



Figure 3.26: Correlated elements [48]

This kind of relationship is beneficial and important to keep track off, for the simple fact that it gives more information to the model. In other words, one measurement tells of something about what the others could be. That is the goal of the Kalman filter, get as much information from the uncertain measurements in order to eliminate the uncertainty (as much as possible). This correlation will be captured by the *covariance matrix*, each element of the matrix $\sum i\, j$ is the degree of correlation between the *ith* state variable and the *jth* state variable. The covariance matrix is symmetric, so it does not matter if you changed $i$ and $j$. Since covariance matrices are often labelled "$\sum$", their elements are "$\sum i\, j$" [48].

Now that the elements have been defined, the next step is to describe the problem with matrices (as the Kalman filter uses matrices for its model). We will be using a Gaussian blob that will provide 2 pieces of information (2 rows) at time *k.* The best estimate that we have come out with will be called $\hat{x}_k$, and its covariance matrix will be called $P_k$[48].

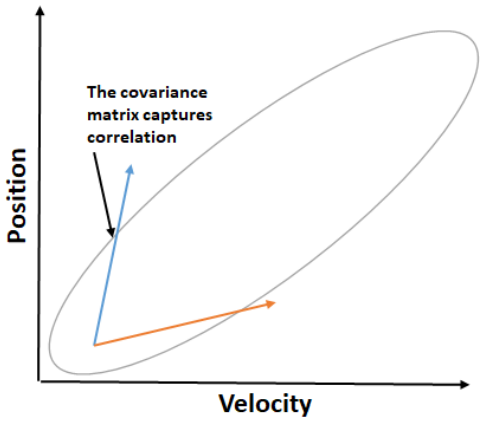



Figure 3.27: Correlated elements [48]

These 2 matrices contain position and velocity, but the states can contain any number of variables, and represent anything the user wants to.

$\hat{x}_k = \begin{bmatrix} position \\ velocity \end{bmatrix}$ $P_k = \begin{bmatrix} \sum pp & \sum pv \\ \sum vp & \sum vv \end{bmatrix}$ [48]

Now, we need to look at the *current state* (time *k-1*) in order to predict the next state at time *k.* The actual state (the real one) is unknown, however, the prediction function does not care about that. It will work on all of the possible states, and give back a new distribution (shown in figure 3.28).

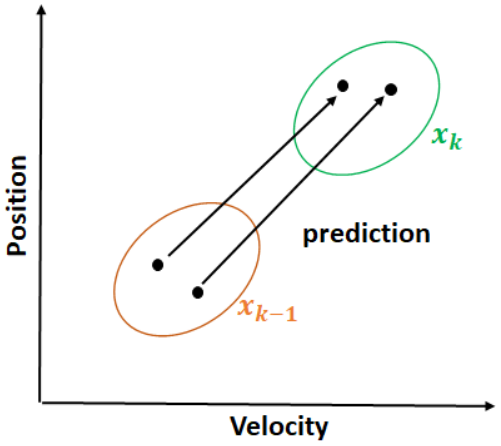



Figure 3.28: Predicted states [48]

This can be represented as a matrix called $F_k$. This matrix will take *every point* in our original estimate and move it to a new predicted location. This would be where the system would move if the original estimate was the right one (Figure 3.29).

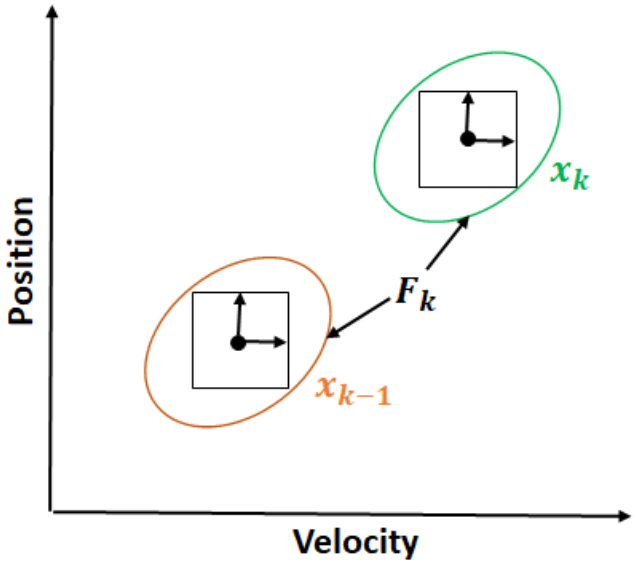



Figure 3.29: $F_k$ predicted states [48]

This basic kinematic formula will explain how to use the matrix to predict position and velocity for the next position:

$p_k = p_{k-1} + \Delta t\, v_{k-1}$ and $v_k = v_{k-1}$

Thus the matrix will look as follows:

$\hat{x}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \hat{x}_{k-1} = F_k \hat{x}_{k-1}$ [48]

Now we have a *prediction matrix* that will give us the next state, but the covariance matrix is still missing; this will require another formula. The following equation will be obtained for every point in a distribution by multiplying the matrix A by the covariance matrix ∑.

$Cov\,(x) = \boldsymbol{\Sigma}$

$Cov\,(\boldsymbol{A}x) = \boldsymbol{A\Sigma A}^T$

Combining with the previous equation, the following is obtained:

$\hat{x}_k = F_k \hat{x}_{k-1}$

$P_k = F_k P_{k-1} F_k^T$ [48]

There are some changes that are not related to the state itself, these are outside "world" effects on the system. For example, the robot navigation software might issue a command to turn the wheels or to stop. If we have access to this additional information of the outside world, its possible to add it to a vector that will be named $\overrightarrow{u_k}$. We will process this information and add it to the prediction as a *correction*. For our robot example, we know the expected acceleration *a* due to the throttle settings of the control commands, from basic kinematics we obtain the following formula:

$$P_k = P_{k-1} + \Delta_t v_{k-1} + \frac{1}{2} a \Delta_t^2$$

$$v_k = v_{k-1} + \frac{1}{2} a \Delta_t$$

The matrix form of these equations are as follows:

$$\hat{x}_k = F_k \hat{x}_{k-1} + \begin{bmatrix} \frac{\Delta_t^2}{2} \\ \Delta \end{bmatrix} \text{a} = F_k \hat{x}_{k-1} + B_k \overrightarrow{u_k} \text{ [48]}$$

$B_k$ is called the *control matrix* and $\overrightarrow{u_k}$ is the *control vector*. If the system is very simple or has no external influence, these can be omitted. So what happens if the states evolve around external forces that are unknown? As mentioned, the robot could slip or bumps on the ground could slow it down. It's difficult to keep track of these things, and if any of it happens, the prediction could be off since those extra forces would not be accounted for. To fix this, we will model the uncertainty associated with the outside world (the things we cannot keep track of) by adding some new uncertainty after every predicted step:

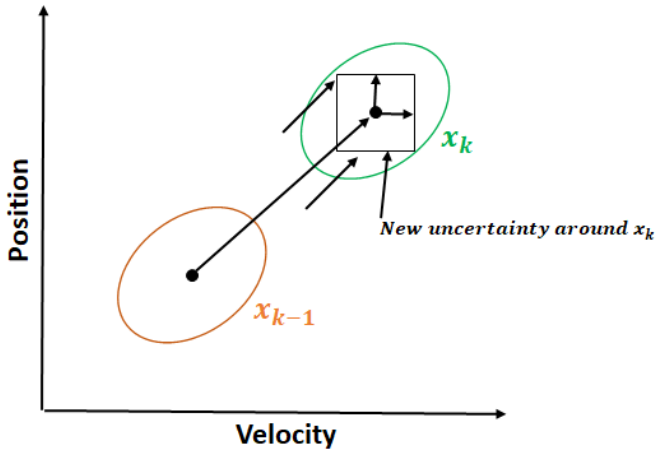



Figure 3.30: "World" uncertainty [48]

As mentioned, every state in the original estimate could have moved to a *range* of states (remember interval computations). This can be interpreted as each point in $\hat{x}_{k-1}$ moving to somewhere inside the Gaussian blob with a covariance $Q_k$. Or in other words, the untracked influences (noise) is associated with covariance $Q_k$.

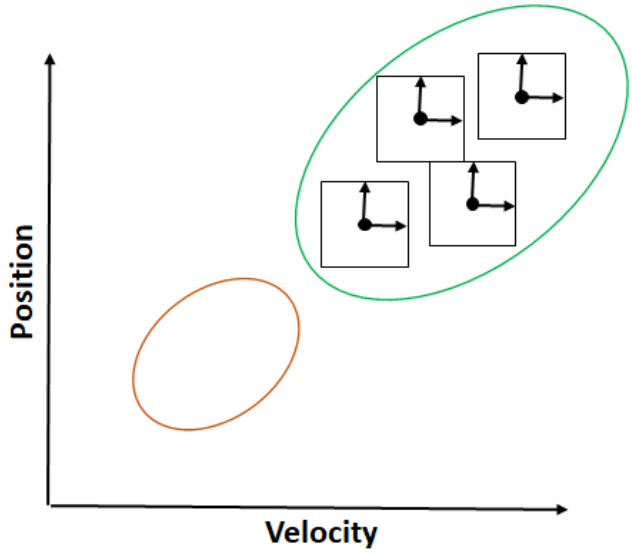



Figure 3.31: Covariance of uncertainty [48]

This will produce a new Gaussian blob, with a large covariance (range) but with the same mean value as before [48].

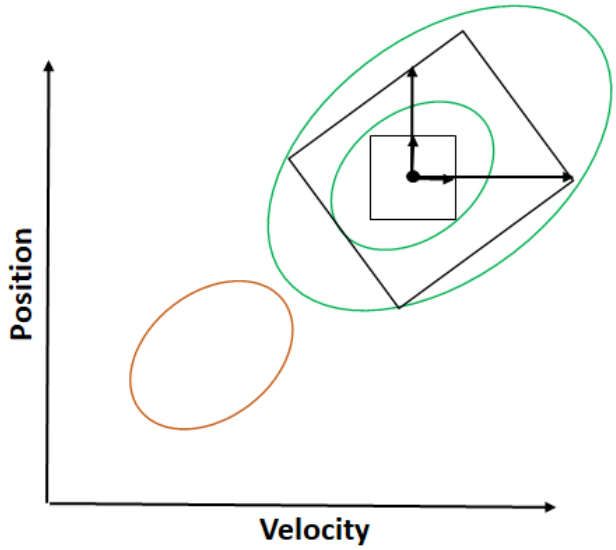



Figure 3.32: Covariance of uncertainty mean [48]

The expanded covariance can be obtained by adding $Q_k$, this will give the complete expression for the prediction step, giving us the following formulas:

$\hat{x}_k = F_k\hat{x}_{k-1} + B_k\overrightarrow{u_k}$

$P_k = F_kP_{k-1}F_k^T + Q_k$

This is the new best estimate, it is a prediction made from the previous best estimate plus a correction for known external influences. And the new uncertainty is predicted from the old uncertainty, with the addition of uncertainty from the environment. Now that we have a fuzzy estimate of where the robot might be, given $\hat{x}_k$ and $P_k$, the next step is to get data from our sensors [48].

The number of sensors that give information about the state varies from application to application. For the time being we do not care what they are measuring (although, in this case we know one sensor is reading position while the other is reading velocity). However, the important concept to gather from here is that each sensor gives us indirect information about the state, in other words, the sensors operate on a state and produce a set of *readings*.

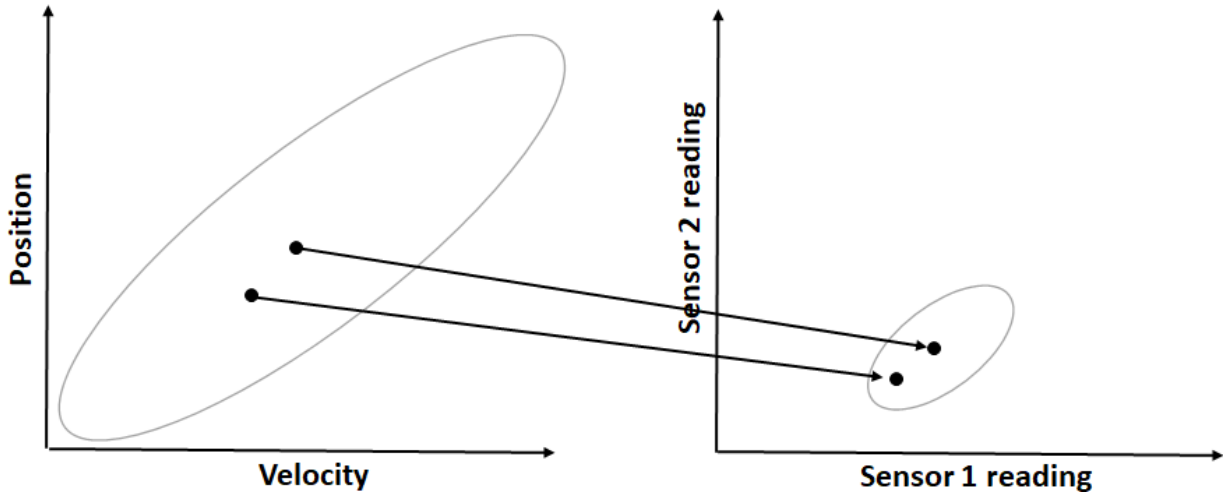



Figure 3.33: Readings from sensors [48]

It is clear that the units and scale of the outputs are not the same (after all one is position the other one is velocity). Nevertheless, a model of the sensors will be created with a matrix called $H_k$. We will figure out the distribution of the sensor readings with the following equations:

$\vec{\mu}_{expected} = H_k \hat{x}_k$

$\sum expected = H_k \hat{x}_k$

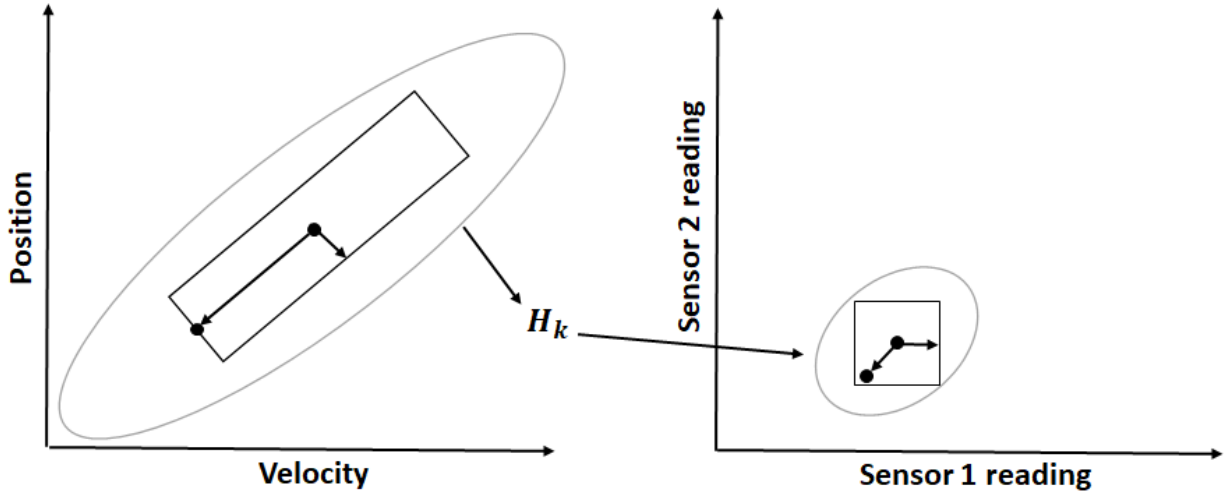



Figure 3.34: Distribution from sensors [48]

Kalman filters are great for dealing with the noise from the sensors. As it is known, the sensors are somewhat unreliable (as explain through this thesis), and every state in our original estimate might result in a range of sensor readings. From each reading we observe, we can guess that our robot was in a particular state. However, because there is uncertainty, some states are much more likely than others to produce the reading [48].

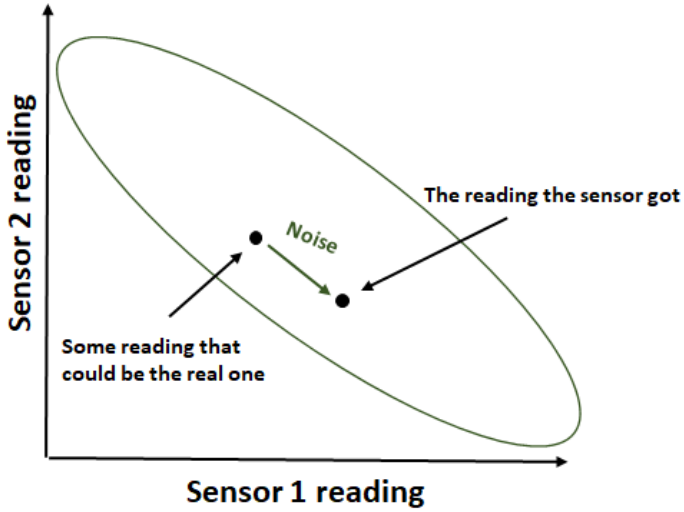



Figure 3.35: Noise from sensors [48]

This is the covariance (the noise of the sensor), called $R_k$. The distribution has a *me*an that is equal to the reading that was observed, it will be called $\vec{z}_k$. We now have two Gaussian blobs, one that is surrounding the predicted state and one that is surrounding the sensor reading.

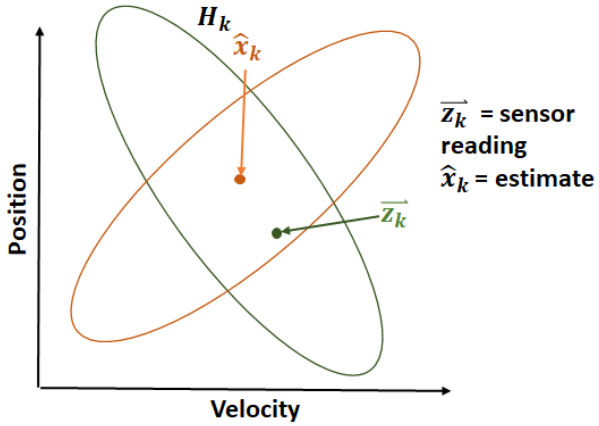



Figure 3.36: Gaussian blobs [48]

The question now arises, what is the most likely state? For the possible readings $(z_1, z_2)$, there are two possibilities: The sensor reading $\vec{z}_k$ is an incorrect measurement of $(z_1, z_2)$, and the probability that the previous estimate thinks the reading $(z_1, z_2)$ should be used. Based on these two probabilities, we can know the chance the both are true by multiplying them together. In other words, we take the two Gaussian blobs and multiply them [49].

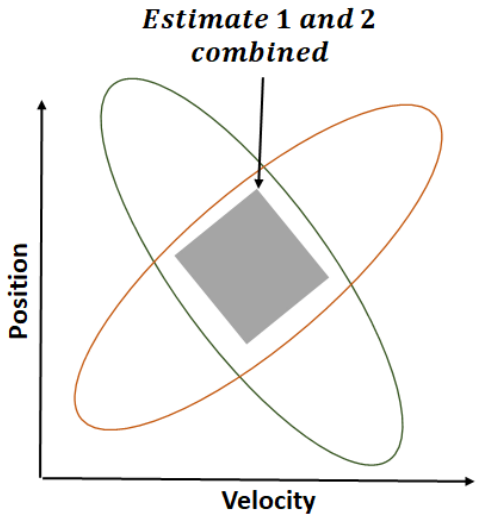



Figure 3.37: Probabilities combined [48]

The grey area in Figure 3.37 is the *overlap,* this is the region where both blobs are likely to happen. This is a lot more precise than either of our previous estimates. The mean of this distribution is the configuration for which *both estimates are more likely to happen,* this is the **best guess** of the true configuration with all the information that we have gathered so far. If you multiply two Gaussian blobs with separate means and covariance matrices, you will get a *new* Gaussian blob with its own mean and covariance matrix. Figure 3.38 shows this new Gaussian blob [49].

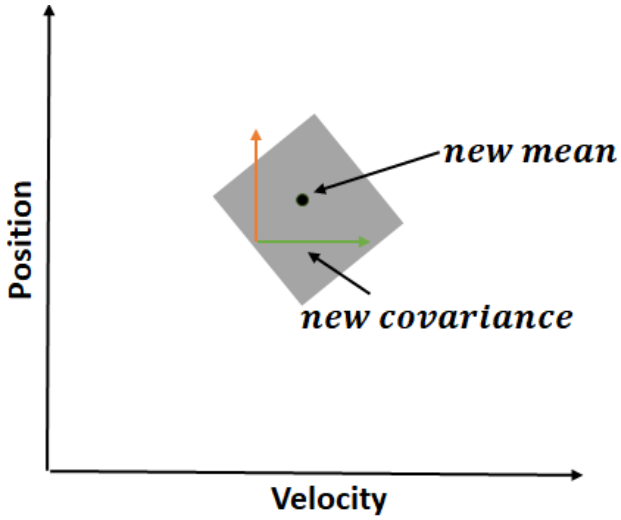



Figure 3.38: Gaussian blob formed by probabilities combined [48]

In order to find the formula to combine the Gaussians we will look at an example in a simplified form, one dimension [48]. This one dimension Gaussian bell curve with a variance that will help us defined a very important term in the Kalman filter; the Kalman gain [48] [49].

A 1 dimension Gaussian bell curve with variance $\sigma^2$ and mean μ is defined by the following equation: $N\ (x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

We are interested in knowing what happens when two Gaussian curves are multiplied together. On Figure 3.39 the blue curve represents the (unnormalized) intersection of the two Gaussian populations, which are the green and orange curves. They give us the following equations:

$\mathrm{N}\ (x, \mu_0, \sigma_0)\ .\ N\ (x, \mu_1, \sigma_1) = \mathrm{N}\ (x, \mu', \sigma')$

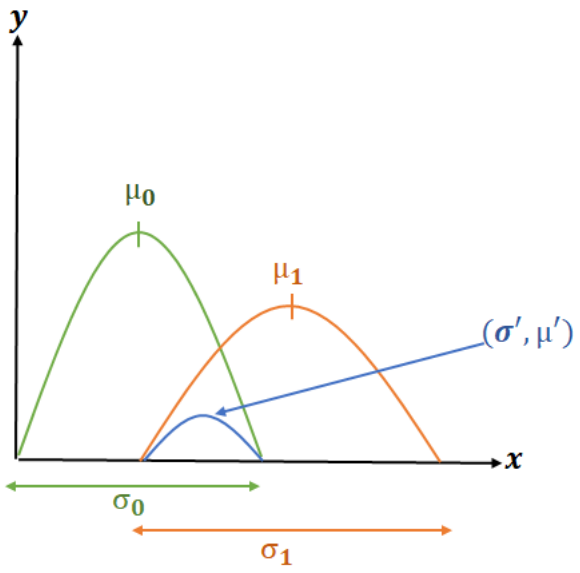



Figure 3.39: Combining Gaussian bell curves [48]

Combining this equation with the previous one (and normalizing it at the same time) we will obtain a combined equation:

$$\mu' = \mu_0 + \frac{\sigma_0^2(\mu_1-\mu_0)}{\sigma_0^2+\sigma_1^2}$$

$$\sigma'^2 = \sigma_0^2 + \frac{\sigma_0^4}{\sigma_0^2+\sigma_1^2}$$

These equations can be simplified by factoring out part of the equations and naming it **k:**

$$k = \frac{\sigma_0^2}{\sigma_0^2+\sigma_1^2}$$

$$\mu' = \mu_0 + k\,(\mu_1 - \mu_0)$$

$$\sigma'^2 = \sigma_0^2 + k\sigma_0^2$$

Since the Kalman filter works in matrix form, let us re-write the equations in matrix form. Σ is the covariance matrix of a Gaussian blob, and $\vec{\mu}$ is the mean along each axis, we obtain the following:

$$\mathrm{K} = \Sigma_0(\Sigma_0 + \Sigma_1)^{-1}$$

$$\vec{\mu}' = \vec{\mu}_0 + \mathrm{K}\,(\vec{\mu}_1 - \vec{\mu}_0)$$

$$\Sigma' = \Sigma_0 - K\Sigma_0$$

*K* is a matrix called the **Kalman gain.** Let us go back to our robot equation and use this Kalman gain in our equation to finish our simulation of the Kalman filter [48] [49].

Going back to our robot model, we have two distributions: the predicted measurement with $(\mu_0, \Sigma_0) = (H_K\hat{x}_k, H_kP_kH_k^T)$, and the measurement from the sensors with $(\mu_1, \Sigma_1) = (\vec{z}_k, R_k)$. Multiplying the Gaussian blobs, we will obtain an equation similar to the previous example:

$H_K\hat{x}'_k = H_K\hat{x}_k + \mathrm{K}\,(\vec{z}_k - H_K\hat{x}_k)$

$H_KP'_kH_K^T = H_kP_kH_k^T - KH_kP_kH_k^T$

The Kalman gain is defined as:

$K = H_kP_kH_k^T(H_kP_kH_k^T + R_k)^{-1}$

By simplifying:

$\hat{x}'_k = \hat{x}'_k + K'\,(\vec{z}_k - H_K\hat{x}_k)$

$P'_k = P_k - K'H_kP_k$

$K = P_kH_k^T(H_kP_kH_k^T + R_k)^{-1}$ [48]

These equations give us the final update step with all the elements necessary for the Kalman filter. $\hat{x}'_k$ will be the new best estimate, and along with $P'_k$, will be added into the prediction state or update as many times as required. Figure 3.40 will provide an overview and summary of the Kalman filter [48] [49].

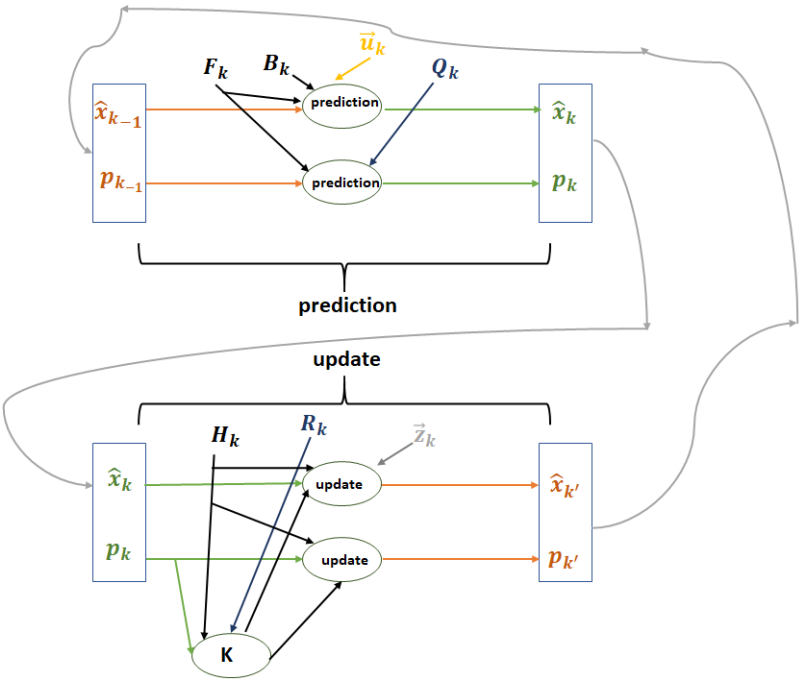



Figure 3.40: Kalman filter summary [48]

As it can be seen the Kalman filter equations are not very difficult to understand, however, the hard part is to create an accurate model of the system and this becomes even harder when using nonlinear systems. For nonlinear systems, the *extended kalman filter* can be used, this works by linearizing the predictions and measurements about their mean, the extended kalman filter is beyond the scope of this thesis, but it is recommended to further explore it for IMU related projects. In addition, there are two more filters that are worth discussing that can be applied to the IMU; the Mahony's filter by Sebastian O.H Madwick and the data fusion DMP by InvenSense. The Mahony's filter works by adding the magnetometer output data into the filter and using quaternion's equations. The DMP uses data fusion on the IMU chip itself, since this data fusion algorithm occurs on the MPU-6050 chip it frees up processor power. These two filters are powerful options that can work to improve data output on the sensor, however, their equations are beyond the scope of this thesis and will not be further touched.

On the next chapter, graphical options for the output of the IMU data will be explored, as well as filtering models implemented on MATLAB. In addition, results will be shown to prove the effect of the filters.

# CHAPTER 4: FILTERING MODELS

Now that the filters have been explained, it is time to show their effect. Most people are visually oriented, thus, it makes sense to use graphs and graphics to display the outputs obtained. In the case of the IMU output data, there are several methods that can be used to show a graphic representation of the data obtained from the various sensors. On the next section we will go over the ones that were used for this thesis. These options are user friendly, open source (free), and do not require a lot of computing power. For these reasons they should work on virtually any computer.

## 4.1 GRAPHICAL OPTIONS FOR ARDUINO IMPLEMENTATION

The first option, as it could be imagined, is to use Arduino. The Arduino has a function called serial plotter, and like its name suggests, it allows the user to plot the output data (given the user has used a Serial.print command).

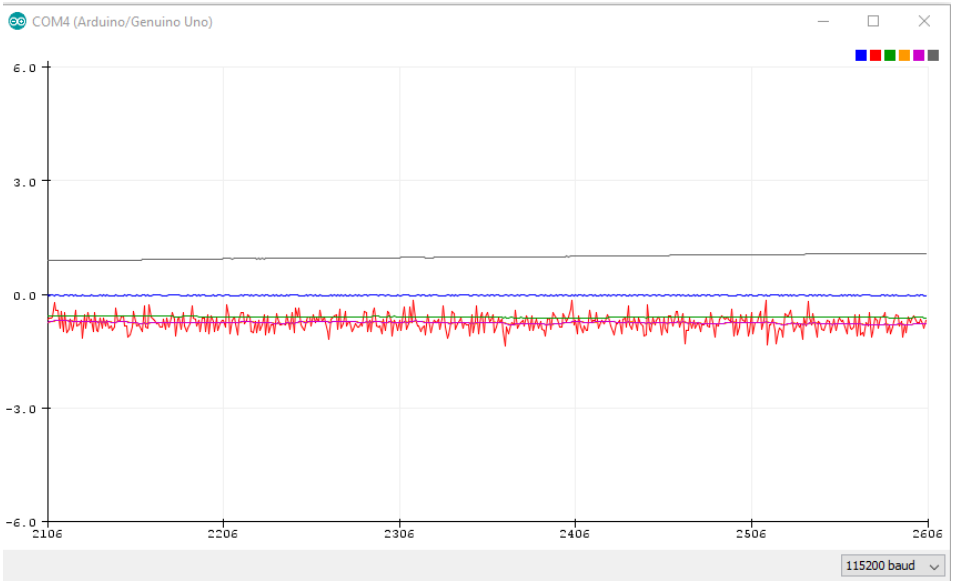



Figure 4.0: Serial plotter Arduino

As it can be seen on Figure 4.0, the Arduino plotter is capable of plotting several output values at once. This is very convenient, as it allows to natively graph serial data from the Arduino to the computer in real time. It can also work to debug a code as the user can see if it is wrong correctly

or not as intended. Unfortunately, there are a couple of key limitations with the Arduino plotter. The user is not able to save the plotted data (snipping tools was used to show this image), there is no way to change the X-axis scale/time required, and the user is not able to use Serial plotter and Serial monitor at the same time. Nevertheless, it is still a powerful tool to use for quick editing and troubleshooting the code.

Before going to the next option, there is an application that is useful for saving the output of a code without the need to use an external data logger; it is called PuTTytel. PuTTytel is a free and open source terminal emulator, in other words, it works as a serial monitor just like the Arduino. In addition to this, it can also use several other protocols (but this is beyond the scope of this thesis). The main advantage of using PuTTytel is that is able to log the data that the serial monitor is displaying. PuTTytel is very user friendly and straightforward, nevertheless, we will go over a few settings that the users should understand in order to use the log features.

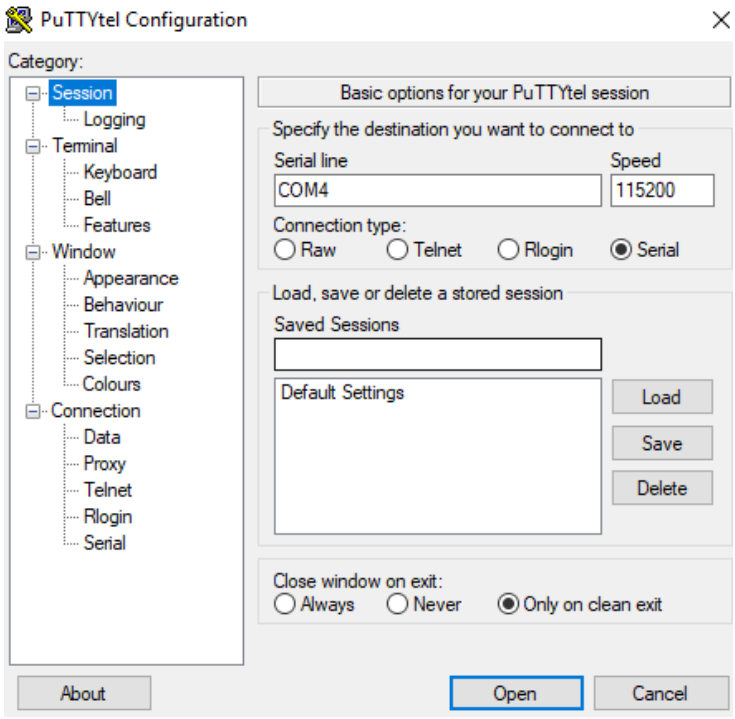



Figure 4.1: PuTTytel settings

The first to keep in mind is the serial line, which is basically the port the computer is using to communicate with the Arduino. To know which port the Arduino is using (On windows), go to Device Manager and then select Ports, the COM the Arduino is using should appear there.

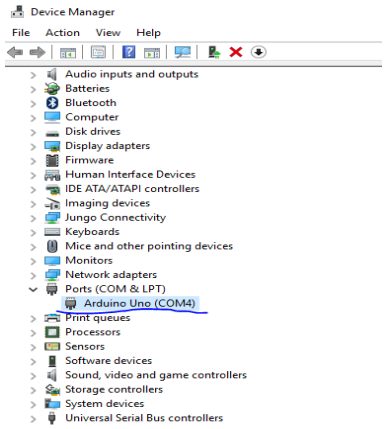



Figure 4.2: Device Manager COM ports

The next step is to choose the right “speed”, the speed is the baud rate on the Arduino. The baud rate is the modulation rate per second, or in other words, pulses per second. The speed is determined by whatever baud rate you are using on the Arduino code. The user should select “Serial”, since the Arduino uses serial communication. The last step is to go to the Logging section and select Printable output, the user then will select a folder and a name that will display the output data. Be sure to select “Ask the user every time” to avoid deleting old data. And as simple as that, the user should be able to save this data and use a program like Excel or Minitab to make useful plots.

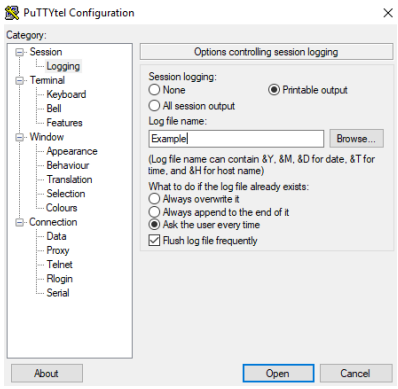



Figure 4.3: PuTTytel logger settings

The next option is called Processing. Processing is a flexible software sketchbook and a language for visual representation. It is an integrated development environment (IDE), just like the Arduino. In addition, it is open source (free) and user friendly (processing uses a simplified version of Java). Finally, just like Arduino, there are libraries available online that can help the user free of cost.

Figure 4.4: Processing example

On Figure 4.4 we can see the gyroscope (yellow), accelerometer (blue), and the combined (green) outputs displayed in a graphical representation. This is useful to show data in real time, and show the effects of the filter in a way that everyone can understand (regardless if they are familiar with filters or not). Processing can use a variety of objects to represent the output data, on Figure 4.5 the Yaw, Pitch, and Roll data are represented by a plane.

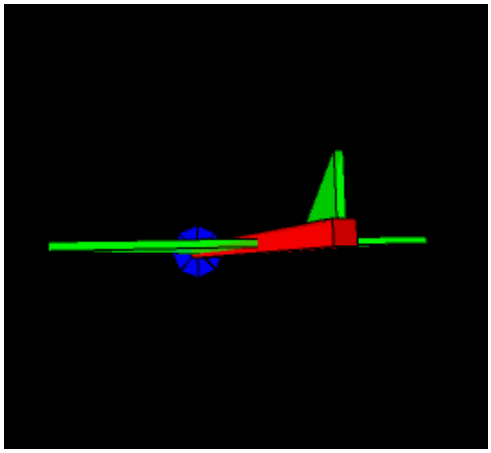


Figure 4.5: Yaw, Pitch and Roll representation

For the AltIMU-10 a Python script was used to display the data (in addition to Processing). Python is a programming language that is simpler to use than Java or C++, thus it makes a good choice to beginners. Just like the MPU, this code graphically represents the Roll, Pitch, and Yaw angles and uses a cube to represent movement.

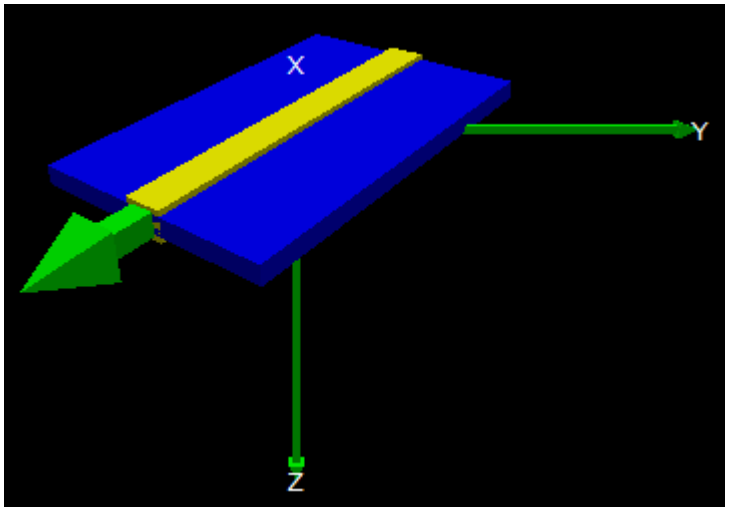



Figure 4.6: Cube example in Python

Python, Processing, and the Arduino Serial Plotter are useful if we want to see our data in real time, but what about graphically displaying data from a previous test? The micro SD logger can record that data for us, or we can use the Bluetooth slave to connect to the PuTTytel and log the data. Of course we could graph this data, but what if we wanted to graphically represent this data? For example, let's pretend we attached an IMU sensor to our little robot and we ordered it go straight to the target. This should normally take about 5 seconds but we noticed that it took 10, we could certainly look at the data, but a graphic representation would show that the robot got moved abruptly to the left without needing to look at all the numbers. Using this method it is possible for anyone to understand the events without the need to understand how the filter works. MATLAB was chosen for this task, in the next section a method to take logged data and represent it visually will be discussed. The sensor that was selected was the MPU-6050, the reason is that it is more accurate, thus, it will display more powerful results. However, the user can use data output

from the AltIMU-10 as well and run it on the MATLAB code, given that they change the code to match the expected outputs.

## 4.2 MATLAB IMPLEMENTATION OF MODELS AND RESULTS

As previously mentioned, it is useful to graphically represent our output data. I adapted the Arduino codes to give us the data that we are interested in, and then record it using the data logger or PuTTytel. After that, a MATLAB code will read the data in a text file and "plot" the results; this is known as post-processing data. Just like Processing (the program), MATLAB is able to create figures to represent output data. A cube was chosen to plot the Arduino data, this is a simple representation that makes sense (since the sensor is a square).

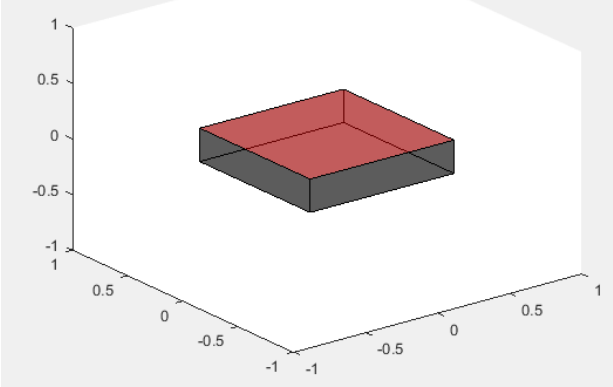



Figure 4.7: Cube example in MATLAB

The data that will be "plotted" is entirely up to the user and application, however, for this particular case it makes sense to use the output of Yaw, Pitch, and Roll data obtained from the sensors. As an example, I chose the following output values, these are angles from different outputs in x, y, and z format. Of course, in a real application it would make more sense to only use the filtered data. The example output printed came out like this:

Accelerationangles:-1.42,-0.37,0.00#Gyrounfiltered:-0.00,-0.00,0.00#Gyrofiltered:-0.06,-0.02,0.00#Fil:-0.11,-0.03,0.00.

The MATLAB code would then read the x, y, and z angles of any of the outputs and the cube would replicate the movement of the sensor. This will be used to shown the effect of the filters as a visual representation, depending of the "set" of angles that user selected.

In addition to this, MATLAB was also used to perform the Fourier Transform on the g-force outputs (since this data will be usable for a possible application of the IMU), as well as using the cube to display the DMP data fusion (best output data) both in real time and for post-processing. More information on each of the codes and results will be further explained on the following sections.

### 4.2.1 Raw data

Now, let us take a closer look at the effect of the filters. In order to visualize this effect we will start with the raw output data (not filtered), since this is the "worst case". Processing, Arduino Serial Plotter, and the MATLAB codes are a great visual aid, however, most of these are real time and do not give quantifiable data (also some of them are impossible to convey in a static environment, such as this thesis paper). For this reason, graphs will be used to show their effectivity. However, there will be a discussion on the codes available for raw data as well.

The first thing to discuss is the setup. The smoothest surface possible should be used for taking data. As mentioned, marble title is a very straight, for this reason it was chosen as the surface for taking data.

Figure 4.9: Table with marble title

To ensure its flatness the AngleCube was used, of course, there is a small error in the AngleCube, but for characterizing the sensors a higher level of precision is not required. It will be assumed that the surface is completely flat for measurement purposes.


Figure 4.10: AngleCube measurement

The data was to be taken from the 3 IMUs (MPU-6050 10 DOF, MPU-6050 6 DOF, and AltIMU-10 DOF) for a period of 5 minutes. The data will then be plotted using excel. Since it is known that accelerometer data cannot compute Yaw; only the X and Y outputs will be evaluated. For the gyroscope, all 3 axes will be evaluated. In addition to this, each of the sensors (accelerometers and gyroscope) shortcomings will be evaluated, this means drifting for gyroscope and sensitivity to force for the accelerometer.

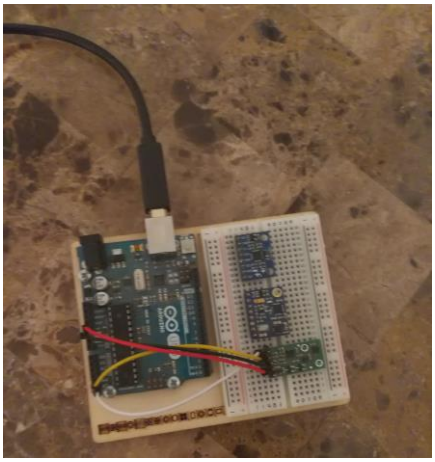


Figure 4.11: Set-up used for taking data

The following tests were performed on the IMU sensors:

1) *No movement on the sensor*; in order to see the effect of drift or random noise on the sensors. This is done on both accelerometer and gyroscope outputs. Raw (AltIMU-10, MP.-6050), complementary (AltIMU-10, MPU-6050), magnetometer Yaw (AltIMU-10, for the Z-axis only), and DMP (MPU-6050) data was taken. The sensor will be standing still for a period of 5 minutes without any movement or external forces applied (at least intentionally).
2) *Taps every minute while standing still*; this will show the effect of force sensitivity on the sensors. This is done on both accelerometer and gyroscope outputs. Raw (AltIMU-10, MPU-6050), complementary (AltIMU-10, MPU-6050), magnetometer Yaw (AltIMU-10, for the Z-axis only), and DMP (MPU-6050) data was taken. The sensor will be standing still for a period of 5 minutes and I will tap on the table (close to the sensor) to introduce noise.
3) *Roll*; the IMU will be moved from the X angle while small vibrations are introduced (by shaking hand). This is done on both accelerometer and gyroscope outputs. Raw (AltIMU-10, MPU-6050), complementary (AltIMU-10, MPU-6050), and DMP (MPU-6050) data was taken. The sensor will be moved around the X axis (Roll) for about 5 minutes while shaking my hand to introduce noise.
4) *Pitch*; the IMU will be moved from the Y angle while small vibrations are introduced (by shaking hand). This is done on both accelerometer and gyroscope outputs. Raw (AltIMU-10, MPU-6050), complementary (AltIMU-10, MPU-6050), and DMP (MPU-6050) data was taken. The sensor will be moved around the Y axis (Pitch) for about 5 minutes while shaking my hand to introduce noise.
5) *Yaw*; the IMU will be moved from the Z angle while small vibrations are introduced (by shaking hand). This is done only on the gyroscope,

magnetometer, and DMP outputs. Raw gyroscope (AltIMU-10, MPU-6050), magnetometer (AltIMU-10), averaged gyroscope (MPU-6050), and DMP (MPU-6050) data was taken. The sensor will be moved around the Z axis (Yaw) for about 5 minutes while shaking my hand to introduce noise.

Unsurprisingly, while examining the data from the MPU-6050 (10-DOF) and the MPU-6050 (6-DOF), the difference was negligible, thus, results from the MPU-6050 (6-DOF) will not be included. This shows that the sensor is robust to manufacturing differences.

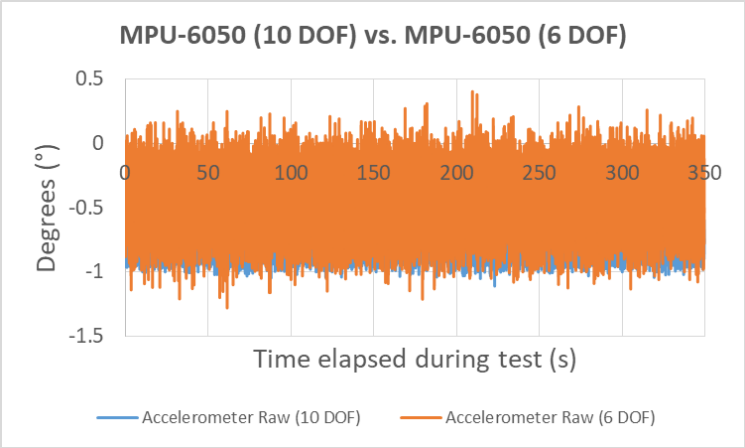



Figure 4.12: MPU Raw 10 DOF vs 6 DOF

Now, let us compare the results from the *No movement on the sensor*; it will start with a comparison between the raw data from the accelerometer (X, Y axes) from the AltIMU-10 and the MPU-6050. It will also show the magnetometer data from the Alt-IMU-10 from the Z axis. All of this data was taken with the sensor completely still for a period of 5 minutes. This will allow the user to get a sense of the shortcomings of the components of the IMU, mainly drift and sensitivity to gravity and other external forces. It will also allow the user to understand the difference between

IMU sensors (not all of them have the same resolution). On Figure 4.13 we will observe the raw accelerometer data from the 2 IMU sensors from the X-axis.

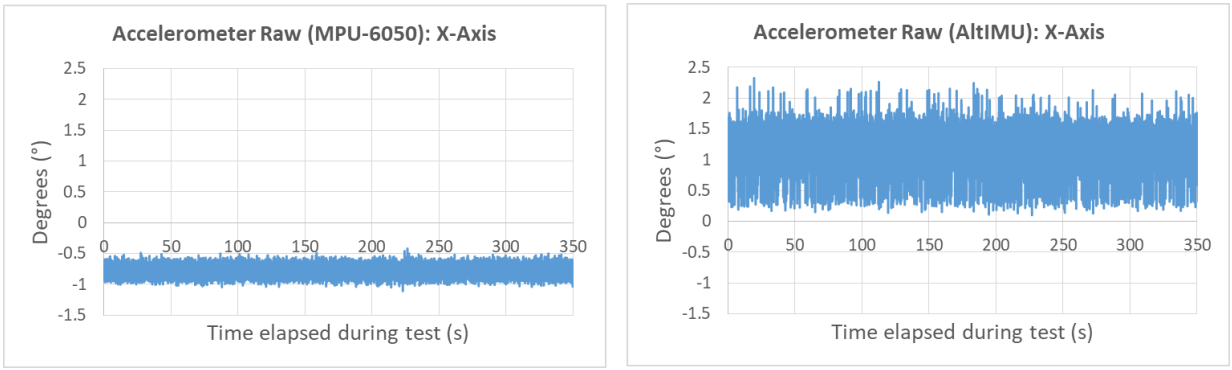



Figure 4.13: Raw accelerometer data X-axis

It can clearly be seen that the AltIMU-10 is more susceptible to noise, as it shows a wider range of deviation. On the Y-axis a similar behavior is observed:

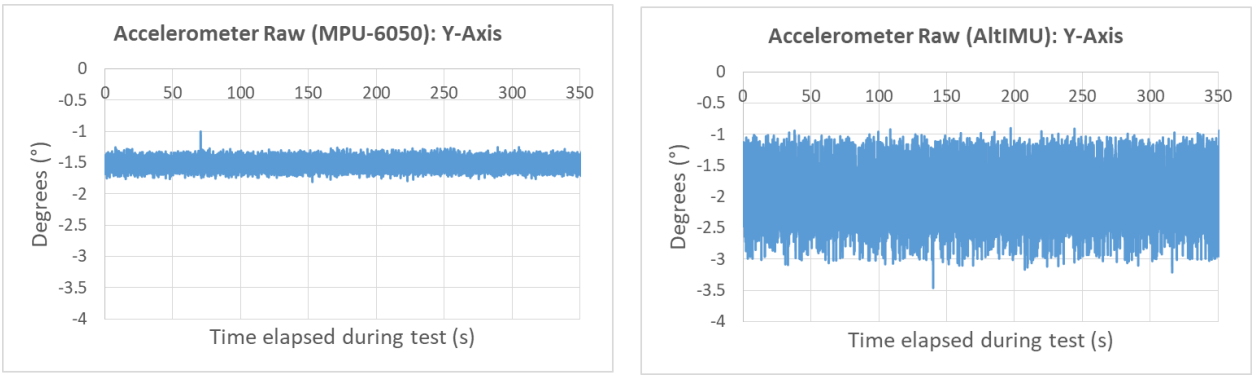



Figure 4.14: Raw accelerometer data Y-axis

The magnetometer data from the AltIMU-10 exhibits a behavior you might expect from a compass, its susceptibility to other electronics. This could be due to other magnetic fields interacting with the magnetometer, so it is susceptible to this kind of noise. Once it finds that point the

magnetometer remains more or less constant. Figure 4.15 will show the output found on the magnetometer Z-axis.

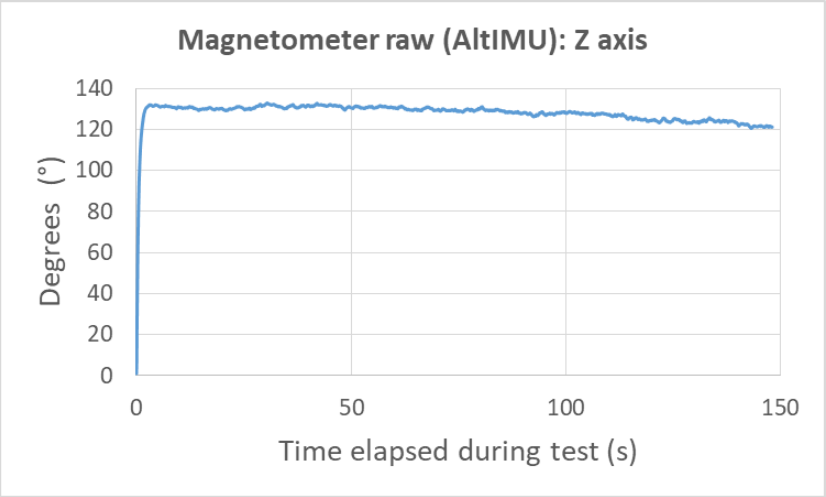
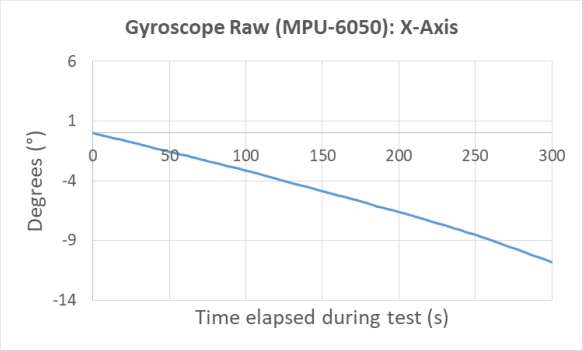



Figure 4.15: Magnetometer data Z-axis

Now, let us take a look at the gyroscope output data of both sensors, drifting is to be expected.



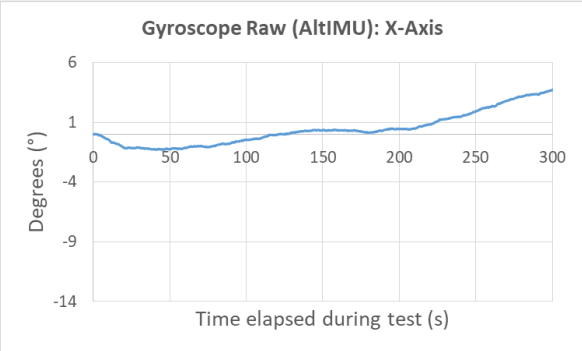




Figure 4.16: Raw gyroscope data X-axis

The AltIMU-10 surprisingly shows less drift than the MPU-6050 for this axis. This may lead us to believe that it will be the case on all of the outputs just like on the accelerometer, however, as we will observe on the next graph, this is not always the case. As the user may deduce, there is no perfect sensor. Depending on the application, one sensor may be preferable over the other. For example if we were to use the output Raw X-axis data, the AltIMU-10 would be the clear choice.

Let us take a look at the Y-axis output of both IMU sensors:

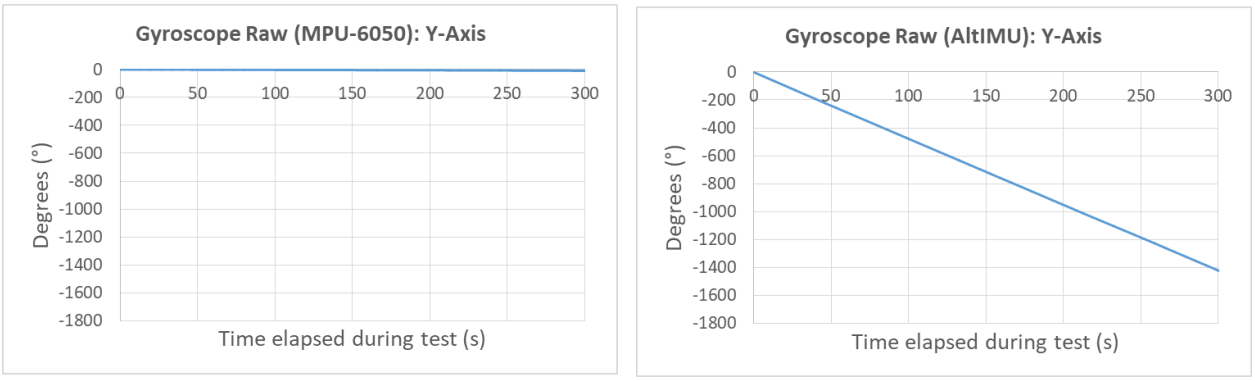



Figure 4.17: Raw gyroscope data Y-axis

Looking at the 2 graphs it may lead to believe that the MPU-6050 is not drifting, however, this is not the case. The MPU-6050 drifted from 0 to -10 similar to the X-axis, however, the AltIMU-10 drifted all the way to -1800. Thus, by comparison the MPU-6050 appears to have no drift by contrast. This reinforces the previous point, no sensor is perfect. The Z-axis will be the last graph from the *No movement on the sensor.*

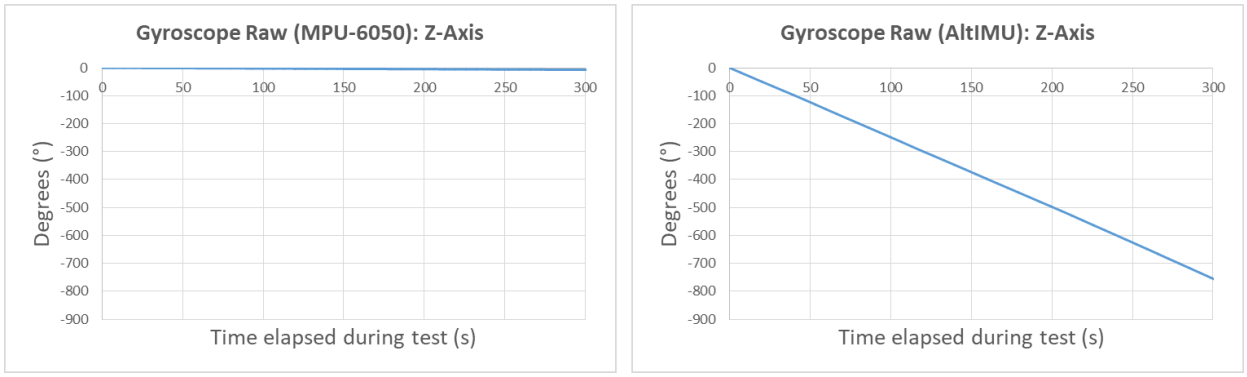



Figure 4.18: Raw gyroscope data Z-axis

Same behavior from the previous axis. Conclusions gathered from the first tests are the following: The MPU-6050 is less susceptible to noise and it confirms that the gyroscope is prone to drifting. Now, let us examine the shortcoming of the accelerometers.

We will compare results from the test "*Taps every minute while standing still*":

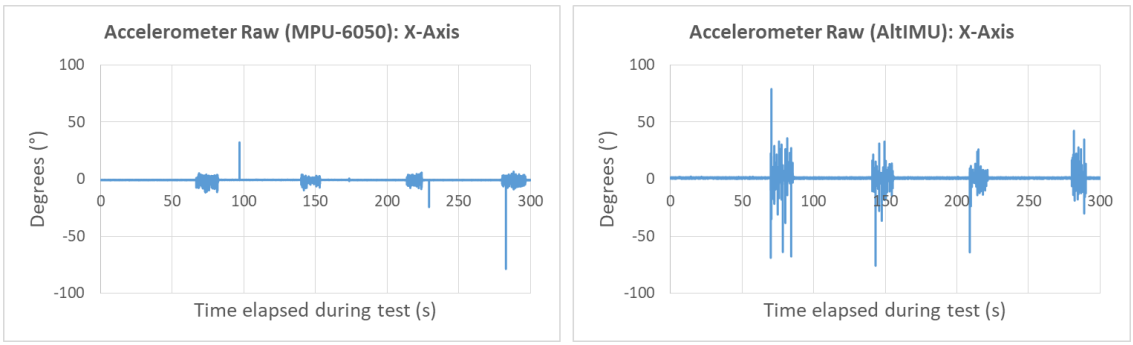



Figure 4.19: Raw accelerometer data X-axis

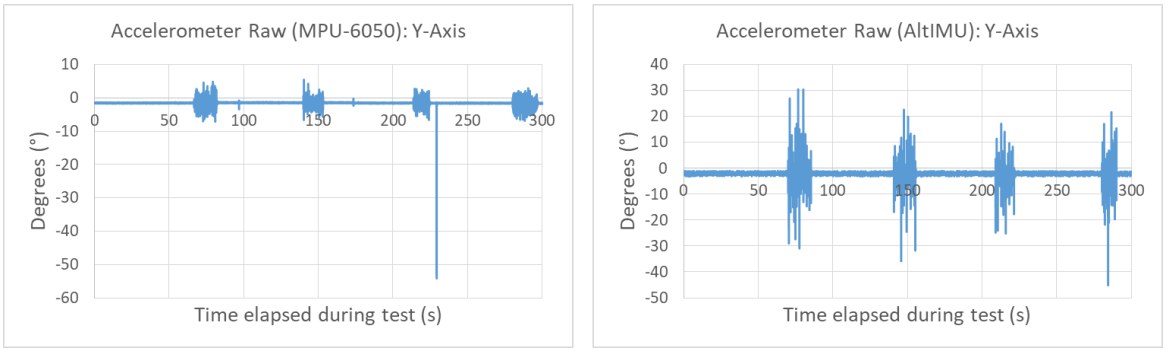



Figure 4.20: Raw accelerometer data Y-axis

Once again, the AltIMU-10 is more susceptible to noise. The gyroscope is not *as* susceptible to force factors, nevertheless we will show the MPU60-50 Y-axis to demonstrate it.

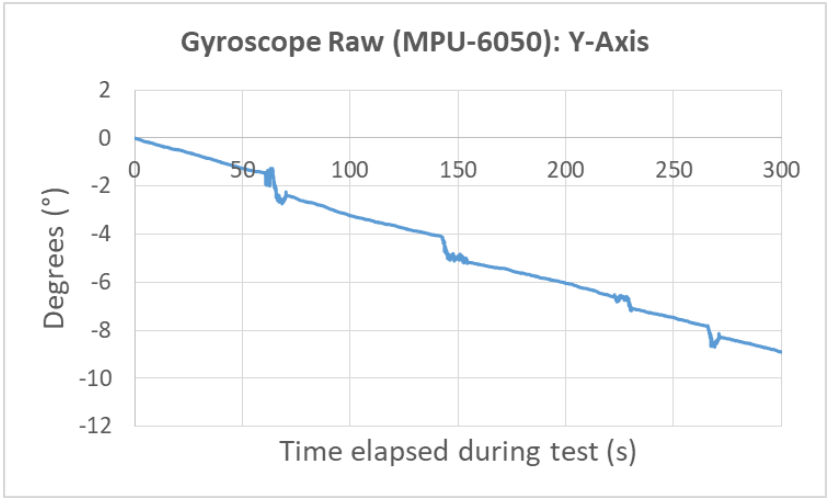



Figure 4.21: Raw gyroscope data Y-axis

As it can be seen, the sensor *does* react to the taps on the table, but the difference is very noticeable. However, as expected the gyroscope continues to drift. Magnetometer data is also impervious to the taps.

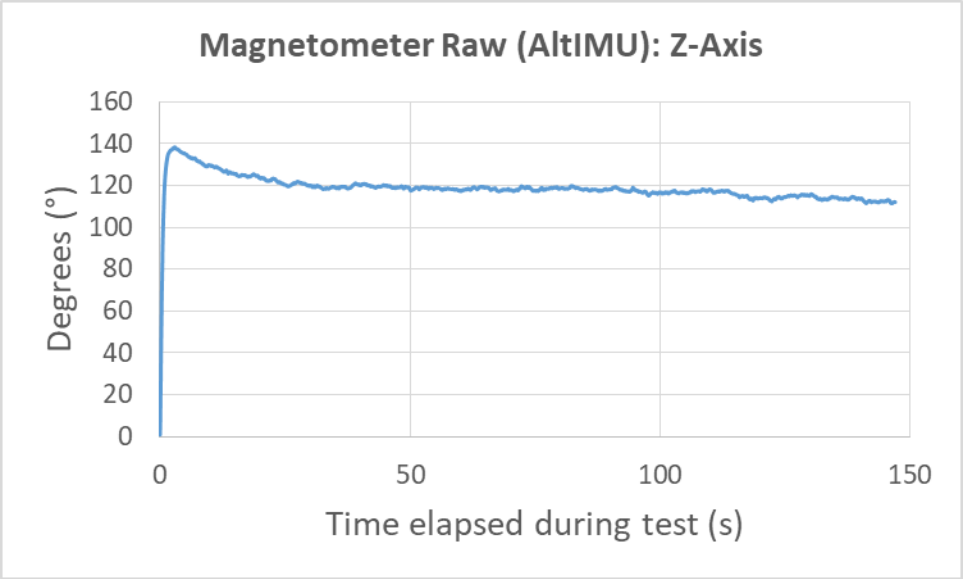



Figure 4.22: Magnetometer data Z-axis tap forces

The conclusion from these tests: once again, it shows evidence that the MPU-6050 is the superior sensor, and proves that the gyroscope is much less sensitive to outside forces. This is the key to the complementary filter, combining the sensor properties to diminish their shortcomings.

Next, we will show the results from the *Roll* and *Pitch* tests from both accelerometer and gyroscope. The MPU-6050 results will be shown only, as the difference is minimum here and it has been proven that the MPU-6050 is superior.

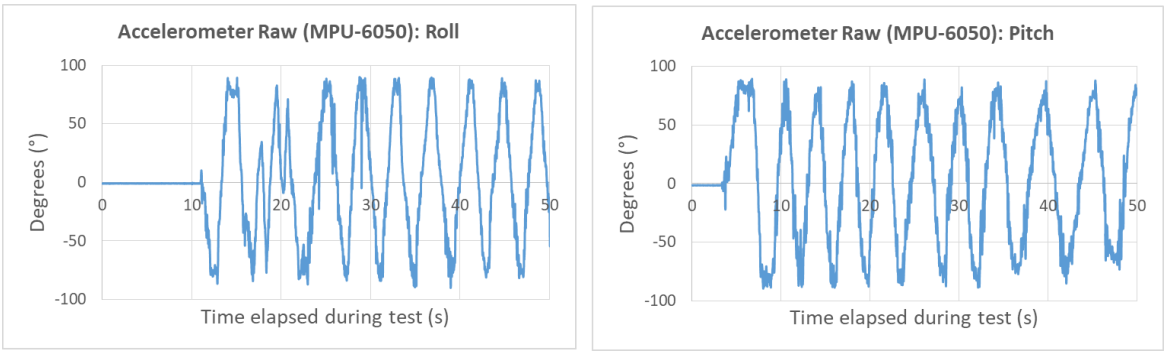



Figure 4.23: Pitch and Roll from accelerometer

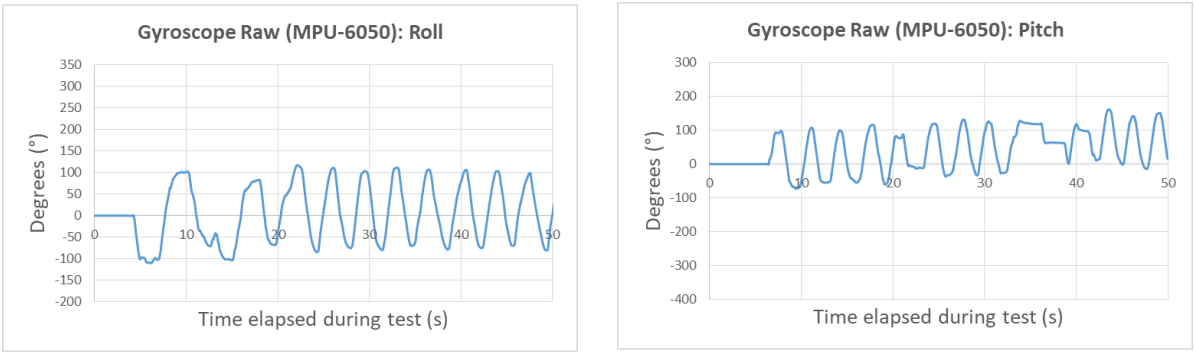



Figure 4.24: Pitch and Roll from gyroscope

Looking at these graphs we can see the same effect, the gyroscope data is much more clean and the accelerometer data is more susceptible to the forces. However, if we increase the sample size, drifting will be visible:

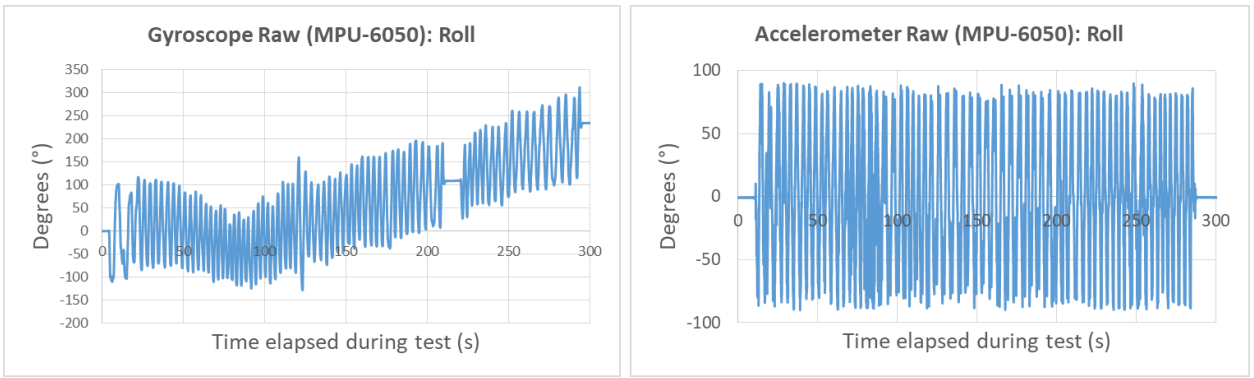



Figure 4.25: Pitch and Roll gyroscope vs accelerometer

Finally, *Yaw* data will be taken from the MPU-6050 gyroscope and the AltIMU-10 magnetometer:

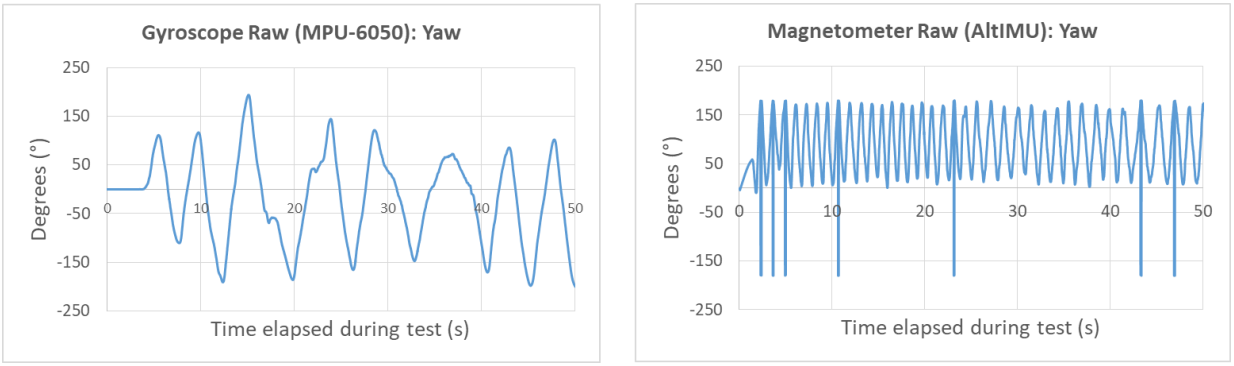



Figure 4.26: Yaw gyroscope vs magnetometer

The magnetometer is more precise and smooth than the gyroscope (however it takes longer to process data), for this reason it is useful for implementing into filters such as the Kalman, Mahony's, and the DMP.

The results from the tests confirmed the theory of both the accelerometers and gyroscopes. It also shows a clearer picture of how the complementary filter improves the output. Results from the complementary filter will be shown on the next section.

### 4.2.2 Complementary filter data

Now that we have seen the tendencies of the sensors, let us see the power of the complementary filter. Only data from the MPU-6050 (10 DOF will be showed), however, it should be noted that the AltIMU-10 complementary filter is very comparable to the MPU-6050 results. Only roll and pitch data are possible to implement with the complementary (as the yaw cannot be

obtained from the accelerometer). The next figures will show the results of all the tests performed on the complementary filter:

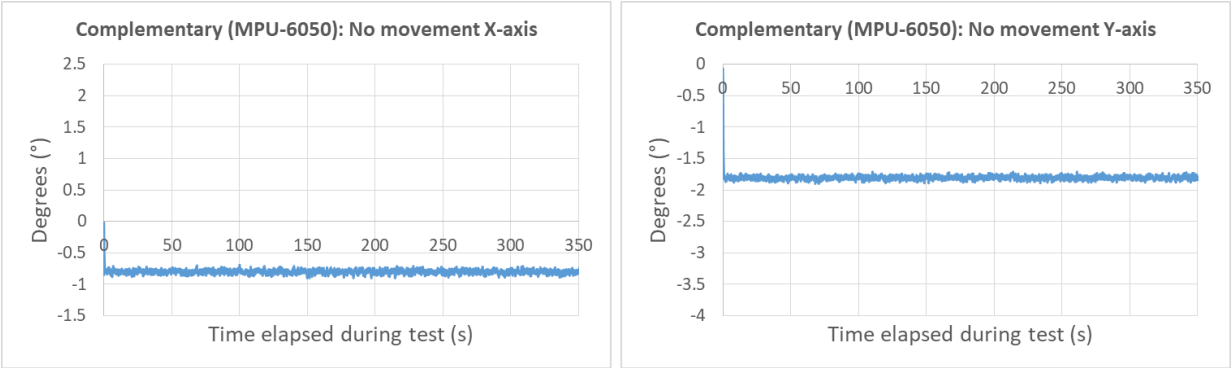



Figure 4.27: Complementary No movement X and Y axis

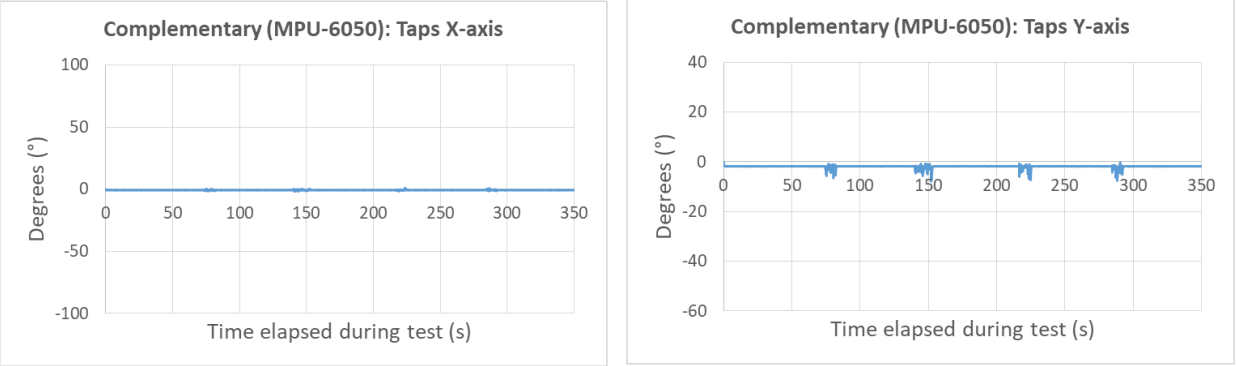



Figure 4.28: Complementary Taps X and Y axis

It can be seen that the complementary filter eliminated the gyroscope drift and also reduced the accelerometers sensitivity. Thus, the complementary combines the best of both worlds and improves them. Note that the same bounds of the raw data was used to show the effect.

Now, let's evaluate the roll and pitch:

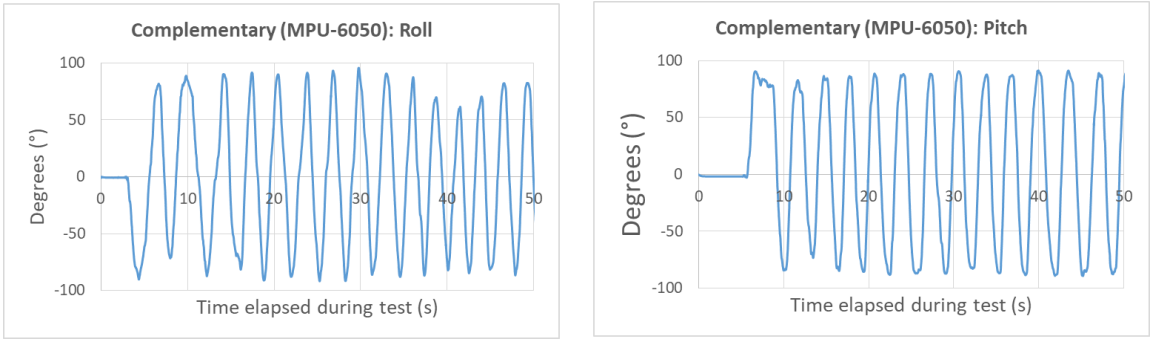



Figure 4.29: Complementary filter Roll and Pitch small sample size

Increasing the sample size:

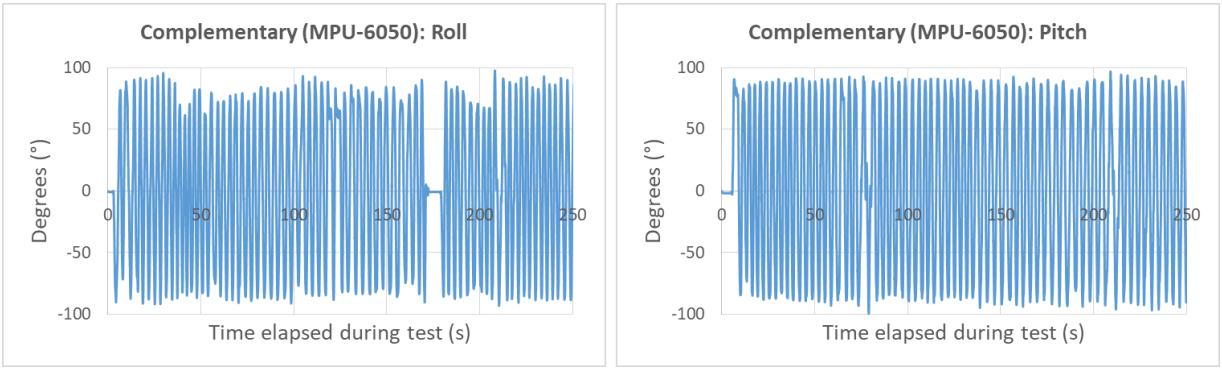



Figure 4.30: Complementary filter Roll and Pitch large sample size

As it can be seen, not only is the data smoother than ever, it is also not susceptible to drifting. This graphs show the effect of the complementary filter, a powerful and simple tool to implement to improve the shortcomings of the sensors. Note: On the left graph there is a point between 15000 and 20000 where the plot goes to 0, I did this to show that the data returns to the original position after being moved around.

### 4.2.3 Digital Motion Processing (DMP) data

As it was mentioned previously, the DMP data fusion is a powerful technique that utilizes data fusion to improve the quality of the output data. This will be shown in the following graphs, where *No movement* and *Tapping the sensor* tests were performed for all of the axis:

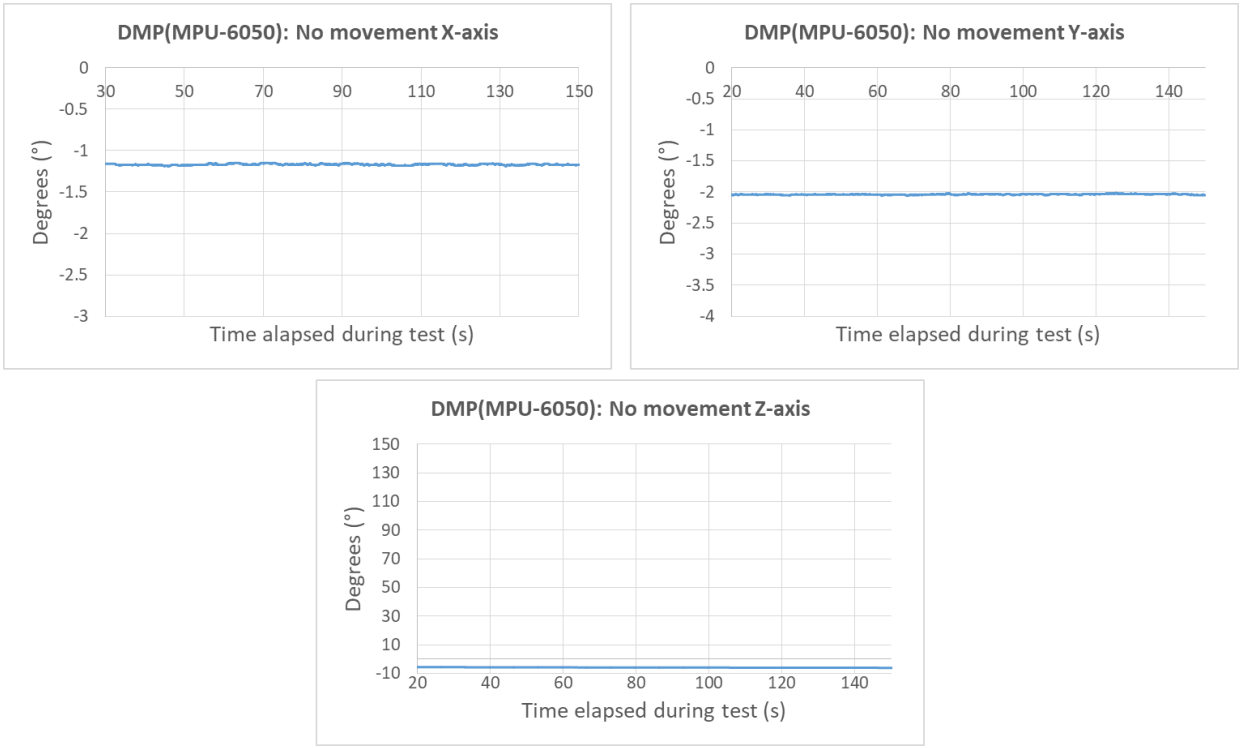



Figure 4.31: DMP No movement X, Y, and Z axes

It can be seen that the data is smoother than the complementary filter (less susceptible to noise) for the X and Y axes. The yaw exhibits a similar behavior as seen on the AltIMU-10 magnetometer, however, it is much smoother and less aggressive (moves only a few degrees).

This behavior can be seen on the X and Y axes as well. Overall it severely reduces drifting in all 3 axes. Now let us take a look at the next test results, *tapping on the sensor:*

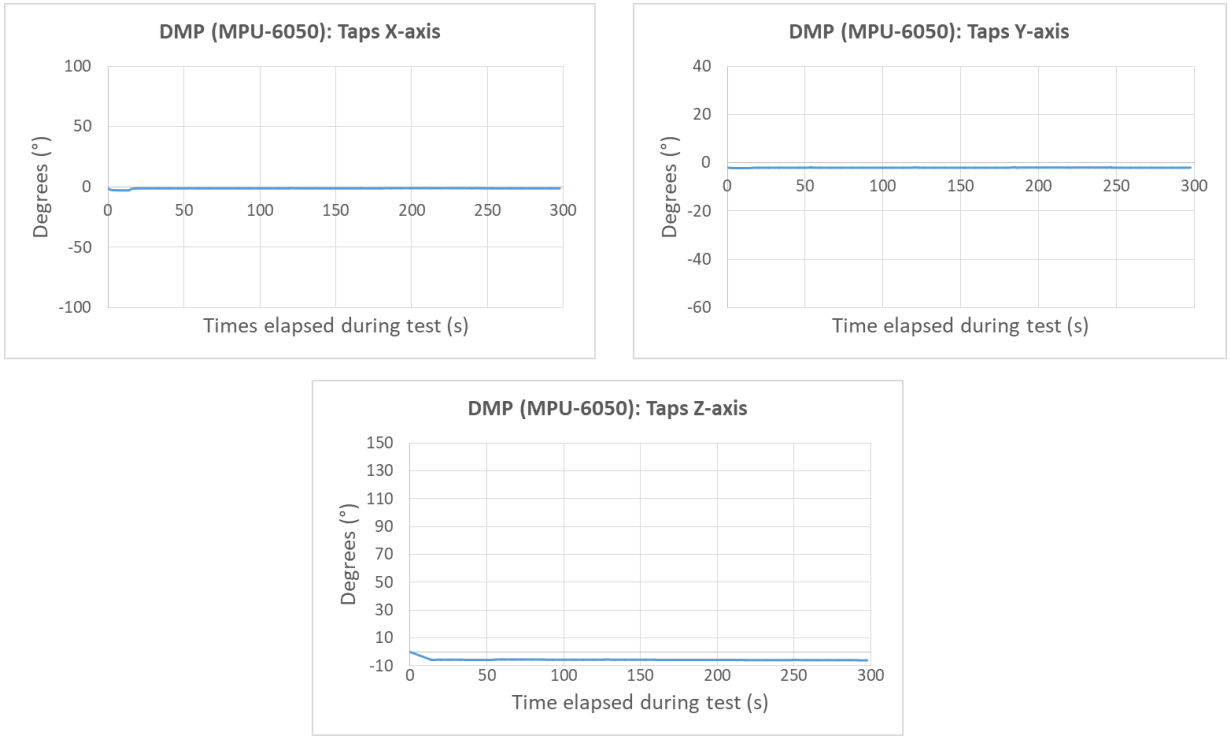



Figure 4.32: DMP Taps X, Y, and Z axes

Once again, it can be seen that the data is less susceptible to the noise factors (in this case the taps) than the complementary filter for all of the axes. It features a similar behavior as the magnetometer, in that it adjust to a certain position. The final test will be Roll, Pitch, and Yaw movement. Since the complementary filter cannot compute Yaw, we will use the best available data; the filtered gyroscope yaw. Of course the gyroscope data (even filtered) will continue to drift, however, it will serve as a basis for comparison to demonstrate the power of the DMP fusion sensor algorithm. After the results are shown the conclusions for the DMP will be explained and we will continue to the next test.

Figure 4.29 will show the results of the DMP roll and pitch, a smaller sample was taken since the previous test data proves that the output does not drift.

It can be seen that the roll and pitch output of the DMP is about as smooth as the complementary filter.

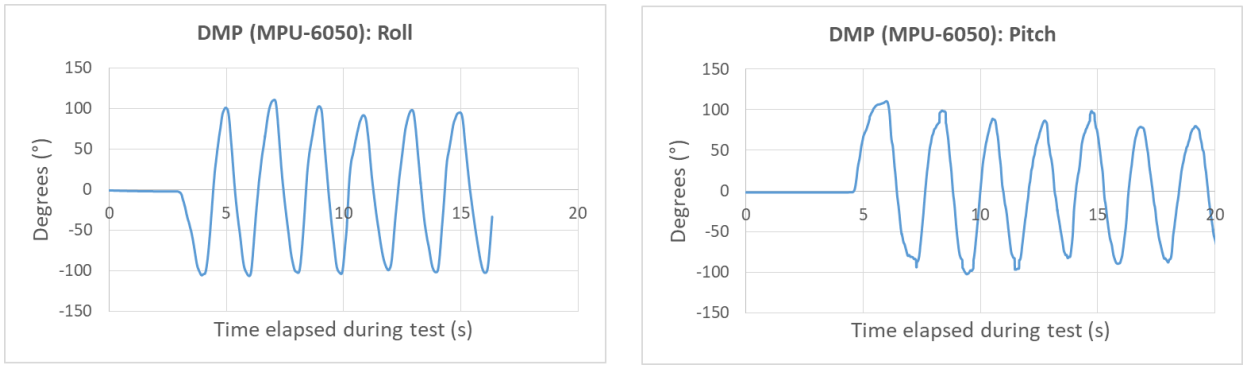



Figure 4.33: DMP Roll and Pitch

The advantage is that the DMP is also able to calculate the yaw, let us compare it to the best output we got from the complementary, the filtered gyroscope output:

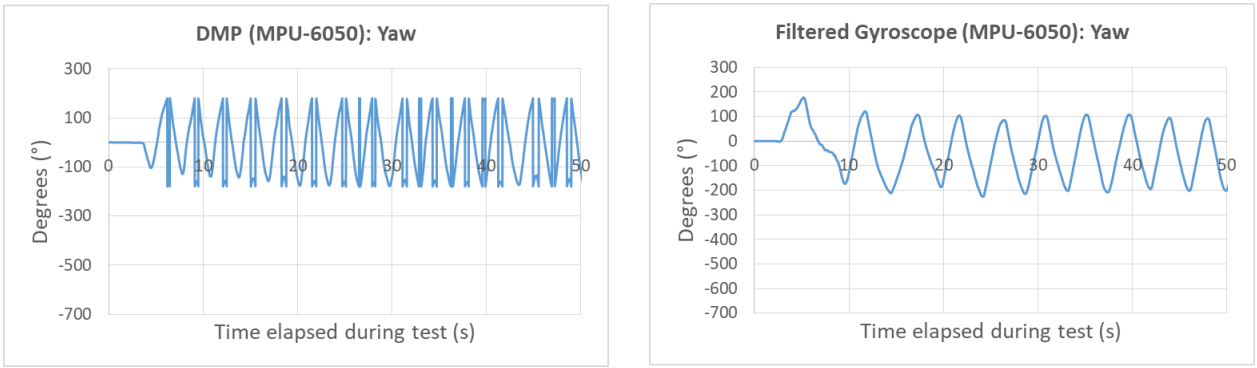



Figure 4.34: DMP Yaw vs. Filtered Gyroscope Yaw small sample

As it can be seen, the yaw from both outputs is very smooth (the DMP was taken at a faster rate), however, the problem with gyroscope data is that it will drift. Figure 4.31 will display the results in a larger sample size.

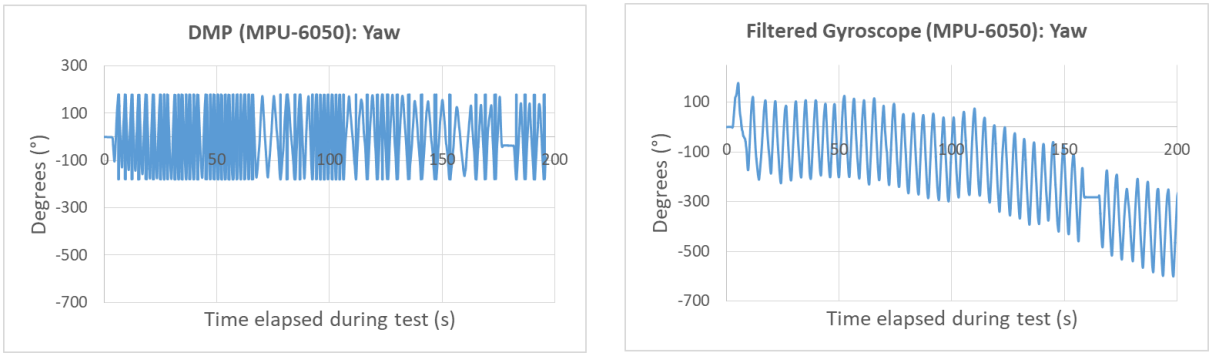



Figure 4.35: DMP Yaw vs. Filtered Gyroscope Yaw larger sample

In conclusion, the DMP is as good (if not better) as the complementary filter for computing roll and pitch. It improves on the noise from random forces from the complementary and it is able to compute the yaw angle (without severe drifting). Thus, for a real application it is the recommended configuration to use. Nevertheless, using and figuring out how the complementary filter works is useful for getting started on the world of the IMUs. On the next section the G-force output will be discussed as well as the Fourier transform plots.

### 4.2.3 Frequency of G-force outputs data

Depending on the application, the user might only care about the G-force. For this reason, a MATLAB code that reads the accelerometer data in G-force (x, y, and z) from the Arduino code outputs and transforms the data using the Fourier transform to frequencies was created. The (Forward) Fourier transform transforms a function of time to frequency by dividing it by sines and cosines, of course, this is a real simplified concept of the Fourier transform. The user should definitely study and understand the Fourier transform, but further explanation of it is beyond the scope of this thesis.

This is useful for several applications, but we saw potential in using the IMU as an accelerometer aid for a vibration shaker. In addition to this, both sensors were tested at a 47 cm drop to demonstrate their measurement capabilities. Results will be shown in the following figures:

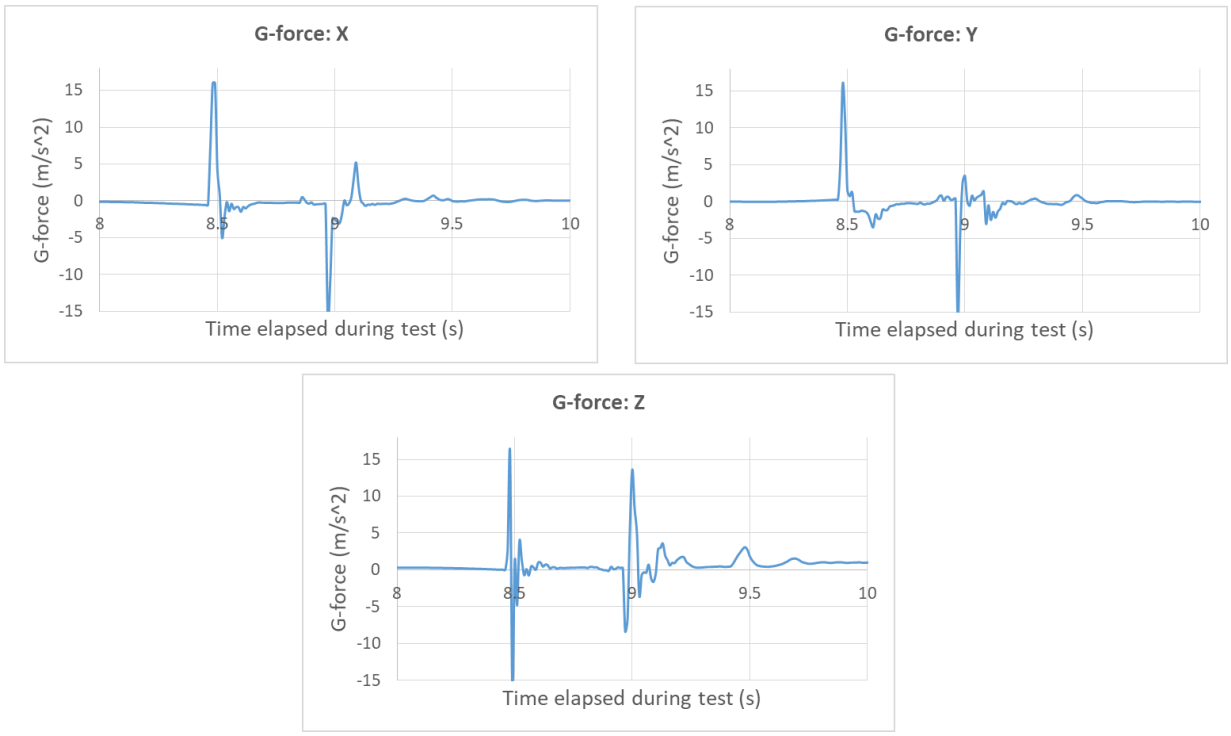



Figure 4.36: G-force output data from 47 cm fall

As it can be seen the moment the sensor drops it is able to read the 16 G-force value. Additionally, it's worth noting that the Z-axis it's always reading 1G, this is because it using gravity as a reference (this is why Yaw cannot be calculated from the accelerometer output). Next, we will see the Fourier transform of the data. From the X, Y, and Z axes. According to the MPU-6050 datasheet, the accelerometer has a sampling rate of 1 KHz, in other words, it is taking 1000 measurements per second. This will be used in the Fourier transform.

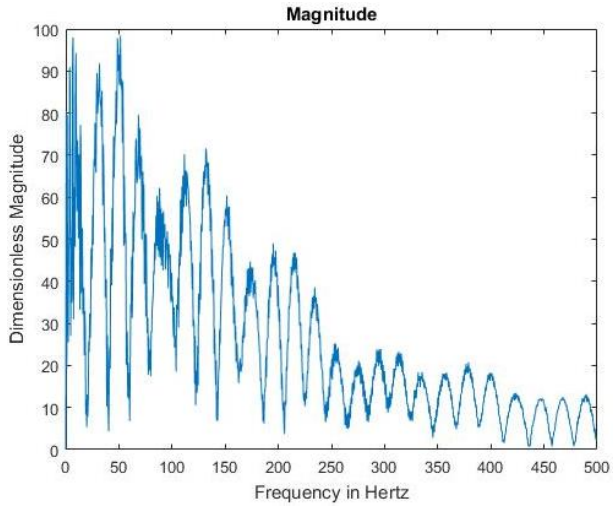



Figure 4.37: Fourier transform of G-force (X-axis)

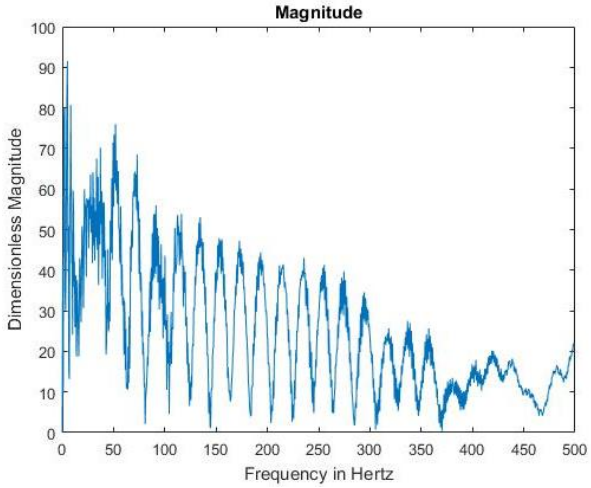



Figure 4.38: Fourier transform of G-force (Y-axis)

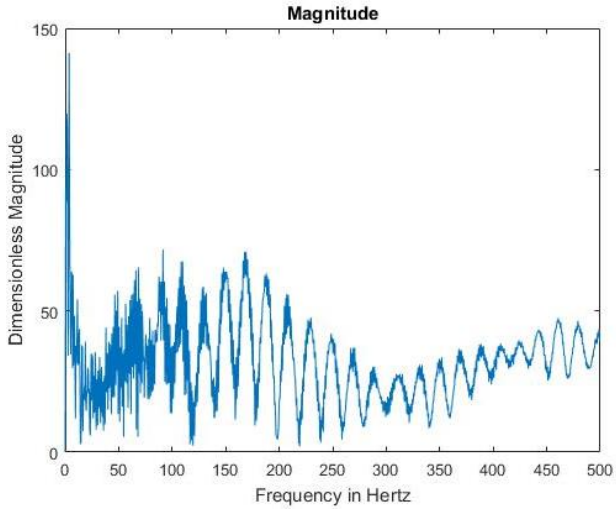



Figure 4.39: Fourier transform of G-force (Z-axis)

On the next section I will provide some information on the codes that I will present as part of the final package.

### 4.2.4 MATLAB, Arduino, and Processing codes

In addition to all the testing done I will also provide several codes that will allow users to ease themselves into the world of the IMUs. I will take this section to provide a brief explanation of the package that will be provided.

The package will include: PuTTytel, all of the necessary Arduino and Processing libraries, useful data (datasheets for the MPU-6050 and the accelerometer/gyroscope of the AltIMU-10), instructions for installing both libraries, and finally Arduino/MATLAB/Processing codes. These codes will allow the user to do all the testing that I have done, they include: AltIMU-10, Bluetooth (HC-06), complementary filters, Kalman filter (an example code), and the DMP data. Going over all of the folders really quick, the *AltIMU-10* contains the complementary filter and a Processing code to visualize, as well the Python code for visualization with the integration of the magnetometer. *Bluetooth* contains a code that allows the user to change the device name, baud rate, and password. *Complementary folder MPU-6050* contains the complementary filter and a Processing code to visualize, as well as a MATLAB code to compare data (from post-processing). *Kalman filter* contains a filter example. *MPU-6050 DMP* contains the airplane example to visualize the algorithm, as well as an alternative code. In addition in contains MATLAB codes to visually both in real time and post-processing the data from the output. It also contains calibration routine files, and finally the G-force/Fourier transform code.

All of these codes have additional user information to further facilitate its use. This will be provided in a separate folder as deliverables. The next chapter will focus on laboratory workshops that can be applied for educational purposes. These workshops will test the student understanding on the topic, and will help students and faculty create innovate uses for the IMUs.

# CHAPTER 5: IMU'S LABORATORY WORKSHOPS

This chapter will introduce some possible laboratory assignments to test the knowledge of students on the subject. In addition, I have thought of 2 projects that could be implement as a final/senior project, as they are more extensive. I will provide some instructions, goals, and suggestions to implement them in a classroom environment.

## 5.1 LAB 1: TAKING MEASUREMENTS FROM AN IMU

*Introduction:* Once students are more familiar with the IMU sensors they should be able to take data the raw data from any sensor and transform it to a desired output (given that they got the corresponding datasheet). It will be assumed that the students has been working with the MPU-6050 IMU sensor.

*Instructions:* Using an AltIMU-10 (or a different one) sensor, create an Arduino code that gives the following outputs:

- Accelerometers values in G-force
- Accelerometers angles in degrees (Roll and Pitch)
- Gyroscope angular velocities (Angular velocity of X,Y, and Z axes)

*Background theory:* In order to obtain values in the desired format, it is necessary to transform the data using the sensitivity factor of the corresponding sensor. For example, on the MPU-6050 the sensitivity scale factor of the gyroscope (when selecting a sensitivity value of ± 250) is 131 $\frac{LSB}{\frac{\circ}{s}}$, thus, in order to convert the output one would have to take the raw gyroscope output and divide it by 131. Similarly, with an accelerometer with a sensitivity factor of 2048 $\frac{LSB}{g}$ applying the following formula would give the results in G-force units. The formula is $\frac{(raw\ accelerometer\ output)}{2048}$. The sensitivity factor of the AltIMU-10 are 0.061 $\frac{mg}{LSB}$ (accelerometer) and 4.375 $\frac{mdps}{LSB}$ (gyroscope). Take a look at the units and think how you would approach this problem.

*Solution:* Since the sensitivity values are inverse of the sensitivity values found on the MPU-6050, in other words the LSB is at the bottom in the division, the formula to obtain the G-force values of the accelerometer would be as follows:

$$\frac{(Raw\ acceleration\ output)(0.061)}{1000}$$

This would be repeated for the X, Y and Z axes. The reason we are dividing by a thousand is because we are given values in mg, and we would want to use g values. Similarly, the equation for the gyroscope would be as follows, which would give us the output in angular velocity:

$$\frac{(Raw\ gyroscope\ output)(4.375)}{1000}$$

Finally, in order to calculate the Roll and Pitch using the raw accelerometer angles we would use the following formulas:

$$Roll = \tan^{-1}\left(\frac{Ay}{\sqrt{Ax^2+Az^2}}\right)$$

$$Pitch = \tan^{-1}\left(\frac{-Ax}{\sqrt{Ay^2+Az^2}}\right)$$

Where Ax, Ay, and Az correspond to the raw accelerometer output of each axes.

## 5.2 LAB 2: COMPLEMENTARY FILTER IMPLEMENTATION

*Introduction:* Once the students are familiar with the IMU sensors and how to obtain their necessary output values, they should be able to create a complementary filter. The complementary filter combines the outputs of the accelerometer and gyroscope to improve their shortcomings. It will be assumed that the students have been using the MPU-6050 IMU sensor.

*Instructions:* Using an AltIMU-10 (could be a different sensor) sensor create an Arduino code that gives the following outputs:

- Complementary filtered Roll angle (X-axis)
- Complementary filtered Pitch angle (Y-axis)
- Graph that compares filtered and unfiltered data (some sort of post-processing)

*Background theory:* Using the following formula on the gyroscope will transform its output to degrees per second: $\frac{(Raw\ gyroscope\ output)(4.375)}{1000}$, use this formula on all of the outputs to obtain the correct format. Additionally, the following formula gives the complementary filter:

Filtered Angle = $\alpha\ (Gyroscope\ angle) + (1 - \alpha)\ (Accelerometer\ angle)$, where $= \frac{\tau}{\tau + \Delta_t}$ . The problem lies in that the gyroscope measures angular velocity, but we want position (angle). Thus it is necessary to integrate the velocity, to do this DT will need to be found. DT is the change in time from measurement "A" to measurement "B". As a hint we will assume the velocity remains constant between point A and B (current measurement and previous measurement respectably).

*Solution:* The gyroscope angles (Roll, Pitch, and Yaw) can be obtained with the following formula:

$$Gyroscope\ angle\ (position\ angle) = Gyro_{current_{angle}} * dt + Gyro_{previous_{angle}}$$

In addition, the user will also have to implement a way to keep track of time, I used the function millis () in Arduino, refer to my thesis for more information.

### 5.3 LAB 3: CONTROL A SERVOMOTOR USING AN IMU

*Introduction:* The IMU can be used for a lot of applications, one of them is controlling servomotors. This workshop will serve as the base for future projects involving the IMU. It will be assumed that the students have been using the MPU-6050 IMU sensor, are familiar with their functionality, and using Arduino. For this project it is recommended to use the DMP algorithm as it provides the most accuracy possible.

*Instructions:* Using the DMP-6050 sensor, create an Arduino code that controls a servomotor by change in degrees, in other words the servos will move according to the angles provided by the IMU. The code should be able to provide the following outputs:

- The servo is controlled by the roll angles
- The servo is controlled by the pitch angles
- The servo is controlled by the yaw angles

*Background theory:* The students should be familiar with the Arduino library, as it provides the necessary tools for this workshop. Students should look at the servo library and the examples sweep and knob. The following line is essential, as it attaches the servo to one of the pins. For example, myservo.attach(9) would attach servo on pin number 9 of the board.

*Solution:* There are multiple solutions to this workshop, the idea here is to get students to become aware of different forms to include the IMU into their projects. The following lines of code should provide good results. Note: This is a reduced code found on github.

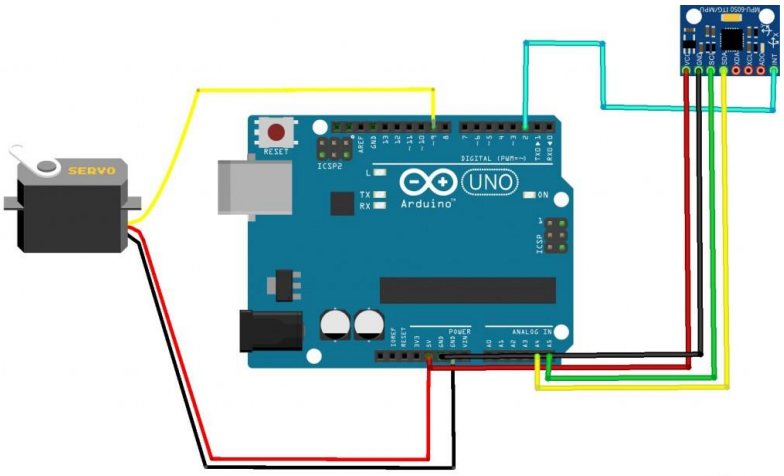



Figure 5.0: Servo connection example [50]

```
int servoPin1 = 3, servoPin2 = 5, servoPin3 = 11;
        Servo servo1, servo2, servo3;
        void setup() {
            servo1.attach(servoPin1);  // attaches servo to pin3
            servo2.attach(servoPin2); // attaches servo to pin5
            servo3.attach(servoPin3); // attaches servo to pin11
            servo1.write(0);
            servo2.write(0);
            servo3.write(0);
        #ifdef OUTPUT_READABLE_YAWPITCHROLL
              mpu.dmpGetQuaternion(&q, fifoBuffer);
              mpu.dmpGetGravity(&gravity, &q);
              mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
```

```
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            servo1.write(map(ypr[0] * 180/M_PI, -180, 180, 0, 180)); //Controls servo1
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            servo2.write(map(ypr[1] * 180/M_PI, -90, 90, 0, 180)); //Controls servo2
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
            servo3.write(map(ypr[2] * 180/M_PI, -90, 90, 0, 180)); //Controls servo3
        #endif [50]
```

The students will hopefully come out with creative solutions, and finishing this laboratory workshop will give leeway to the next 2 projects.

### 5.4 LAB 4: CONTROL THE MOVEMENT OF A ROBOT CAR USING AN IMU

*Introduction:* This project will work as a final/senior project for students, it will combine mechanical knowledge (building the car), electrical knowledge (creating the circuits), and programming knowledge (programming the Arduino) into one package. It will work as an improvement to the current senior project as it provides a greater challenge and depth. The students will need to provide the following deliverables:

- Build a car (no restrictions here) with the materials provided, to be as robust as possible. This is already part of the current senior project.
- In addition to the physical tests from previous years, it will also include an obstacle course that should be driven using MPU implementation.
- The car should be able to communicate and respond to external commands from the sensor in order to avoid the obstacles.

*Background theory:* The students should be familiar with the Arduino library, as it provides very useful tools. In addition, they must use shields in order to communicate with the MPU,

motors, etc. It will require a lot of research in this area to create this car, thus, it should provide an excellent challenge and teach the students more about the mechatronics area.

*Solution:* There are multiple solutions to this final project/senior project, students can be as creative as possible. One possible solution is to implement several commands from different output of the sensors. For example, the yaw will control servos that are able to change direction for lateral movements. On the other hand, the pitch will control a motor that will increase, reduce speed, or stop depending on the position. The roll could provide a different function. The purpose of this project is not to build the best car, but to be as creative as possible, which I believe will enrich and better prepare students for industry.

## 5.5 LAB 5: CREATE A SELF BALANCING ROBOT USING THE IMU

*Introduction:* An alternative project for a final/senior project. Similarly, this project will combine mechanical knowledge (building the robot), electrical knowledge (creating the circuits), and programming knowledge (programming the Arduino) into one package. It will work as an alternative to the current senior project and provide a greater challenge and depth. The students will need to provide the following deliverables:

- Build a self-balancing robot (no restrictions here) with the materials provided, to be as robust as possible. There are plenty of examples online, the following citation provides an excellent example [51].
- Minimum components required are: 2 motors, a motor controller, an IMU, a microcontroller (Arduino).
- The robot should be able to self-balance for a period of time (this will be defined by the professor).

*Background theory:* The students should be familiar with the Arduino library, as it provides very useful tools for the project. In addition, they must use shields in order to communicate with the MPU, motors, etc. Self-balancing systems are a very important tool in many applications, thus, it will provide a base for many applications.

*Solution:* There are multiple solutions to this final project/senior project, students can be as creative as possible. The purpose of this project is to introduce students to the world of self-balancing systems. Even if the robot does not properly self-balance, students will have the experience of building the robot, coding the sensors, and use creative thinking to come up with a solution. They will have to experience researching, and many other tools that will surely help them in industry or a master's program.

In conclusion, this chapter brought up a few practical implementations of the IMU. There are an infinite number of applications for it, but these should provide a good baseline. The next and final chapter will focus on conclusions, recommendations, and future work for the IMU sensors.

# CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS

## 6.1 CONCLUSIONS

The following conclusions can be drawn from the work presented in this thesis:

1. This thesis provided a descriptive guide into the world of the IMU sensors, its components, noise factors, recommendations for selection, and filtering techniques.
2. The individual components of the IMU are extremely susceptible to noise. The gyroscope component tends to drift after time (because it is integrating), while the accelerometer is sensitive to forces. The magnetometer is susceptible to electrical equipment.
3. Each individual component can be used to give information for attitude control. The gyroscope gives a high quality signal in a short term. The magnetometer gives precise angles, but it takes a longer time to get an accurate result. And the accelerometer is good for measuring changes in speed.
4. With the test data presented, it was clearly shown that when applying filters and algorithms to the outputs of the IMU, the sensor not only reduces the individual noise factors that affect each sensor, but enhances their ability. Combining the accelerometer and gyroscope together gives the basic complementary filter. The complementary filter allows for the precision quality signal output of the gyroscope, while the accelerometer corrects the drifting. The output also becomes more robust against noise factors. A magnetometer can be further added to correct the gyroscope.
5. More advance options exist, such as the DMP, Kalman filter, Mahony's filter, and Madwick's filter. The results from the DMP were showed, and it provide clear results of superior quality compared to the complementary filter.
6. Arduino provides a platform for students and newcomers to work with these sensors. For this reason, an introductory guide was also provided to provide the necessary tools to get started with Arduino.

7. In addition, a package of readily available codes was provided. These codes allow anyone to replicate the tests and results found.
8. MATLAB and Processing were used to provide visual representation of the codes, as well as post-processing, these were added as part of the package as well.

## 6.2 RECOMMENDATIONS AND FUTURE WORK

The more advance filtering techniques are recommended lecture material, as they allow a higher level of precision that might be necessary in a more complex application. In addition, more powerful microcontrollers are recommended (once the students have a clear grasp on the technology). Implementing a PID controller will also be useful for future projects.

Possible applications of the IMU include: autonomous vehicle technology, self-balancing robots, implementations in a quadcopters, controllers, camera stabilization, and hand gesture applications.

# LIST OF REFERENCES


[1] Barbour, N., and G. Schmidt. "Inertial sensor technology trends." *Proceedings of the 1998 Workshop on Autonomous Underwater Vehicles (Cat. No.98CH36290)*, 2001.

[2] Wrigley, W. "History of Inertial Navigation." Navigation 24, no. 1 (1977): 1-6. doi:10.1002/j.2161-4296.1977.tb01262.x.

[3] Dunn, William C. "Accelerometer Design Considerations." *Micro System Technologies 90*, 1990, 131-36. doi: 10.1007/978-3-642-45678-7_19.

[4] Mcclary, C. "The technological evolution of inertial reference systems." *Proceedings of Position, Location and Navigation Symposium - PLANS 96*, 2002. doi:10.1109/plans.1996.509097.

[5] MarketsandMarkets, H. (n.d.). Market Research Reports. Retrieved July 01, 2017, from http://www.marketsandmarkets.com.

[6] Whishaw, I. Q., Hines, D. J., & Wallace, D. G. (2001). Dead reckoning (path integration) requires the hippocampal formation: evidence from spontaneous exploration and spatial learning tasks in light (allothetic) and dark (idiothetic) tests. Behavioral Brain Research, 127(1-2), 49-69. doi: 10.1016/s0166-4328(01)00359-x.

[7] Chiang, K. W., H. Hou, X. Niu and N. El-Sheimy (2004) "Improving the Positioning Accuracy of DGPS/MEMS IMU Integrated Systems Utilizing Cascade De-noising Algorithm" in Proceedings of ION GPS, 21-24 September, Long Beach CA, pp. 809-818, U. S. Institute of Navigation, Fairfax VA.

[8] Armenise, M. N. (2010). Emerging Gyroscope Technologies. Advances in Gyroscope Technologies, 103-108. doi:10.1007/978-3-642-15494-2_7

[9] Shieh, J., Huber, J., Fleck, N., & Ashby, M. (2001). The selection of sensors. Progress in Materials Science, 46(3-4), 461-504. doi:10.1016/s0079-6425(00)00011-6

[10] Mobile Satellite, Guidance, and Stabilization Systems from KVH Industries. (n.d.). Retrieved July 01, 2017, from http://www.kvh.com

[11] Taking the Mystery out of RF Design. (2009). Circuits and Systems Tutorials. doi:10.1109/9780470544235.ch29

[12] Billah, K. Y., & Scanlan, R. H. (1991). Resonance, Tacoma Narrows bridge failure, and undergraduate physics textbooks. American Journal of Physics, 59(2), 118-124. doi:10.1119/1.16590

[13] OEM-HG1900 MEMS Inertial Measurement Unit (IMU). (n.d.). Retrieved July 01, 2017, from http://www.novatel.com/products/span-gnss-inertial-systems/span-imus/span-mems-imus/OEM-HG1900

[14] TDK | Attracting Tomorrow. (n.d.). Retrieved July 01, 2017, from https://www.invensense.com

[15] Home - STMicroelectronics. (n.d.). Retrieved July 01, 2017, from http://www.st.com

[16] Heideman, M., Johnson, D., & Burrus, C. (1984). Gauss and the history of the fast Fourier transform. IEEE ASSP Magazine, 1(4), 14-21. doi:10.1109/massp.1984.1162257

[17] (n.d.). Retrieved July 02, 2017, from https://lambda.gsfc.nasa.gov/

[18] R.F. Tinder, Synthesis Lectures on Engineering 1, 1 (2006)

[19] Taylor, E. F., Wheeler, J. A., Wheeler, J. A., & Wheeler, J. A. (2007). Spacetime physics: introduction to special relativity. New York: Freeman.

[20] Feature Articles. (n.d.). Retrieved July 16, 2017, from http://www.electronicproducts.com.

[21] Hampshire, A. (2009). G force. Markham, Ont.: Fitzhenry & Whiteside.

[22] A Guide to using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. (n.d.). Retrieved August 24, 2017, from http://www.starlino.com/imu_guide.html

[23] Share and discover research. (n.d.). Retrieved August 08, 2017, from https://www.researchgate.net

[24] MEMS Accelerometer Gyroscope Magnetometer & Arduino. (2016, May 16). Retrieved August 09, 2017, from http://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino

[25] Vodenikov (1996). A portable magnetometer. Biomedical Engineering.

[26] Mao, X., Wen, X., Song, Y., Li, W., & Chen, G. (2017). Eliminating drift of the head gesture reference to enhance Google Glass-based control of an NAO humanoid robot. International Journal of Advanced Robotic Systems, 14(2), 172988141769258. Doi: 10.1177/172988141769258

[27] Availability Bias, Source Bias, and Publication Bias in Meta-Analysis. (n.d.). Methods of Meta-Analysis: Correcting Error and Bias in Research Findings, 513-551. doi:10.4135/9781483398105.n13

[28] [TUT] [HARD] Gyros and Accelerometers: The Basics. (n.d.). Retrieved September 12, 2017, from http://www.avrfreaks.net/forum/tut-hard-gyros-and-accelerometers-basics?name=PNphpBB2&file=viewtopic&t=89294

[29] Freescale Semiconductor acquired by consortium. (2006). III-Vs Review, 19(8), 6. doi: 10.1016/s0961-1290(06)71837-3

[30] Shen, X., & Meng, S. (2014). Allan variance segmented circular fitting method for laser gyroscopes random error analysis. Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference. doi:10.1109/cgncc.2014.7007493

[31] Accelerometers. (n.d.). Retrieved September 17, 2017, from https://www.xsens.com/tags/accelerometers/

[32] Prommee, P., Thongdit, P., & Angkeaw, K. (2017). Log-domain high-order low-pass and band-pass filters. AEU - International Journal of Electronics and Communications, 79, 234-242. doi:10.1016/j.aeue.2017.06.014

[33] Mixed-signal and digital signal processing ICs | Analog Devices. (n.d.). Retrieved October 16, 2017, from http://www.analog.com/

[34] (n.d.). Retrieved October 16, 2017, from http://www.gcdataconcepts.com/calibration.html

[35] (n.d.). Retrieved October 26, 2017, from https://learn.sparkfun.com/tutorials/what-is-an-arduino

[36] My Cable Mart. (n.d.). Retrieved October 26, 2017, from https://www.mycablemart.com/

[37] The Arduino Playground. (n.d.). Retrieved October 28, 2017, from https://playground.arduino.cc/

[38] (n.d.). Retrieved October 29, 2017, from https://learn.sparkfun.com/tutorials/pulse-width-modulation

[39] MPU-6050 Accelerometer Gyro. (n.d.). Retrieved October 29, 2017, from https://playground.arduino.cc/Main/MPU-6050

[40] P. (2017, March 08). Pololu/lsm6-arduino. Retrieved October 29, 2017, from https://github.com/pololu/lsm6-arduino

[41] Guangzhou HC Information Technology Co., Ltd. [Company]. (2017, October 29

[42] Security Laboratory. (n.d.). Retrieved October 29, 2017, from https://www.sans.edu/cyber-research/security-laboratory/article/bluetooth

[43] #670407, M., H, M., P., #519940, M., #146957, M., K., #271784, M. (n.d.). SparkFun OpenLog. Retrieved October 29, 2017, from https://www.sparkfun.com/products/13712

[44] Higgins, W. (1975). A Comparison of Complementary and Kalman Filtering. IEEE Transactions on Aerospace and Electronic Systems, AES-11(3), 321-325. doi:10.1109/taes.1975.308081

[45] IMU Data Fusing: Complementary, Kalman, and Mahony Filter. (n.d.). Retrieved November 05, 2017, from http://www.olliw.eu/2013/imu-data-fusing

[46] Kreinovich, Vladik, "Interval Computations as an Important Part of Granular Computing: An Introduction" (2007). *Departmental Technical Reports (CS).* Paper 155

[47] [TUT] [THEORY] what is a Kalman Filter? (n.d.). Retrieved November 13, 2017, from http://www.avrfreaks.net/forum/tut-theory-what-kalman-filter

[48] (n.d.). Retrieved November 13, 2017, from http://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures

[49] Faragher, R. (2012). Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]. IEEE Signal Processing Magazine, 29(5), 128-132. doi:10.1109/msp.2012.2203621

[50] Build software better, together. (n.d.). Retrieved November 26, 2017, from https://github.com/

[51] Introduction to Self-Balancing Robots. (2016, May 18). Retrieved November 26, 2017, from http://wired.chillibasket.com/2014/09/balancing-robot-intro/

# VITA

Master of Science student in Mechanical Engineering at University of Texas at El Paso, August 2015- present. Thesis tittle: “Microelectromechanical Systems Inertial Measurement Unit as an attitude and heading reference system”.

Bachelor of Science (May 2015) in Mechanical Engineering, University of Texas at El Paso.

Test Engineer at Cts Corporation October 2017- Present. Responsibilities include: Providing technical assistance, testing actuators performance under different temperatures, loads, and voltages, configure calibrations on the actuator to improve performance.

Product Engineer at Delphi Automotive May 2016-October 2017. Responsibilities include: Providing technical assistance, supervising production facilities, and evaluating process for Remote Control Actuators, managing BOM, drawing revisions and testing products.

Graduate Teaching Assistant, Department of Mechanical Engineering, University of Texas at El Paso, August 2015-May 2017. Responsibilities include: assisting professors with the preparation of undergraduate courses, grading and tutoring.

Undergraduate Research Assistant to Noe Vargas, Department of Mechanical Engineering, University of Texas at El Paso, Spring 2014. Research activities include: 3D Cad Design, cost analysis and part selection.

Top senior Undergraduate Student, Department of Mechanical Engineering, University of Texas at El Paso 2015. Dean’s list, Department of Mechanical Engineering, University of Texas at El Paso. Mascareñas foundation scholarship, IME Scholarship, Benito Juarez scholarship, the Jimmy and Yolanda Janacek scholarship recipient.

Contact Information: fjsalcedoortega@miners.utep.edu/fjsalcedoortega@gmail.com

This thesis was typed by Francisco Javier Salcedo Ortega.