Master's Thesis in Electrical Engineering with emphasis in Signal Processing
June 2017

Orientation estimation and movement recognition using low cost sensors

Álvaro López Revuelta

Department of Applied Signal Processing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology and is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Álvaro López Revuelta
E-mail: alli16@student.bth.se

University advisors:
Benny Lövström
Department of Applied Signal Processing
E-mail: benny.lovstrom@bth.se

Ronnie Lövström
Department of Applied Signal Processing
E-mail: ronnie.lovstrom@bth.se

Department of Applied Signal Processing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Orientation estimation is a very well known topic in many fields such as in aerospace or robotics. However, the sensors used are usually very expensive, heavy and big, which make them not suitable for IoT (Internet of Things) based applications. This thesis presents a study of how different sensor fusion algorithms perform in low cost hardware and in high acceleration scenarios. For this purpose, an Arduino MKR1000 is used together with an accelerometer, gyroscope and magnetometer. The objective of the thesis is to choose the most suitable algorithm for the purposed practical application, which consists on attaching the device to a moving object, such as a skate board or a bike. Once the orientation is estimated, a movement recognition algorithm that was developed is able to match what trick or movement was performed. The algorithm chosen was the Madgwick one with some minor adjustments, which uses quaternions for the estimation and is very resilient when the device is under strong external accelerations.

Keywords: sensor fusion, orientation, quaternion, euler angles, rotation matrices.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Outline of the Thesis

The estimation of the orientation of a body consists on determining its relative position to a local three dimensional axis, often given by the Euler angles Roll ϕ, Pitch θ and Yaw ψ. It is a very well known topic in many fields like in aerospace, robotics, human motion analysis or navigation. However in all these scenarios very expensive sensors of several hundred or thousands euros and large size and weight are used. This thesis focuses on orientation estimation using small and low cost sensors, in the order of milliliters and few euros.

In Ch. 2 the most important techniques for describing the orientation of a body are explained, which are the quaternion form, Euler angles and rotation matrices. Then, in Ch. 3 different sensors used to estimate the orientation are described, which are accelerometers, gyroscopes and magnetometers. Some estimation techniques using single sensors are explained, and also calibration for each sensor is described. Later on in Ch. 4 different sensor fusion techniques are presented, from the simplest complimentary filter to the DCM and the Madgwick algorithms. Then in Ch. 5 the experimental setup of the thesis is explained, which is based on an Arduino and a PCB with sensors. This setup will be used to test the performance of different algorithms, which results are described in Ch. 6. Using the device described in the experimental setup, in Ch. 7 a study case is presented, where the sensor is attached to a movement object and some movement recognition is done using data processing in Matlab. This study case can be for example a bike or skate board moving and doing different movements or tricks. To conclude, in Ch. 8 the summary and conclusions of the thesis are provided and in Ch. 9 some ideas for future related work are given.

1.2 Research Questions

In the following section different research questions are presented:

- Can the orientation be estimated using low cost sensors such as accelerometers, gyroscopes and magnetometers in high acceleration scenarios, using them separately?
- Can the orientation be estimated using low cost sensors such as accelerometers, gyroscopes and magnetometers in high acceleration scenarios, using sensor fusion techniques?
- Can these algorithms run in low cost hardware such as Arduino, at frequencies in the order of 50-100 Hz?

1.3 Related works

This section presents an overview of previous works done in orientation estimation. Two different approaches are considered: using each sensor separately and using sensor fusion algorithms.

Regarding to single sensor orientation estimation, it is important to highlight that the accelerometer by itself can't provide a full estimation of the orientation, due to the nature of the gravity. Also, magnetometers can't estimate the full orientation because of the same reason. However, gyroscopes can estimate all Euler angles by integration, but when it comes to cheap sensors, the result drifts with time. There are different approaches that address gyroscope drift cancellation that typically uses Kalman filter based algorithms [40], [39]. There are also other approaches like [39] that uses neural networks. The results are promising, but they tend to be very computationally demanding, which makes them not implementable in low cost devices such as Arduino.

Other approach to estimate the orientation, and solve the drift, is to use sensor fusion algorithms. This method is the one addressed by this thesis. In [24], [23] an algorithm called Madgwick is proposed, which uses the quaternion form for the estimation, where accelerometer and magnetometer is combined formulating an optimization problem by gradient descent. Other approach that also uses sensor fusion is presented in [5], [14], where a DCM (Direction Cosine Matrix) is used as the core of the algorithm.

Chapter 2

Orientation

Representing the attitude orientation is a well known topic in the aerospace field, where it has an important role in aircrafts or UAVs but it also has taken relevance in other fields. In the following chapter the three main mathematical ways of representing the attitude of a rigid body in a three dimensional space are presented. These ways are: Rotation Matrix [15][16] found in Sec. 2.1, Euler Angles [7] in Sec. 2.2 and Quaternions [22] in Sec. 2.3.

It is also really important to know the relation between these three ways, which can be found in [8], [18], [12]. Note also that the ZYX convention will be used for the thesis, and in [2] there is a really good compilation of all possible representations. The most important relations, always assuming ZYX can be found in Eq. 2.12, Eq. 2.26, Eq. 2.27 and Eq. 2.28.

2.1 Rotation Matrix

A rotation matrix represents a rotation in the Euclidean Space. It can be defined in spaces of n dimensions, but are typically used in two and three dimensional spaces. For example the rotation matrix in Eq. 2.1 represents a rotation of θ degrees in counter clock wise.

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2.1}$$

A rotation matrix is orthogonal, which means that $R^{-1} = R^t$ and its determinant $\det(R) = \pm 1$. When $\det(R) = 1$ the matrix is called *proper* and when $\det(R) = -1$ is called *improper*. This means that the rotation is made counter clock wise and clock wise, respectively.

So a vector with components (x, y) can be rotated θ using Eq. 2.2, where

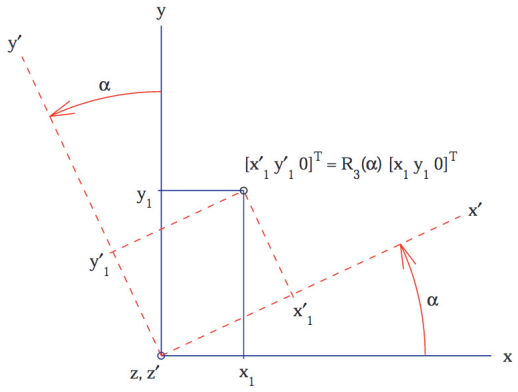

Figure 2.1: Rotation along the z axis using a rotation matrix R.

(x', y') are the components after performing the rotation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.2}$$

The opposite sense of rotation, clock wise, is presented in Eq. 2.3.

$$R(-\theta) = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2.3}$$

However, for this thesis it is more interesting the three dimensional rotations, so in $\mathbb{R}^3$ the coordinate system rotations of the x,y and z axis in counter clock wise direction is given by the matrices in Eq. 2.4, Eq. 2.5 and Eq. 2.6.

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2.4}$$

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{2.5}$$

$$R_z(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

An example of the rotation described above is presented in Fig. 2.1, where a rotation is produced along the z axis by an angle α.

An interesting property of rotation matrices is that two different rotations using two matrices R_1 and R_2 described in Eq. 2.7 and Eq. 2.8, is equivalent to

a single rotation using a single matrix R_3 determined in Eq. 2.9.

$$\mathbf{v}' = R_1\mathbf{v} \tag{2.7}$$

$$\mathbf{v}'' = R_2\mathbf{v}' \tag{2.8}$$

$$\mathbf{v}'' = R_2R_1\mathbf{v} = R_3\mathbf{v} \tag{2.9}$$

Where $R_3 = R_2R_1$ is the new equivalent rotation matrix and note that rotations are applied in reverse order.

So any rotation can be described as one rotation in each axis x, y, z and that can be represented by a $3x3$ rotation matrix like Eq. 2.10. Coefficients a_{ij} can be obtained by multiplying the different rotation matrices. Note that the results will vary depending on the order of the rotations. It is not the same rotation x, y, z than rotating z, y, x.

$$\begin{pmatrix} {x_1}' \\ {x_2}' \\ {x_3}' \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.10}$$

There is also one particular case of rotation matrix, called DCM or *Direction Cosine Matrix* which is often used in the literature of orientation. As the name indicates, it is a matrix composed by the cosines of the unsigned angles that the body forms with the reference axis. Reference axis are denoted by (x, y, z) and body axes by (x', y', z'). The angles are denoted by $\theta_{x'x}$. So the general equation for a DCM matrix is Eq. 2.11.

$$R_{DCM} = \begin{pmatrix} \cos(\theta_{x'x}) & \cos(\theta_{x'y}) & \cos(\theta_{x'z}) \\ \cos(\theta_{y'x}) & \cos(\theta_{y'y}) & \cos(\theta_{y'z}) \\ \cos(\theta_{z'x}) & \cos(\theta_{z'y}) & \cos(\theta_{z'z}) \end{pmatrix} \tag{2.11}$$

Later on in the thesis, the DCM algorithm for calculating the Euler angles will be explained. That algorithm uses a DCM matrix to perform sensor fusion, and in the last step Euler angles are calculated. That conversion from DCM matrix to Euler angles can be performed using Eq. 2.12. Note that *pitch*, *roll* and *yaw* are the Euler Angles explained in Sec. 2.2. Note also that $R_{x,y}$ stands for the element x, y of the matrix.

$$\begin{pmatrix} pitch \\ roll \\ yaw \end{pmatrix} = \begin{pmatrix} -\arcsin\left(R_{31}\right) \\ \arctan\left(\frac{R_{31}}{R_{33}}\right) \\ \arctan\left(\frac{R_{21}}{R_{11}}\right) \end{pmatrix} \tag{2.12}$$

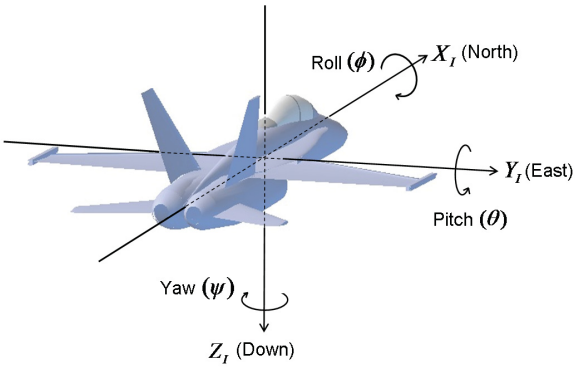

Figure 2.2: Example of Euler Angles in an aircraft.

2.2 Euler Angles

Euler Angles were first introduced by Leonhard Euler. According to his theorem, any rotation can be described using three angles (ϕ, θ, ψ) also referred as roll, pitch and yaw. In Fig. 2.2 an example is provided. It can also be viewed using head movements: saying no will be the yaw, saying yes will be the pitch and leaning the head to left or right will be the roll.

Note that the order that the angles are represented is not important at all, but the order of rotation is. For this thesis the rotation order ZYX (see DCM section) will be used. The following equation shown in Eq. 2.13 is a very important expression, which represent the rotation along the three axes in the ZYX order. Note that for shortening purposes, $\cos(x)$ is $c(x)$ and $\sin(x)$ is $s(x)$.

$$R(\phi,\theta,\psi) = \begin{pmatrix} c(\phi) & -s(\phi) & 0 \\ s(\phi) & c(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c(\psi) & -s(\psi) \\ 0 & s(\psi) & c(\psi) \end{pmatrix} \quad (2.13)$$

After doing the maths, in Eq. 2.14 the rotation matrix for angles ϕ, θ, ψ is represented using the order Z, Y, Z, which can also be denoted as $R_{ZYX}(\phi, \theta, \psi)$.

$$R = \begin{pmatrix} c(\phi)c(\theta) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ s(\phi)c(\theta) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) \\ -s(\theta) & c(\theta)s(\psi) & c(\theta)c(\psi) \end{pmatrix} \quad (2.14)$$

Note that in some literature, these angles can also be referred as α, β, γ, equivalent to ϕ, θ, ψ. An example of the rotation order is represented in Fig. 2.3.

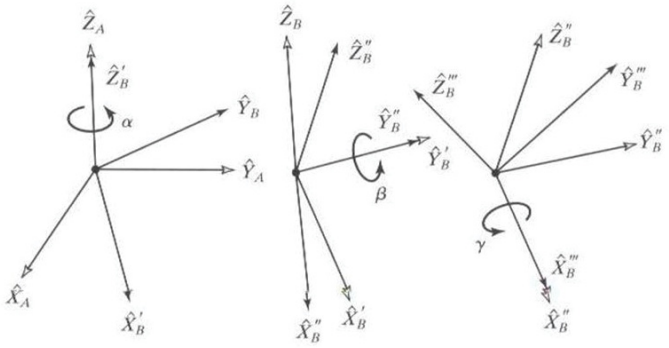

Figure 2.3: Euler Angles rotation for ZYX order.

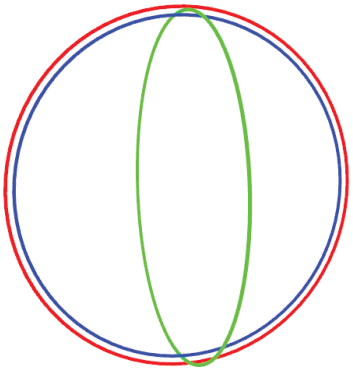

Figure 2.4: Gimbal lock example

Euler angles seem to be intuitive and easy to work with. However, they have a limitation called *gimbal lock*. This singularity is the loss of one degree of freedom in a three-gimbals system. It occurs when two out of the three gimbals are aligned, and a degree of freedom is lost. An example is shown in Fig. 2.4.

Depending on the application, the Euler rotation convention $XYZ, ZYX, ...$ can be changed to avoid the *gimbal lock* in the typical scenarios. In the study case of this project (Ch. 7) the convention will be the ZYX, which leads to a *gimbal lock* when $\theta = \pi/2$. However, the algorithms that are used (DCM and Madgwick) don't work with Euler Angles, so this won't be a problem at all. Substituting in Eq. 2.14 that value of θ the result is Eq. 2.15.

$$R = \begin{pmatrix} 0 & c(\phi)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\psi) + s(\phi)s(\psi) \\ 0 & s(\phi)s(\psi) + c(\phi)c(\psi) & s(\phi)c(\psi) - c(\phi)s(\psi) \\ -1 & 0 & 0 \end{pmatrix} \tag{2.15}$$

Using some basic trigonometric relations Eq. 2.15 becomes Eq. 2.16. In this expression it can be seen that changing the values ϕ, ψ, has the same effect. One degree of freedom has been lost. In this case a *pitch* of 90 degrees will lead to a *gimbal lock*.

$$R = \begin{pmatrix} 0 & -\sin(\phi - \psi) & \cos(\phi - \psi) \\ 0 & \cos(\phi - \psi) & \sin(\phi - \psi) \\ -1 & 0 & 0 \end{pmatrix} \tag{2.16}$$

2.3 Quaternions

Quaternions were introduced by Hamilton (1843) and are also used to represent the attitude of a rigid body in the space. They can be viewed as an extension of complex numbers. A complex number can be used to represent the rotation in a two dimensional space. A quaternion is similar, but instead of one axis it is extended to three axes. For its representation, four values are needed, where there is one real component q_0 and three imaginary q_1, q_2, q_3 shown in Eq. 2.17.

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{2.17}$$

It is also represented as a vector matrix $\mathbf{q}$, shown in Eq. 2.18, where q_1, q_2, q_3 are multiplied by i, j, k respectively.

$$\mathbf{q} = [q_0 \; q_1 \; q_2 \; q_3]^t \tag{2.18}$$

The quaternion that represents a rotation of θ degrees around an axis defined by the vector $\mathbf{n} = [n_x \; n_y \; n_z]$ is given by the Eq. 2.19.

$$\mathbf{q} = [n_x i \; n_y j \; n_z k] \sin(\theta/2) + \cos(\theta/2) \tag{2.19}$$

This means that a rotation of $\theta = \pi/2$ around the axis $\mathbf{n} = [1\ 0\ 0]$ will correspond to a quaternion given by the expression in Eq. 2.20.

$$\mathbf{q} = \left[\frac{\sqrt{2}}{2} \; \frac{\sqrt{2}}{2} \; 0 \; 0 \right] \tag{2.20}$$

A quaternion is often represented as a normalized vector. It can be normalized using Eq. 2.21, where $||q||$ can be calculated using Eq. 2.22.

$$\mathbf{q}_{norm} = \frac{q}{||q||} \tag{2.21}$$

$$||q|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{2.22}$$

Different operations can be done with quaternions. Adding two quaternions $q_a + q_b$ can be done by adding each element of the vector. Multiplying by a scalar, is done by multiplying the scalar by each element, like if it were a simple two or three dimensional vector. One of the most important operation is the product. Let $(a_1 + b_1 i + c_1 j + d_1 k)$ be the q_a quaternion and $(a_2 + b_2 i + c_2 j + d_2 k)$ the second quaternion q_b. They can be multiplied $q_a q_b$ by using the Hamilton product which is given by the expression in Eq. 2.23. Note that the product does not have the

commutative property, so $q_a q_b$ is not the same as $q_b q_a$.

$$\begin{aligned}(a_1 + b_1 i + c_1 j + d_1 k)(a_2 + b_2 i + c_2 j + d_2 k) = \\ a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 \\ +(a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) i \\ +(a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2) j \\ +(a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2) k\end{aligned} \tag{2.23}$$

The Eq. 2.24 determine all the possible combinations of i, j, k multiplications, so for example $k = ij$ or $ji = -k$

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.24}$$

The Hamilton product is very interesting for the scope of this project, because it can combine two rotations into only one. For example if the quaternion in Eq. 2.20, which represent a rotation of $\theta = \pi/2$ in the axis $\mathbf{n} = [1\ 0\ 0]$ is used two times, the result rotation will be $\theta = \pi$ in the same axis. Doing the maths, the Hamilton product of that quaternion multiplied with itself, gives Eq. 2.25.

$$\mathbf{q} = [0\ 1\ 0\ 0] \tag{2.25}$$

Which is the same as substituting $\theta = \pi$ and $\mathbf{n} = [1\ 0\ 0]$ in Eq. 2.19. Both results are the same.

Now that the main ways of representing the attitude of a body in the space were explained, it is important to know how to change between different representations. For this project the ZYX or 3,2,1 conversion will be used, and the most important expressions are given. Using Eq. 2.26 Euler angles $\theta_1, \theta_2, \theta_3$ can be converted to the quaternion representation.

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \sin(\theta_1/2)\sin(\theta_2/2)\sin(\theta_3/2) + \cos(\theta_1/2)\cos(\theta_2/2)\cos(\theta_3/2) \\ -\sin(\theta_1/2)\sin(\theta_2/2)\cos(\theta_3/2) + \sin(\theta_3/2)\cos(\theta_1/2)\cos(\theta_2/2) \\ \sin(\theta_1/2)\sin(\theta_3/2)\cos(\theta_2/2) + \sin(\theta_2/2)\cos(\theta_1/2)\cos(\theta_3/2) \\ \sin(\theta_1/2)\cos(\theta_2/2)\cos(\theta_3/2) - \sin(\theta_2/2)\sin(\theta_3/2)\cos(\theta_1/2) \end{pmatrix} \tag{2.26}$$

In Eq. 2.27 the conversion between quaternion to rotation matrix is given and in Eq. 2.28 the expression for converting from quaternion to Euler is given.

$$R = \begin{pmatrix} 2q_0^2 - 1 + 2q_1^2 & 2q_1 q_2 + q_0 q_3 & 2q_1 q_3 - q_0 q_2 \\ 2q_1 q_2 - q_0 q_3 & 2q_0^2 - 1 + 2q_2^2 & 2q_2 q_3 + q_0 q_1 \\ 2q_1 q_3 + q_0 q_2 & 2q_2 q_3 - q_0 q_1 & 2q_0^2 - 1 + 2q_3^2 \end{pmatrix} \tag{2.27}$$

$$\begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} \arctan(\dfrac{2q_2q_3 - q_0q_1}{2q_0^2 - 1 + 2q_3^2}) \\ \arctan\left(\dfrac{2q_1q_3 + q_0q_2}{\sqrt{1 - (2q_1q_3 + q_0q_2)^2}}\right) \\ \arctan(\dfrac{2q_1q_3 - q_0q_3}{2q_0^2 - 1 + 2q_1^2}) \end{pmatrix} \tag{2.28}$$

Chapter 3

Sensors

In this chapter, the sensors used for the project are described: accelerometers in Sec 3.1, gyroscopes in Sec. 3.2 and magnetometers in Sec. 3.3), explaining what they can measure and how to get them calibrated. Then, in Sec. 3.4 the most important parameters of the sensors are explained, which are very important to select one for an application. To conclude, in Sec. 3.5 a first approach on how to calculate Euler angles is given without using sensor fusion (using each sensor separately), highlighting the limitations of each method.

3.1 Accelerometer

This sensors are used in many fields and industries: aerospace, biology, structural monitoring, medical applications, navigation, transport or consumer electronics. In the study case of this project, it will be used to determine the orientation of a body in three dimensions.

An accelerometer [35] is a device or sensor that measures acceleration in g or m/s^2 where $1g$ is equivalent to $9.8m/s^2$. Note that it measures the total acceleration, so even if the device is static, it will measure $-1g$ pointing to the z axis. If it were in free fall, the acceleration would be $0g$. A typical accelerometer can measure acceleration in x, y, z axis. In Fig. 3.1 a generic accelerometer is represented, where the ball inside moves if there is any acceleration, and the walls are pressure sensitive.In Fig. 3.2 the sensor is static, but as it was explained before, the earth gravity causes an acceleration of $1g$ pointing to the floor.

If the device is moved $\pi/4$ degrees, the $1g$ acceleration pointing to the floor will still exist, but will affect the sensor in a different way, because it has been rotated. This gives a first idea on how to calculate the orientation of a body using only an accelerometer, but this only will be valid if there are not other external accelerations.

It is very important to calibrate the sensors before using them. For the accelerometer case, it is quite simple and can be done with only few calculations

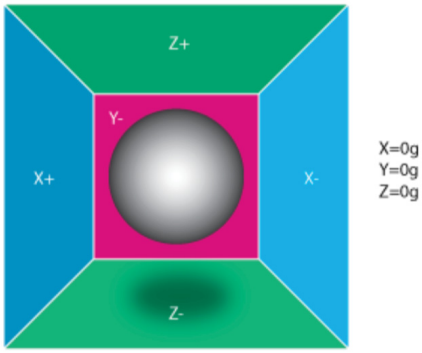

Figure 3.1: Generic accelerometer.

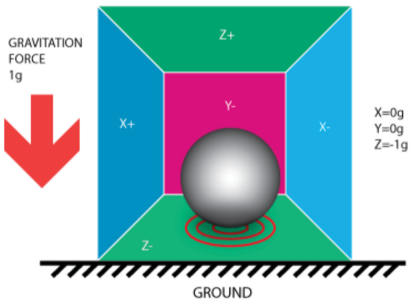

Figure 3.2: Static accelerometer measuring gravity

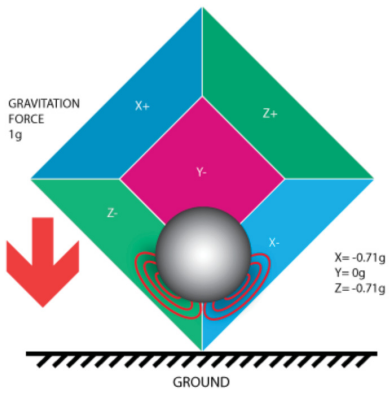

Figure 3.3: Rotated static accelerometer measuring gravity.

that won't take much CPU time. Note that these operations have to be done in each iteration, right after reading the sensor values. The first step is to know if the accelerometer readings are correct. This can be done by using the gravity acceleration of $1g$ as a reference. So first put the accelerometer in a flat surface and measure the acceleration. If the sensor were ideal, the measurement should be $-1g$. And if the sensor is turned around the value will be $1g$. However, this won't be always true. There is always some errors due to mechanical characteristics, humidity, pressure of temperature. Knowing this, the values we have to get are the following. The minimum will be close to -1g and the maximum to +1g:

- A^X_{min}, A^X_{max}
- A^Y_{min}, A^Y_{max}
- A^Z_{min}, A^Z_{max}

Once minimum and maximum values are known, it is time to calculate the offset using Eq. 3.1. Let A^X_O, A^Y_O, A^Z_O be the offsets for each axis.

$$\begin{pmatrix} A^X_O \\ A^Y_O \\ A^Z_O \end{pmatrix} = \begin{pmatrix} (A^X_{min} - A^X_{max})/2 \\ (A^Y_{min} - A^Y_{max})/2 \\ (A^Z_{min} - A^Z_{max})/2 \end{pmatrix} \tag{3.1}$$

Note that if the sensor were ideal, A_{min}, A_{max} would be the same and the offsets would be zero. Next step is to calculate the scale factor A^X_S, A^Y_S, A^Z_S given by Eq. 3.2.

$$\begin{pmatrix} A^X_S \\ A^Y_S \\ A^Z_S \end{pmatrix} = \begin{pmatrix} \frac{gravity}{A^X_{max} - A^X_O} \\ \frac{gravity}{A^Y_{max} - A^Y_O} \\ \frac{gravity}{A^Z_{max} - A^Z_O} \end{pmatrix} \tag{3.2}$$

Once offset and scale factor values are calculated, the last step is to correct the measurements by subtracting the offset and scaling the value. Expression is given in Eq. 3.3. Let $A^X_{meas}, A^Y_{meas}, A^Z_{meas}$ be the measurements for x, y, z axis before applying calibration and $A^X_{cal}, A^Y_{cal}, A^Z_{cal}$ the calibrated measurements.

$$\begin{pmatrix} A^X_{cal} \\ A^Y_{cal} \\ A^Z_{cal} \end{pmatrix} = \begin{pmatrix} (A^X_{meas} - A^X_O)A^X_S \\ (A^Y_{meas} - A^Y_O)A^Y_S \\ (A^Z_{meas} - A^Z_O)A^Z_S \end{pmatrix} \tag{3.3}$$

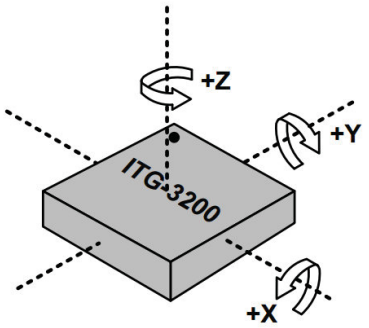

Figure 3.4: Typical IC low cost gyroscope.

3.2 Gyroscope

Gyroscopes are another type of sensors [9], but they measure angular speeds in $degrees/s$ or rad/s. They are also used in many fields such as aerospace, navigation or consumer electronics.

A typical gyroscope measures angular speed in x, y, z axis. An example can be found in Fig. 3.4. So if the sensor is static or moving straight ahead in any axis, the output value must be zero. Knowing this, one might think that gyroscopes can be used to determine the orientation of a body, and that is true but with some limitations. Later on in the thesis, Euler angles calculation using gyroscope will be explained using integration.

Gyroscopes have also to be calibrated. The calibration is performed by subtracting the offset value to each axis. This has to be done because even if the device is static and without any movement, the output values won't be exactly zero. There always will be some degrees of offset that had to be corrected. It is common to average some thousands of samples to get the gyroscope offsets G_O^X, G_O^Y, G_O^Z. Note also that $G_{meas}^X, G_{meas}^Y, G_{meas}^Z$ correspond with the measurements before calibration and $G_{cal}^X, G_{cal}^Y, G_{cal}^Z$ are the calibrated values. See Eq. 3.4.

$$\begin{pmatrix} A_{cal}^X \\ A_{cal}^Y \\ A_{cal}^Z \end{pmatrix} = \begin{pmatrix} A_{meas}^X - G_O^X \\ A_{meas}^Y - G_O^Y \\ A_{meas}^Z - G_O^Z \end{pmatrix} \tag{3.4}$$

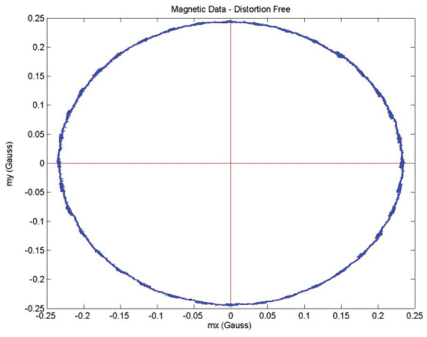

Figure 3.5: Ideal measurements for a magnetometer with two axes.

3.3 Magnetometer

A magnetometer is a device that measures magnetic field, typically expressed in Tesla T or Gauss G. In the study case of this project, a triple axis magnetometer x, y, z will be used. The application will be to measure the Earth magnetic field, which will allow to determine the orientation in the yaw plane. Although magnetometers are a great tool, they are very influenced by its surroundings. If some kind of metal or magnets are close to the magnetometer, the measurements will drift and become unreliable. However some effects can be fixed with a good calibration.

The magnetometer is the most important sensor to calibrate properly. There are different ways of doing it, but the most common is based on correcting the soft iron and hard iron effects. Assuming only two axis x, y for the sake of simplicity, an ideal plot of the measurements is displayed in Fig. 3.5, where no distortion effects are present. However, in real scenarios where metallic or magnetic devices can be close to the magnetometer, this won't happen.

There are different ways of calibrating a magnetometer [38], [42], [29] but the used for this project is the described in [21][6].

3.3.1 Hard Iron

The hard iron effect is a distortion produced by materials that have a constant field. This will cause that the magnetometer readings will measure the real value plus an additional constant field. A hard iron effect is displayed in Fig 3.6, where there is an offset.

Hard iron effects are easy to correct. The first step is to know the offset in each axis, and the next step is to subtract the offset to the real measurements. Note that α is the offset in the x axis and β is the offset in the y axis.

$$\alpha = \frac{x_{max} - x_{min}}{2} \tag{3.5}$$

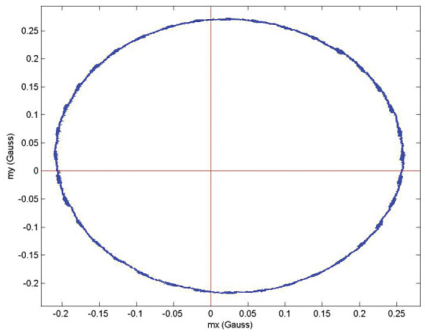

Figure 3.6: Hard iron effects in a magnetometer.

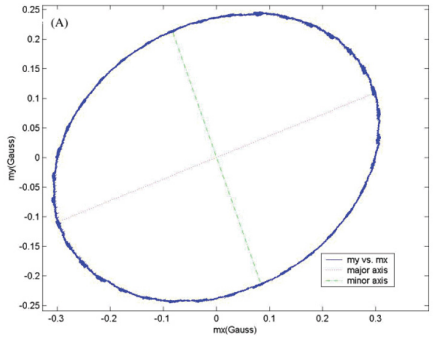

Figure 3.7: Soft iron effects in a magnetometer.

$$\beta = \frac{y_{max} - y_{min}}{2} \tag{3.6}$$

However, hard iron effects are not always enough to get accurate measurements from the magnetometer. For more accuracy, soft iron distortions are corrected.

3.3.2 Soft Iron

Soft iron distortion is produced by materials that distorts the magnetic field, but is not additive. An example of metals that cause that effects are iron or nickel. The main problem of this type of distortion is that it is dependent on the orientation of the material. In 3.7 soft iron effects are shown, where it can be seen that the shape is now an ellipse.

In order to correct the soft iron effects, the first step is to rotate the ellipse to have the major axis aligned with the x axis. This can be done in two dimensions using rotation matrices (Sec. 2.2). In 3.8 an example is displayed, and θ, r can be calculated using Eq. 3.7 and Eq. 3.8.

$$r = \sqrt{x_1^2 + y_1^2} \tag{3.7}$$

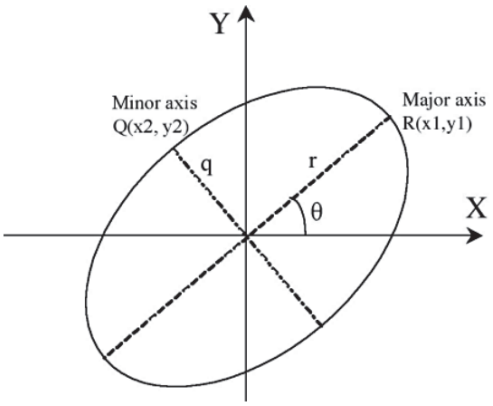

Figure 3.8: Soft iron calibration using an ellipsoid.

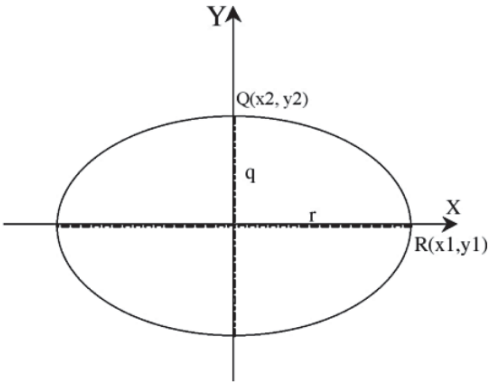

Figure 3.9: Soft calibration after rotating the ellipsoid.

$$\theta = \arcsin(\frac{y_1}{r}) \tag{3.8}$$

Once the ellipse is rotated, next step is to scale the major axis in order to convert it to a circle. Let q, r be the mayor and minor axis shown in 3.9 and σ the scale factor calculated according to Eq. 3.9.

$$\sigma = \frac{q}{r} \tag{3.9}$$

For this thesis, three dimensional calibration is needed for the three magnetometer axis x, y, z. Instead of an ellipse and ellipsoid will be used, which is the expansion to three dimension and is described by nine parameters A, B, C, D, E, G, H, I as shown in 3.10.

$$Ax^2 + By^2 + Cz^2 + 2Dxy + 2Exz + 2Fyz + 2Gx + 2Hy + 2Iz \tag{3.10}$$

Now let $M_{cal}^X, M_{cal}^Y, M_{cal}^Z$ be the magnetometer calibrated values calculated using Eq. 3.11, $M_{meas}^X, M_{meas}^Y, M_{meas}^Z$ the measured values, $e_{11}...e_{33}$ the ellipsoid transform matrix and C^X, C^Y, C^Z the center values of the ellipsoid. See [6] for further information.

$$\begin{pmatrix} M_{cal}^X \\ M_{cal}^Y \\ M_{cal}^Z \end{pmatrix} = \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} \begin{pmatrix} M_{meas}^X - C^X \\ M_{meas}^Y - C^Y \\ M_{meas}^Z - C^Z \end{pmatrix} \tag{3.11}$$

3.4 Sensor Parameters

When choosing a sensor for a particular application, it is important to know the meaning of some parameters that will determine the capacities and limitations of the sensor. Some of them are:

- Range: A range in a sensor indicates what are the minimum and maximum values that it can measure. If the real variable goes above or below that value, the output value will be limited to the maximum, so the measurement won't be correct. It is expressed in the same units that the sensor measures, g for acceleroemter, $degrees/s$ for gyroscope and G for magnetometer.

- Sensitivity: It is defined as the slope of the characteristic curve of the sensor, which is the relation between the input variable (i.e. an acceleration) and the output variable (i.e. a voltage). So its unit is determined by the variable that the sensor measures and the output that it gives. In digital sensors it is common to give the sensibility in LSB/u o u/LSB, where u is the unit that the sensor measures. This means that when the LSB bit changes, the real variable changes in u units.

- Precision: It is related to the degree of reproducibility of a measurement, so a very precise sensor will output the same (i.e. voltage) when the same input is given. Note that is not the same as accuracy.

- Accuracy: It is the difference that exist between the real value and the measured one. Note that one sensor can be very precise but not accurate.

- Noise: It will depend on the internal characteristics of the sensor and the operation mode. It is typically measured in LSB rms but in some applications it is given as a function of the operation frequency.

- Rate: Indicates the sampling frequency of the device, measured in Hz. So it is the number of samples that the sensor can take per second. This parameter can usually be chosen among different values.

There are also other characteristics such as: power or current consumption, temperature of operation, supply voltages, connection interface, dimensions or weight. They all can be checked in the data sheet of the sensor.

3.5 Euler Angles without fusion

In this section, techniques to calculate euler angles using each sensor separately will be explained, highlighting they drawbacks and limitations.

3.5.1 Accelerometer

Pitch and roll angles can be calculated using only one accelerometer [30]. Note that yaw estimation is not possible using this technique. Pitch is given by Eq. 3.12 in degrees and Roll by Eq. 3.13. Remember from previous sections that $A^X_{cal}, A^Y_{cal}, A^Z_{cal}$ are the x, y, z calibrated values of the accelerometer.

$$pitch^A = \arctan\left(-\frac{A^X_{cal}}{\sqrt{(A^Y_{cal})^2 + (A^Z_{cal})^2}}\right)\frac{180}{\pi} \tag{3.12}$$

$$roll^A = \arctan\left(\frac{-A^Y_{cal}}{A^Z_{cal}}\right)\frac{180}{\pi} \tag{3.13}$$

This way is very simple and does not require many operations. However, the pitch/roll calculations will only be valid when there are no external accelerations in the device. If the device is moving up and down, the pitch and roll will become unreliable. This will be discussed in next sections.

3.5.2 Gyroscope

Using only the angular velocity given by the gyroscope, pitch/roll/yaw can be calculated with a very simple operation [43]. It is well known the relation between the position of an object and its velocity by the expression Eq. 3.14, where the position x is the integral of the velocity v plus the start point x_0.

$$x(t) = x_0 + \int_0^t v(t)dt \tag{3.14}$$

In this case, angular speed in deg/s is known, and position in deg is the desired variable. In discrete time, it can be calculated using Eq. 3.15.

$$\theta = \theta + w\Delta t \tag{3.15}$$

Where θ is the angle in deg, w is the angular speed in deg/s and Δt is the time between measurements. Remembering from previous sections that $G^X_{cal}, G^Y_{cal}, G^Z_{cal}$ are the measurements from the gyroscope after calibration for x, y, z axis, pitch/roll/yaw can be calculated as indicated in Eq. 3.16.

$$\begin{pmatrix} pitch_{n+1} \\ roll_{n+1} \\ yaw_{n+1} \end{pmatrix} = \begin{pmatrix} pitch^G_n + G^X_{cal}\Delta t \\ roll^G_n + G^Y_{cal}\Delta t \\ yaw^G_n + G^X_{cal}\Delta t \end{pmatrix} \tag{3.16}$$

Note that in the case of this project the axis are aligned so x corresponds with the pitch, y with the roll and z with the yaw, but other set up will lead to a different order.

3.5.3 Magnetometer

The yaw or heading parameter can be obtained from a three axis magnetometer measurements using Eq.3.17 where $M_{cal}^X, M_{cal}^Y, M_{cal}^Z$ are the x, y, z calibrated measurements from the magnetometer.

$$yaw = \arctan\left(\frac{M_{cal}^Y}{M_{cal}^X}\right) \tag{3.17}$$

However, this way of getting the yaw is not accurate, because if the sensor is tilted (pitch or roll different from zero) the result won't be valid. To solve this, there is a method called tilt compensation [33].

Chapter 4

Sensor Fusion

Sensor fusion consists in combining different data derived from different sensors in order to get a resulting variable that has less uncertainty than using the sources individually. In this chapter different sensor fusion techniques to estimate the orientation of a body , using a three-axis accelerometer, gyroscope and magnetometer are explained. The fist one and simplest is the Complimentary Filter described in Sec. 4.1, which only uses the gyroscope and accelerometer. The second one is the DCM algorithm found in Sec. 4.2 that uses a rotation matrix and a PI controller. Last algorithm based on quaternions is the Madgwick one explained in Sec. 4.3, which although of being the most complex, it is implemented very efficiently.

4.1 Complimentary Filter

In chapter Ch. 3 different ways of estimating pitch roll and yaw using different sensors was explained. However, these techniques only use data from one sensor to do the calculations, which has some limitations. Complimentary filter [10], [43] is a first simple sensor fusion algorithm that combines the power of the accelerometer and gyroscope. Pitch and roll is corrected using these two sensors, and yaw (also called heading) is taken only from the gyroscope. Remembering from last section, let $roll^A, pitch^A$ be the pitch and roll using the accelerometer explained in Eq. 3.12,3.13 and $roll^G, pitch^G, yaw^G$ the pitch, roll and yaw values using the gyroscope in Eq. 3.16. The Euler angles $pitch^C, roll^C, yaw^C$ using a complimentary filter can be calculated using Eq. 4.1.

$$\begin{pmatrix} pitch^C_{n+1} \\ roll^C_{n+1} \\ yaw^C_{n+1} \end{pmatrix} = \begin{pmatrix} (pitch^C_n + G^X_{cal}\Delta t)\alpha + (1-\alpha)pitch^A_n \\ (roll^C_n + G^Y_{cal}\Delta t)\alpha + (1-\alpha)roll^A_n \\ yaw^G \end{pmatrix} \tag{4.1}$$

Where α is a value between 0 and 1, typically with values greater than 0.9. The complimentary filter is a kind of filter, where the accelerometer is low pass and gyroscope high pass. This is done because the gyroscope is accurate in short time, but drifts in long time. In the other hand, accelerometer is accurate in long

time but not in short periods of time. Note that yaw is not corrected by the accelerometer.

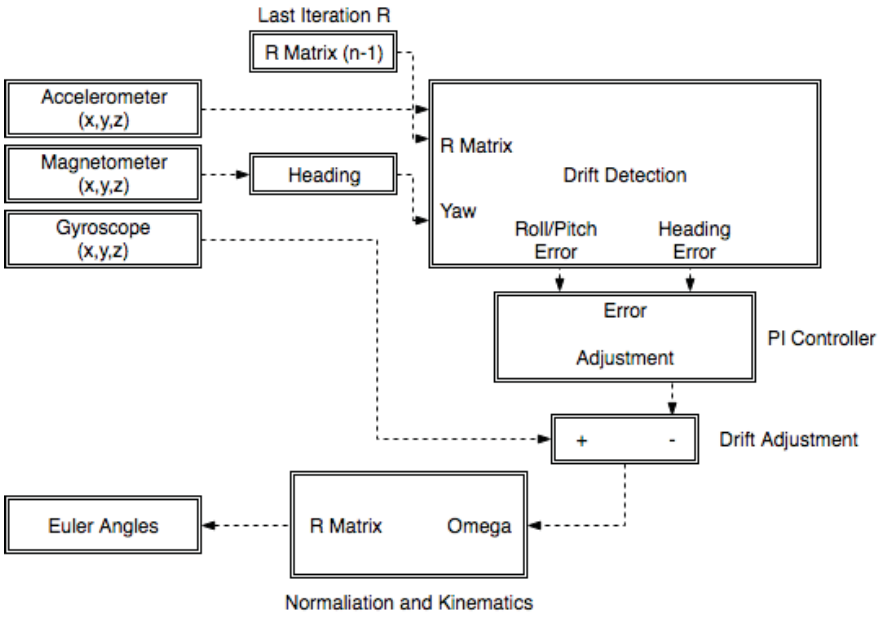

Figure 4.1: Schematic of the DCM sensor fusion algorithm.

4.2 DCM

Other way of calculating the orientation of a rigid body in the space, is using the DCM (*Direction Cosine Matrix*) algorithm [5], [14], [37], [36], [32]. It uses a rotation matrix (see Sec. 2.1) for calculating the rotation, and at the end Euler angles are obtained using Eq. 2.12. In Fig. 4.1 a diagram of the algorithm is presented. Note that the core of the algorithm is the rotation matrix denoted by R that is calculated using gyroscope data and corrected using the magnetometer and accelerometer. A PI control system is used to correct the drift, where P corresponds to the proportional term and I to the integrative one. At the end of each iteration, Euler angles are calculated using the DCM matrix.

The algorithm can be divided in the following steps, assuming that the input data is calibrated:

- Initialize DCM: This part is only for the first iteration. It will initialize the DCM matrix with the first available data. See Sub. 4.2.1.
- Calculate heading: Calculate the heading using the magnetometer. See Sub. 4.2.2.
- Update matrix: Update the DCM matrix. See Sub. 4.2.3.
- Normalize: Keep the vectors orthogonal and normalize them. See Sub. 4.2.4.

- Correct drift: Correct the drift of the gyroscope using a PI control. See Sub. 4.2.5.
- Calculate Euler: Calculate Euler angles from DCM matrix. See Sub. 4.2.6

4.2.1 Initialize DCM

The first step is to initialize the DCM matrix. This is done by using the following expression explained in previous section Eq. 2.14. Remember that this rotation matrix assumes the ZYX convention explained before. To feed the equation, pitch/roll/yaw values are calculated as follows. Pitch is calculated using the accelerometer data, explained in Sec. 3.5.1 in Eq. 3.12. Roll can be calculated using Eq. 3.13 but as stated in [4] it can also be calculated using 4.2. Note that $\vec{u}_x$ is the unitary vector in x direction, A_{cal} is a vector that contains the accelerometer calibrated values and $\times$ is the cross product.

$$aux = \vec{u}_x \times (A_{cal} \times \vec{u}_x) \tag{4.2}$$

Using the calculated aux vector, the roll can be calculated using Eq. 4.3. Note that aux_y and aux_z are the second and third components of the vector.

$$roll = \arctan\left(\frac{aux_y}{aux_z}\right) \tag{4.3}$$

Last step to feed the matrix, is calculating the yaw. This step is explained in 4.2.2.

4.2.2 Calculate heading

In each iteration of the algorithm, the heading value (also referred as yaw) is calculated. Calibrated values of the magnetometer $M^X_{cal}, M^Y_{cal}, M^Z_{cal}$ are used. See Eq. 4.4. Note that $c(), s()$ is equivalent to $\cos(), \sin()$.

$$yaw = \arctan\left(\frac{-M^Y_{cal}c(roll) + M^Z_{cal}s(roll)}{M^X_{cal}c(pitch) + M^Y_{cal}s(roll)s(pitch) + M^Z_{cal}c(roll)s(pitch)}\right) \tag{4.4}$$

4.2.3 Update matrix

Next step is to update the DCM matrix with the values of Ω_I, Ω_P calculated in the last iteration. These values will be zero in the first iteration, because the are calculated in next steps (see Sub. 4.2.5). First, using Eq. 4.5, Ω is calculated. This new value of omega is the gyroscope values after drift correction. 4.5

$$\Omega = G_{cal} + \Omega_I + \Omega_P \tag{4.5}$$

To understand the following steps, is necessary to explain some facts of the rotation matrix. Having calibrated gyroscope data $G^X_{cal}, G^Y_{cal}, G^Z_{cal}$ that after drift correction using Eq. 4.5 become Ω with components $\Omega_x, \Omega_y, \Omega_z$, the rotation matrix update can be calculated with Eq. 4.6.

$$R(t+dt) = R(t)\begin{pmatrix} 1 & -\Omega_z dt & \Omega_y dt \\ \Omega_z dt & 1 & -\Omega_x dt \\ -\Omega_y dt & \Omega_x dt & 1 \end{pmatrix} \tag{4.6}$$

So in each step, the rotation matrix will be updated using the read gyroscope values and corrected according to the PI controller that will be explained then.

4.2.4 Normalize

This step is very important to the performance of the algorithm. Remember from Seq. 2.1 the properties of a rotation matrix. By definition a rotation matrix is orthogonal, which means that $R^t R = RR^t = I$. The problem is that numerical errors can keep accumulating and break the orthogonality condition, which will result in a bad performance because the matrix will no longer represent a rotation.

In order to avoid this, in each iteration two steps are done: first make the vectors orthogonal again, second normalize them. The Gram-Schmidt method for orthogonalization is a very well known process, but for this purpose a different approach is taken. The used method does not fix one of the vector and calculated the others. Both vectors are changed, applying to each one half of the error.

First step is to compute the X and Y orthogonal vectors of the rotation matrix. This can be done by using the Eq. 4.7 , 4.8.

$$X_{orthogonal} = X - \frac{error}{2}Y \tag{4.7}$$

$$Y_{orthogonal} = Y - \frac{error}{2}X \tag{4.8}$$

The error is calculated as the dot product of $X \cdot Y$. Remember that the dot product of $a \cdot b$ is calculated as $|a||b|\cos(\theta)$, where θ is the angle between both vectors. If they were orthogonal, $\cos(\theta)$ would be zero and the error would be also zero.

Next step is to calculate the Z orthogonal vector to X and Y. This is done by calculating the cross product between X and Y $X \times Y$. Remember the geometric representation of the cross product in Fig. 4.2.

The last step is to re normalize the calculated vectors X, Y, Z. This step is trivial and can be done by dividing the vector by the Euclidean norm. However,

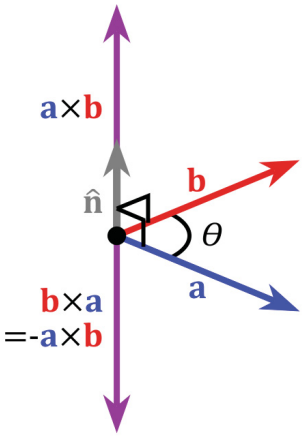

Figure 4.2: Geometric representation of cross product.

a different approach presented in [32] is used. This way assumes that the norm is not much greater than one, so Taylor's expansion can be used. The norm of a vector $\vec{u}$ using this approach is shown in Eq. 4.9.

$$\vec{u}_{norm} = \frac{1}{2}(3 - \vec{u} \cdot \vec{u})\vec{u} \tag{4.9}$$

4.2.5 Correct drift

Next step is to correct the drift of the gyroscope. The accelerometer will be used to correct the pitch and roll values, and the magnetometer to correct the yaw or heading. For correcting the drift, a PI controller will be used where P stands for proportional and I for integrative. A PI controller is similar to a PID controller, which is a very well known control system. See Eq. 4.10, where $e(t)$ is the error.

$$y(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \tag{4.10}$$

A PI control will only have the proportional and integrative term, like in Eq. 4.11. In the case of study of this project there will be two different controllers: one for the pitch/roll and other for the yaw. The constants corresponding to the integral and proportional term will be $K^P_{pitch/roll}, K^I_{pitch/roll}$ and K^P_{yaw}, K^I_{yaw}

$$y(t) = K_p e(t) + K_i \int e(t)dt \tag{4.11}$$

First step is to calculate the error in pitch and roll. This can be calculated using the cross product between the acceleration calibrated vector A_{cal} and the last row of the DCM matrix $DCM_{31}, DCM_{32}, DCM_{33}$. See Eq. 4.12.

$$error_{pitch/roll} = A_{cal} \times (\, DCM_{31} DCM_{32} DCM_{33} \,) \tag{4.12}$$

Next step is to calculate the correction terms Ω_P, Ω_I that will be used to correct the gyroscope measurements using Eq. 4.13, 4.14. Note that $|\vec{A_{cal}}|$ is the Euclidean norm of the acceleration vector. It is recommended to constrain this value between 0 and 1. This will reduce the error when the sensor is under big accelerations, i.e. moving it with the hand.

$$\Omega_P = error_{pitch/roll} K^P_{pitch/roll} |\vec{A_{cal}}| \tag{4.13}$$

$$\Omega_I = \Omega_I + error_{pitch/roll} K^I_{pitch/roll} |\vec{A_{cal}}| \tag{4.14}$$

Now it is time to correct the yaw drift using the yaw obtained by the magnetometer using Eq. 4.15.

$$error_{yaw} = (\, DCM_{31} DCM_{32} DCM_{33} \,) \, (DCM_{11} \sin(yaw) - DCM_{21} \cos(yaw)) \tag{4.15}$$

And now it is time to add the contribution of the yaw to Ω_I, Ω_P using Eq. 4.16, 4.17.

$$\Omega_P = \Omega_P + error_{yaw} K^P_{yaw} \tag{4.16}$$

$$\Omega_I = \Omega_I + error_{yaw} K^I_{yaw} \tag{4.17}$$

In the next iteration this values will be used to correct the drift of the gyroscope values, as explained in Sub. 4.2.3.

4.2.6 Calculate Euler

Once this step has reached, there is nothing much left. Just calculate the Euler angles from the updated rotation matrix. This can be calculated using the Eq. 2.12 explained before. After this section the algorithm will run from the beginning again.

4.3 Madgwick

The Madgwick algorithm (Sebastian 2010) [24][23] is an orientation filter that can be applied to IMU (accelerometer + gyroscope) or MARG (accelerometer + gyroscope + magnetometer). It allows to get the orientation of the device using a quaternion representation, that can also be transformed intro Euler angles θ, ϕ, ψ. One of the benefits that this filter include against the typical Kalman filter approach is the low computational complexity. The IMU version only requires 109 scalar arithmetic operations per update, and the MARG 277, which makes the Madgwick algorithm very convenient for low size, cost and energy consumption microprocessors. The author reports a good behavior even using low sampling rates, close to 10 Hz. Different implementations of the algorithm are provided in different programming languages [26], [27],[25] ,[41]. Some improvements and error corrections of the algorithm can be found in [1],[34].

The DCM algorithm presented in Sec. 4.2, used a rotation matrix to compute the orientation, and at the end of each iteration the Euler angles were computed using that matrix. However, the Madgwick algorithm uses other approach, using the quaternion representation to determine the orientation.

To properly understand the algorithm, the quaternion representation described in Ch. 2 must be understood. There is also some important notation that will be explained. For example, the letter $\boldsymbol{q}$ describes a quaternion with four components and ${}^{A}_{B}\hat{\boldsymbol{q}}$ describes the quaternion that represent the orientation of frame B relative to frame A. Also similar, ${}^{A}\hat{\boldsymbol{r}}$ represents a vector described in frame A. Also, the Hamilton product explained before is denoted as $\otimes$ and the conjugate of the quaternion ${}^{A}_{B}\hat{\boldsymbol{q}}$ is denoted as ${}^{A}_{B}\hat{\boldsymbol{q}}^{*}$

Before going in depth in the maths behind the Madgwick filter, it is important to understand properly what can different sensors provide. For the study case of this thesis, the MARG approach will be used, so it is assumed that an accelerometer, gyroscope and magnetometer are present in the device. For simplification, the data is assumed to be calibrated as explained in Ch. 3 where the accelerometer and gyroscope are calibrated for the offset and magnetometer is calibrated with soft and hard iron effects. Knowing that, these are the steps that will be followed to obtain the estimated quaternion in each iteration. Note that S stands for *sensor* and E for *earth*, so the quaternion is always describing the rotation of the sensor frame in relation to the earth.

- First of all the gyroscope measurements will be used to compute the quaternion denoted by ${}^{S}_{E}\boldsymbol{q}_{\omega,t}$. It represent the rotation at time t described by the angular speed measured in that iteration.
- Next step is to use the measurements provided by the accelerometer and

magnetometer to get the orientation estimation. It is very important to note that these two sensors can't estimate the full orientation θ, ϕ, ψ, so they have to be combined to get the quaternion ${}^{S}_{E}\boldsymbol{q}_{\nabla,t}$. This is done by formulating an optimization problem, that will be solved using the gradient descent algorithm that will be explained later.

- Finally, using both estimations explained before ${}^{S}_{E}\boldsymbol{q}_{\omega,t}$ and ${}^{S}_{E}\boldsymbol{q}_{\nabla,t}$, a quaternion will be obtained by using a sensor fusion filter similar to the complimentary filter to get ${}^{S}_{E}\boldsymbol{q}_{est,t}$. This last quaternion will express the orientation of the body in each time t, and it is almost trivial to get the Euler angles from it.

As said before, first step is to compute the orientation from the gyroscope. In Eq. 4.18 data from the gyroscope in $rads^{-1}$ is arranged in ${}^{s}\boldsymbol{\omega}$. The orientation ${}^{S}_{E}\boldsymbol{q}_{w,t}$ can be calculated by integrating the quaternion derivative ${}^{S}_{E}\dot{\boldsymbol{q}}_{w,t}$ as presented in Eq. 4.19, 4.20. Note that Δt is the sampling period, ${}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}$ is the previous estimation and ${}^{s}\boldsymbol{\omega}_{t}$ is the gyroscope measurement at time t.

$$ {}^{s}\boldsymbol{\omega} = [0 \quad w_x \quad w_y \quad w_z] \tag{4.18} $$

$$ {}^{S}_{E}\dot{\boldsymbol{q}}_{w,t} = \frac{1}{2} {}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1} \otimes {}^{s}\boldsymbol{\omega}_{t} \tag{4.19} $$

$$ {}^{S}_{E}\boldsymbol{q}_{w,t} = {}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1} + {}^{S}_{E}\dot{\boldsymbol{q}}_{w,t}\Delta t \tag{4.20} $$

Next step is to use the acceleration provided by the accelerometer to get the orientation quaternion. Note that an accelerometer will measure not only the earth gravity but also the linear accelerations due to the motion of the sensor. This is a problem that will be explained in Ch. 6 because it is assumed that the accelerometer only measures gravity.

Using that the direction of the earth's field is known in the earth frame, a measurement of the sensor will allow to determine the orientation relative to the earth frame. However, note that the solution won't be a unique value, but a line of infinite points. This is because the accelerometer can't estimate the heading, only the pitch and roll.

Let ${}^{S}_{E}\hat{\boldsymbol{q}}$ be the orientation of the sensor as described in Eq. 4.21, ${}^{E}\hat{\boldsymbol{d}}$ the predefined reference direction of the field in the earth frame described in Eq. 4.22 and ${}^{S}\hat{\boldsymbol{s}}$ the measurement of the field in the sensor frame described by Eq. 4.23, which in this case is the acceleration values.

$$ {}^{S}_{E}\hat{\boldsymbol{q}} = [q_1 \quad q_2 \quad q_3 \quad q_4] \tag{4.21} $$

$$^{E}\hat{\boldsymbol{d}} = [0 \quad d_x \quad d_y \quad d_z] \tag{4.22}$$

$$^{S}\hat{\boldsymbol{s}} = [0 \quad s_x \quad s_y \quad s_z] \tag{4.23}$$

The estimation $^{S}_{E}\hat{\boldsymbol{q}}$ can be formulated as an optimization problem described by Eq. 4.24, where the objective function is Eq. 4.25.

$$\min_{^{S}_{E}\hat{\boldsymbol{q}} \in \Re^4} \quad f(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}, ^{S}\hat{\boldsymbol{s}}) \tag{4.24}$$

$$f(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}, ^{S}\hat{\boldsymbol{s}}) = {}^{S}_{E}\hat{\boldsymbol{q}}^{*} \otimes {}^{E}\hat{\boldsymbol{d}} \otimes {}^{S}_{E}\hat{\boldsymbol{q}} - {}^{S}\hat{\boldsymbol{s}} \tag{4.25}$$

There are different ways of solving that optimizing algorithm, but the gradient descent is one of the simplest to implement and compute. In Eq. 4.26 the algorithm is displayed, where $k = 0, 1, 2, n$ is the number of iteration, $^{S}_{E}\boldsymbol{q}_{k+1}$ is the orientation estimation based on the previous one $^{S}_{E}\boldsymbol{q}_{k}$ and μ is the step size. Note that ∇ stands for gradient and it can be calculated using the Jacobian matrix as shown in Eq. 4.27.

$${}^{S}_{E}\boldsymbol{q}_{k+1} = {}^{S}_{E}\boldsymbol{q}_{k} - \mu \frac{\nabla f(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}, ^{S}\hat{\boldsymbol{s}})}{\|\nabla f(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}, ^{S}\hat{\boldsymbol{s}})\|} \tag{4.26}$$

$$\nabla f(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}, ^{S}\hat{\boldsymbol{s}}) = J^{T}(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}) f(^{S}_{E}\hat{\boldsymbol{q}}, ^{E}\hat{\boldsymbol{d}}, ^{S}\hat{\boldsymbol{s}}) \tag{4.27}$$

Doing the maths, the objective function $f_g(^{S}_{E}\hat{\boldsymbol{q}})$ can be calculated as shown in Eq. 4.28 and the Jacobian as displayed in Eq. 4.29. Note that one simplification has been made to reach these expression, which is the following. The direction of the field $^{E}\hat{\boldsymbol{d}}$ only has the d_z component non zero and equal to one, so the solution is not dependent of that value.

$$f_g(^{S}_{E}\hat{\boldsymbol{q}}, ^{S}\hat{\boldsymbol{a}}) = \begin{pmatrix} 2(q_2 q_4 - q_1 q_3) - a_x \\ 2(q_1 q_2 + q_3 q_4) - a_y \\ 2(0.5 - {q_2}^2 - {q_3}^2) - a_z \end{pmatrix} \tag{4.28}$$

$$J_g(^{S}_{E}\hat{\boldsymbol{q}}) = \begin{pmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{pmatrix} \tag{4.29}$$

Now it is turn to the magnetometer. The used approach to get the objective function and Jacobian matrix will be quite similar to the acceleration case. In this case the earth's magnetic field can be considered to have two components, one in the horizontal axis and one in the vertical. This vector $^{E}\hat{\boldsymbol{b}}$ is represented

in Eq. 4.30 where there are only two non zero components. Measurements of the magnetometer ${}^{S}\hat{\boldsymbol{m}}$ are represented in Eq. 4.31.

$$ {}^{E}\hat{\boldsymbol{b}} = [0 \quad b_x \quad 0 \quad b_z] \tag{4.30} $$

$$ {}^{S}\hat{\boldsymbol{m}} = [0 \quad m_x \quad m_y \quad m_z] \tag{4.31} $$

Following the same steps as before, the objective function $f_b({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}})$ and Jacobian matrix $J_b({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}})$ are shown in Eq. 4.32, 4.33 respectively.

$$ f_b({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}) = \begin{pmatrix} 2b_x(0.5 - {q_3}^2 - {q_4}^2) + 2b_z(q_2q_4 - q_1q_3) - m_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 + q_3q_4) - m_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z(0.5 - {q_2}^2 - {q_3}^2) - m_z \end{pmatrix} \tag{4.32} $$

$$ J_b({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}) = \begin{pmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & 2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{pmatrix} \tag{4.33} $$

As discussed before none of the measurements taken by the accelerometer or magnetometer can provide a full solution to the orientation of the body, because the solution is not unique. Remember that the accelerometer can't estimate the heading. To achieve a complete solution, both sensors have to be combined.

$$ f_{g,b}({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{S}\hat{\boldsymbol{a}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}) = \begin{pmatrix} f_g({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{S}\hat{\boldsymbol{a}}) \\ f_b({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}) \end{pmatrix} \tag{4.34} $$

$$ J_{g,b}({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}) = \begin{pmatrix} J_g({}^{S}_{E}\hat{\boldsymbol{q}}) \\ J_b({}^{S}_{E}\hat{\boldsymbol{q}}, {}^{E}\hat{\boldsymbol{b}}) \end{pmatrix} \tag{4.35} $$

So the solution to the estimated orientation can be calculated using the gradient descent algorithm shown in Eq. 4.36, where ${}^{S}_{E}\boldsymbol{q}_{\nabla,t}$ is the orientation at time t. The subscript ∇ denotes that the orientation is computed using the gradient descent.

$$ {}^{S}_{E}\boldsymbol{q}_{\nabla,t} = {}^{S}_{E}\boldsymbol{q}_{est,t-1} - \mu\frac{\nabla f}{\|\nabla f\|} \tag{4.36} $$

$$ \nabla f = J^{T}_{g,b}({}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}, {}^{E}\hat{\boldsymbol{b}}) f_{g,b}({}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}, {}^{S}\hat{\boldsymbol{a}}, {}^{E}\hat{\boldsymbol{b}}, {}^{S}\hat{\boldsymbol{m}}) \tag{4.37} $$

Note that a conventional optimization problem runs for n iterations until a certain error is minimized or a number of iterations are completed. However, in this case as the author explains, it is enough to compute one iteration per time

sample. Also the step size μ should be recalculated in each iteration but in order to save calculations, it can be set as Eq. 4.38.

$$\mu = \alpha \| {}^{S}_{E}\dot{\boldsymbol{q}}_{\omega,t} \| \Delta t, \quad \alpha > 1 \tag{4.38}$$

Doing the maths, the final expression for $\nabla f_{g,b}$ is shown in Eq. 4.39, 4.40, 4.41, 4.42. Note that it is a four component vector so that $\nabla f_{gb} = [\nabla f^0_{g,b} \nabla f^1_{g,b} \nabla f^2_{g,b} \nabla f^3_{g,b}]$. Demonstration can be seen in enclosed Matlab file.

$$\begin{aligned} \nabla f^0_{g,b} = {} & 8{b_x}^2 q_2 q_3 q_4 + 4 q_1 {b_x}^2 {q_3}^2 - 4 q_1 {b_x}^2 {q_4}^2 + 8 b_x b_z q_3 {q_4}^2 - 2 m_z b_x q_3 - \\ & 2 m_y b_x q_4 + 4 q_1 {b_z}^2 {q_2}^2 + 4 q_1 {b_z}^2 {q_3}^2 - 2 m_y b_z q_2 + 2 m_x b_z q_3 + \\ & 4 q_1 {q_2}^2 - 2 a_y q_2 + 4 q_1 {q_3}^2 + 2 a_x q_3 \end{aligned} \tag{4.39}$$

$$\begin{aligned} \nabla f^1_{g,b} = {} & 2 q_1 \left(2 q_1 q_2 - a_y + 2 q_3 q_4\right) - \\ & \left(2 b_x q_3 + 2 b_z q_1\right) \left(m_y + 2 b_x \left(q_1 q_4 - q_2 q_3\right) - 2 b_z \left(q_1 q_2 + q_3 q_4\right)\right) - \\ & \left(2 b_x q_4 - 4 b_z q_2\right) \left(m_z - 2 b_x \left(q_1 q_3 + q_2 q_4\right) + 2 b_z \left({q_2}^2 + {q_3}^2 - \frac{1}{2}\right)\right) - \\ & 2 q_4 \left(a_x + 2 q_1 q_3 - 2 q_2 q_4\right) + 4 q_2 \left(2 {q_2}^2 + 2 {q_3}^2 + a_z - 1\right) - \\ & 2 b_z q_4 \left(m_x + 2 b_z \left(q_1 q_3 - q_2 q_4\right) + 2 b_x \left({q_3}^2 + {q_4}^2 - \frac{1}{2}\right)\right) \end{aligned} \tag{4.40}$$

$$\begin{aligned} \nabla f^2_{g,b} = {} & 2 q_1 \left(a_x + 2 q_1 q_3 - 2 q_2 q_4\right) - \\ & \left(2 b_x q_2 + 2 b_z q_4\right) \left(m_y + 2 b_x \left(q_1 q_4 - q_2 q_3\right) - 2 b_z \left(q_1 q_2 + q_3 q_4\right)\right) + \\ & \left(4 b_x q_3 + 2 b_z q_1\right) \left(m_x + 2 b_z \left(q_1 q_3 - q_2 q_4\right) + 2 b_x \left({q_3}^2 + {q_4}^2 - \frac{1}{2}\right)\right) - \\ & \left(2 b_x q_1 - 4 b_z q_3\right) \left(m_z - 2 b_x \left(q_1 q_3 + q_2 q_4\right) + 2 b_z \left({q_2}^2 + {q_3}^2 - \frac{1}{2}\right)\right) + \\ & 2 q_4 \left(2 q_1 q_2 - a_y + 2 q_3 q_4\right) + 4 q_3 \left(2 {q_2}^2 + 2 {q_3}^2 + a_z - 1\right) \end{aligned} \tag{4.41}$$

$$\begin{aligned} \nabla f^3_{g,b} = {} & \left(2 b_x q_1 - 2 b_z q_3\right) \left(m_y + 2 b_x \left(q_1 q_4 - q_2 q_3\right) - 2 b_z \left(q_1 q_2 + q_3 q_4\right)\right) \\ & - 2 q_2 \left(a_x + 2 q_1 q_3 - 2 q_2 q_4\right) + \\ & \left(4 b_x q_4 - 2 b_z q_2\right) \left(m_x + 2 b_z \left(q_1 q_3 - q_2 q_4\right) + 2 b_x \left({q_3}^2 + {q_4}^2 - \frac{1}{2}\right)\right) + \\ & 2 q_3 \left(2 q_1 q_2 - a_y + 2 q_3 q_4\right) - 2 b_x q_2 \left(m_z - 2 b_x \left(q_1 q_3 + q_2 q_4\right) + 2 b_z \left({q_2}^2 + {q_3}^2 - \frac{1}{2}\right)\right) \end{aligned} \tag{4.42}$$

Now that ${}^{S}_{E}\boldsymbol{q}_{\omega,t}$ and ${}^{S}_{E}\boldsymbol{q}_{\nabla,t}$ are known, next step is to merge both to get the final estimation of the orientation ${}^{S}_{E}\boldsymbol{q}_{est,t}$. This fusion is done by using Eq. 4.43.

$${}^{S}_{E}\boldsymbol{q}_{est,t} = \gamma_t {}^{S}_{E}\boldsymbol{q}_{\nabla,t} + (1-\gamma_t){}^{S}_{E}\boldsymbol{q}_{\omega,t} \quad 0 \leq \gamma_t \leq 1 \tag{4.43}$$

After some simplifications and assumptions [24] the expression in Eq. 4.43 can be simplified to Eq. 4.44 and it is the equation that will be running in the Arduino and is the core of the algorithm.

$${}^{S}_{E}\boldsymbol{q}_{est,t} = -\frac{\beta \nabla f \Delta t}{\|\nabla f\|} + {}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1} + {}^{S}_{E}\dot{\boldsymbol{q}}_{w,t}\Delta t \tag{4.44}$$

Note that β represents the divergence rate of the measurement from the gyroscope, so it has to be adjusted for different sensors and sampling frequencies. Note also that ∇f is a four dimensional vector which components were shown before in Eq. 4.39, 4.40, 4.41, 4.42. Also Δt is the sampling period, ${}^{S}_{E}\hat{\boldsymbol{q}}_{est,t-1}$ is the previous estimation and ${}^{S}_{E}\dot{\boldsymbol{q}}_{w,t}$ was described in Eq. 4.19.

If necessary, the last step is to convert from quaternions to Euler angles representation, which is more convenient and intuitive. That can be done as explained before in Eq. 2.28.

Chapter 5

Experimental Setup

In this chapter the experimental setup of the thesis is explained and is divided into three parts. The first one explains all the components used for the project and budget (Seq. 5.1), the second one describes the hardware setup (Sec. 5.2), and the last part explains the most important facts about the software (Sec. 5.3).

5.1 Components and budget

The objective of the the thesis is to test the performance of different sensor fusion algorithms for calculating the attitude of a body in a three dimensional space. For this purpose, and as was explained before three different sensors are used: accelerometer, gyroscope and magnetometer. So in one side these three sensors and needed, but also a microprocessor able to perform the fusion. After considering different options, Arduino has been chosen as the microprocessor, using also a three in one accelerometer, gyroscope magnetometer called SparkFun Sensor Stick.

Arduino is a very well known company that manufactures different kind of microprocessors or microcontrollers with different sizes and for different purposes, but with one thing in common, they are open source. There is a huge community behind, that is continually improving the product and collaborating with each other in forums. One of the most important factors that lead to chose Arduino as a developing platform is that it can be programmed in C through the Arduino IDE, which can save a lot of time comparing with other kind of microcontrollers.

Particularly, Arduino MKR1000 [3] (see Fig. 5.1) was chosen because of its reduced size and integrated WiFi module, which makes it very suitable for IoT applications.

The SparkFun 9DOF Sensor Stick [13] shown in Fig. 5.2 was chosen because it gathers into one single small board the three sensors that are needed. It can be connected to any device with only four pins, which are: ground, voltage, SCA and SCL. This last two pins allow to interact with the sensor to send (i.e. configuration parameters) and receive data (i.e. different sensor measurements).

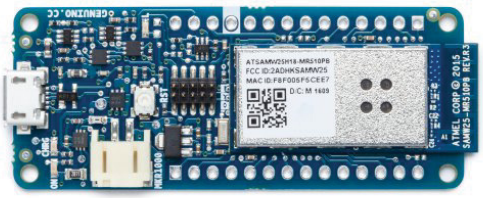

Figure 5.1: Arduino MKR1000 board.

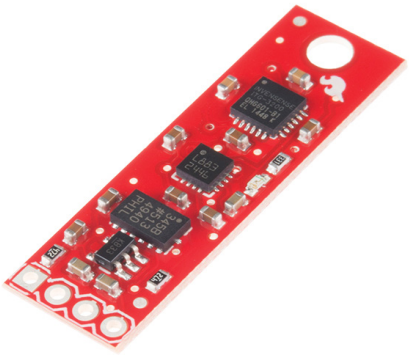

Figure 5.2: SparkFun 9DOF Sensor Stick.

This Sensor Stick has inside three different sensors whose specifications can be found in Table. 5.1[11], 5.2 [19], 5.3 [17] for the accelerometer, gyroscope and magnetometer respectively.

In Table 5.4 a budget for the project is presented. This budget also has a battery, which allows the Arduino and sensor to operate stand alone, without the need off being plugged into the USB. The battery has a capacity of 1400 mAh, which may be over sized for the project but was the only available at that moment. It is important to keep in mind that due to the Arduino battery circuit charger the battery has to be grater than 700 mAh or otherwise it can be damaged while charging. This happens because the charge is done at 350 mA, and the typical charge ratio of batteries is C/2.

Table 5.1: Accelerometer specifications

Accelerometer: ADXL345	
Interface	I2C or SPI
Extra	Tap detection, falling, activity.
Shock survival	10.000 g
Dimensions	3mm x 5mm x 1mm
Data rate	From 6 to 3200 Hz
Resolution	10 or 13 bit
Range	From +-2 g to +-16 g
Supply voltage	From 2.0 to 3.6 V
Weight	20 mg

Table 5.2: Gyroscope specifications

Gyroscope: ITG3200	
Interface	I2C
Extra	Temperature sensor
Shock survival	10000 g
Dimensions	4mm x 4mm x 0.9mm
Data rate	Up to 8 kHz
Sensibility	14,375 LSB/^{o}/s
Range	+- 2000 deg/s
Supply voltage	From 2.1 to 3.6 V

Table 5.3: Magnetometer specifications

Magnetometer: HMC5883L	
Interface	I2C
Dimensions	3mm x 3mm x 0.9mm
Data rate	From 0.75 to 75 Hz
Weight	18 mg
Range	+- 0.88 to +-8.1 Ga
Supply voltage	From 2.16 to 3.0 V

Table 5.4: Project budget

Budget (Swedish Kr)			
Component	Cost/Unit (Kr)	Units	Subtotal (Kr)
Arduino MKR1000	499	1	499
Sensor Stick	529	1	529
Battery	129	1	219
Total:			1157

5.2 Hardware Setup

The hardware setup can be seen in Fig. 5.3 where the sensor is connected to the Arduino through four pins, and a battery is connected to the Arduino to supply it with 3.7 V. Note that a simple switch was added to be able to switch on and off the device. The pins are the following:

- GND: Zero voltage reference displayed in black color.
- VCC: Supply voltage, 3.7 V in color red.
- SCL: Clock pin of the I2C protocol, displayed in green color.
- SDA: Data pin of the I2C protocol, presented in blue color.

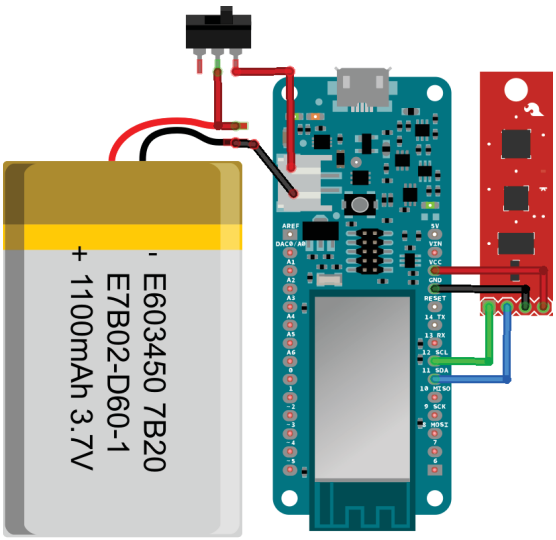

Figure 5.3: Schematic setup of the project.

In Fig. 5.4 and 5.5 the final experimental setup of the project can be seen. It can be seen that is composed by the Arduino MKR1000, the sensor board with the accelerometer, magnetometer, gyroscope, a battery and a switch to turn on and off the device.

5.3 Software

Once the hardware setup used for the project has been explained, it is turn to describe the software that will be running in the Arduino to test different algorithms. All the programming is done using the Arduino IDE (*Integrated Development Environment* which allows to program the Arduino in C. In Fig. 5.6

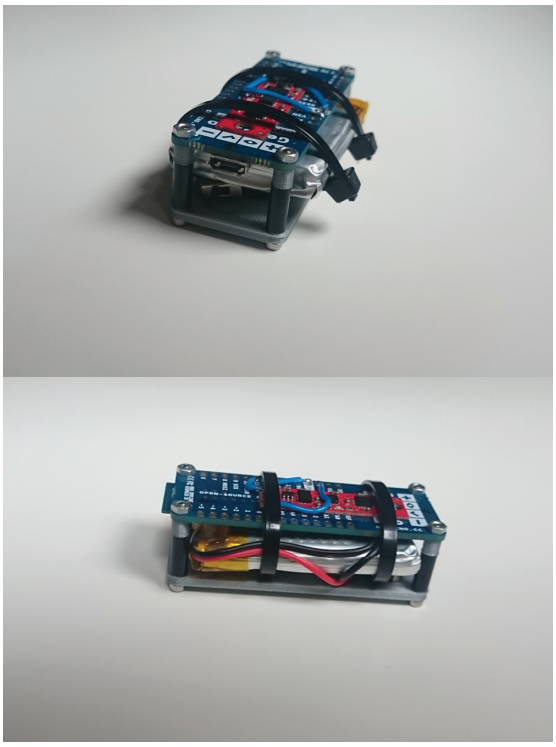

Figure 5.4: Hardware setup. Arduino, sensor, battery and switch.

the IDE is shown.

In order to interact with the sensors easily, the *Wire* library [20] is used. It provides high level functions to read/write from/to registers usin the I2C serial protocol. For sending data from the Arduino to the computer *WiFi 101* library, [28] is used. In [4] a function for calibrating the sensor is provided.

The flow of execution is shown in Fig. 5.7 where five major parts are presented. First the code is initialized, then data is read through I2C. After that each reading is calibrated before feeding the sensor fusion algorithm. Finally, the resulting data is sent through the serial port in UDP packets via WiFi. After that, different scripts in Matlab have been developed, in order to post process or plot the incoming data from the Arduino.

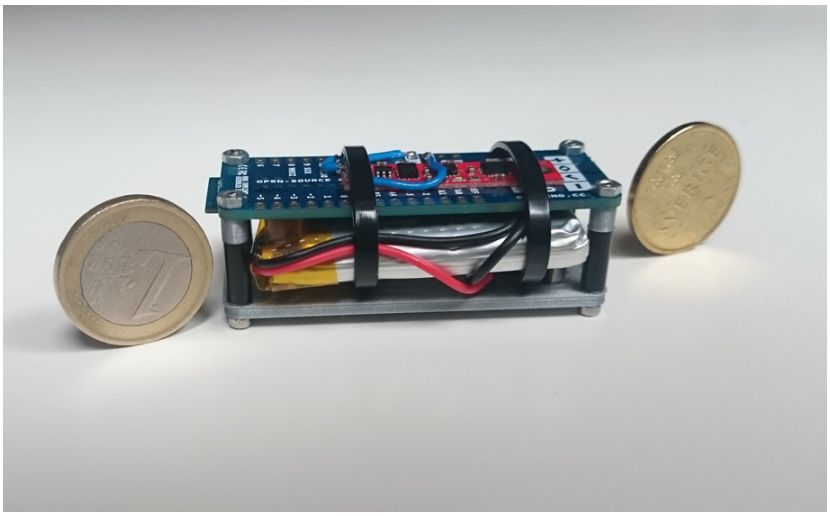

Figure 5.5: Hardware setup. Arduino, sensor, battery and switch in relation to a coin.

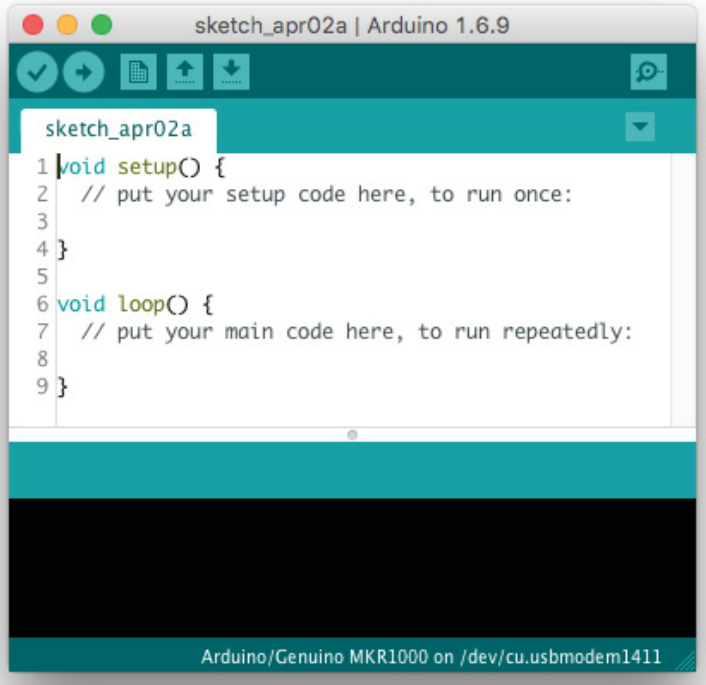

Figure 5.6: Developing IDE of Arduino.

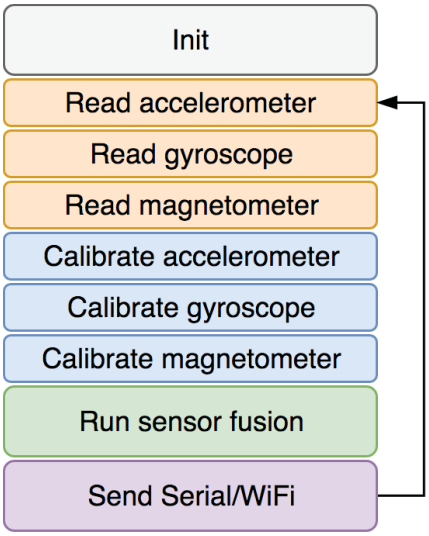

Figure 5.7: Schematic of the code running in the Arduino.

Chapter 6

Testing and Results

After having studied different sensor fusion techniques from a theoretical point of view in Ch. 4, in this chapter they are studied in an experimental way, using a specific sensor and an specific microprocessor. One of the main problems that was detected during the study of the algorithms, is the lack of an absolute and perfect reference to compare the algorithms. This can be done with quite expensive optic sensors, but unfortunately this was not possible. However, good results can also be archived by comparing the algorithms with each other using the same device.

In Sec. 6.1 sensor calibration is explained, which is a very important factor to take into account for accurate measurements. Secondly, in Sec. 6.2 Euler angles are obtained using only the accelerometer and in Sec. 6.3 using only the gyroscope. After that in Sec. 6.4 the complimentary filter is used and in Sec. 6.5 DCM and Madgwick algorithms are compared.

6.1 Calibration

Previously, in Ch. 3 calibration for the accelerometer, magnetometer and gyroscope was explained. Each sensor is calibrated through a different process in order to compensate different problems.

In the case of the accelerometer, an offset and a scale factor is used. These two parameters are calculated using the minimum and maximum values in each axis, and then corrected as shown in Eq. 3.3. Note that Eq. 3.1 is used to calculate the offset and Eq. 3.2 for the scale. In Table 6.1 the different minimum and maximum measured values of the gravity are displayed.

Table 6.1: Accelerometer calibration paremeters.

	X Axis	Y Axis	Z Axis
Minimum g	-1.028	-1.140	-1.228
Maximum g	+1.076	+1.120	+0.952

Table 6.2: Gyroscope calibration parameters.

	X Axis	Y Axis	Z Axis
Gyroscope offset (deg/s)	-0.97398	+5.63517	+0.41742

Gyroscope calibration it is more straight forward, as shown in Eq. 3.4. Only the offset of the sensor in each axis has to be measured, and then subtracted to the real measurements. That offset can be measured by leaving the sensor idle during some seconds and averaging the results. In Table 6.2 the offset for each axis is shown. Note that in different conditions (temperature, humidity, pressure) will give different offset values.

The magnetometer has also to be calibrated and is the most vulnerable sensor. Note that the magnetometer can be calibrated against soft or hard iron effects, but in the case of this project hard iron calibration was used, because it is more accurate and fewer more operations are needed. To perform this calibration, the sensor has to be rotated in all directions in order to get the value of the magnetic field in all direction in all axis. After that a three dimensional plot can be used to visualize the data. An ideal measurement will be a perfect sphere, but however due to external influences of ferromagnetic materials or external fields, that is not true. In Fig. 6.1 the measured field is represented in red, and in blue the compensated measurements after perform the ellipsoid fit calibration method (see Sec. 3.3.

One of the major problems when calibrating the magnetometer, was the presence of a WiFi module in the Arduino, close to the magnetometer. Also, having a battery close to the sensor introduced variations in the measurements. This results that the device has to be calibrated with the WiFi module switched on, and with the battery attached to the device. In Fig. 6.2 the difference of measurements with the WiFi turned on and off is displayed. The error is quite big, and in practice, if this calibration is not done, the output Euler angles are not reliable at all.

6.2 Accelerometer

As explained before in Eq. 3.12, 3.13 the accelerometer by itself can be used to estimate the attitude (pitch and roll) of an object. It can't be used to estimate the heading or yaw because of the nature of the gravity.

In Fig. 6.3 an example of pitch and roll estimation with the accelerometer is presented. The y-axis represents the angle and the x-axis the time. In this test the device was turned to the left and to the right, having it held with the hand.

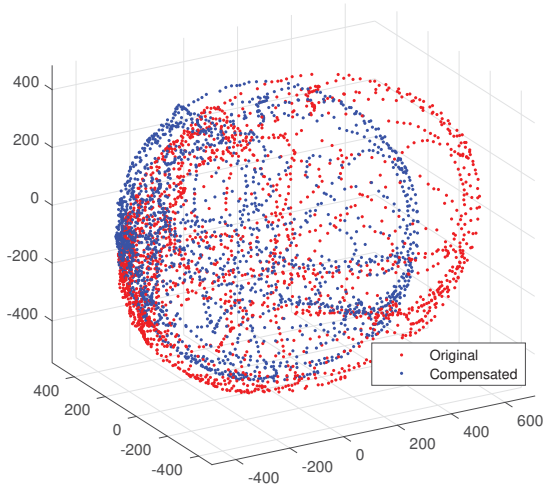

Figure 6.1: Magnetometer calibration, ellipsoid to sphere.

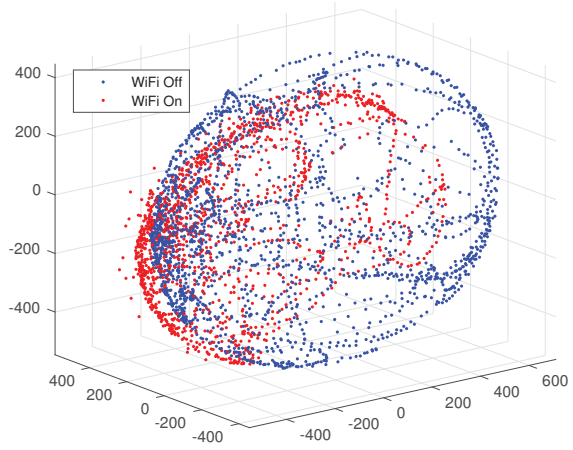

Figure 6.2: Magnetometer bias with and without WiFi module working.

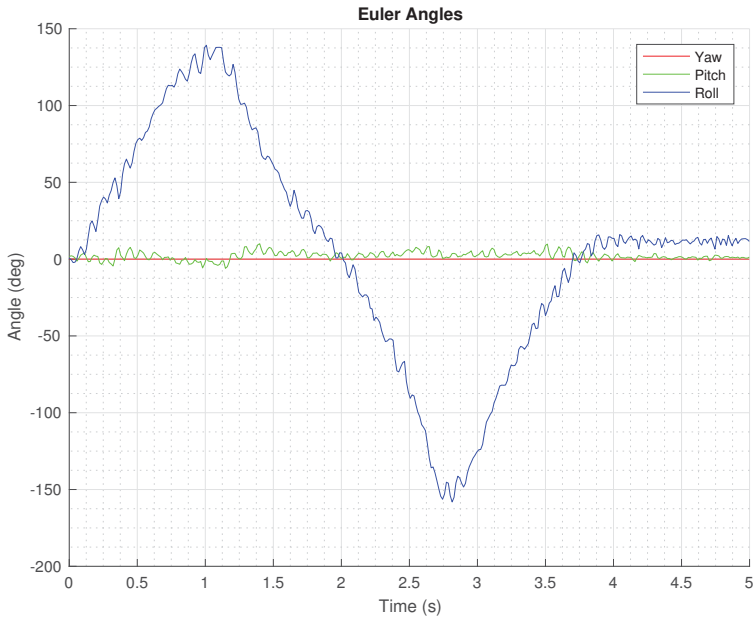

Figure 6.3: Euler estimation with accelerometer, testing the roll.

Same test but pitching to the front is represented in Fig. 6.3. As seen in the figure, when the pitch goes close to 90 degrees, the measurements start to randomly change. Note also that the roll should be zero but it is not. This happens because when the pitch is 90, the y and z-components should be zero but they are slightly bigger and have a bit of noise, which causes this behavior.

During the last tests shown in Fig 6.3, 6.4 the sensor was moved slowly, so the acceleration that the sensor was measuring was the pure earth gravity. In the test displayed in Fig 6.5 the sensor is moved upward and backward at a frequency in the order of 2 or 3 Hz. In this case the Euler angles should not change, because the device is moving along one axis without changing the orientation, but the results are noisy measurements that does not make any sense. When the accelerometer is corrupted by external accelerations, the formulas used to calculate the attitude are no longer valid. It can be concluded that the accelerometer by itself can be a first cheap approach (in terms of money and computational cost) but it is not suitable for applications where the heading is required, nor where the device is moving experimenting external accelerations.

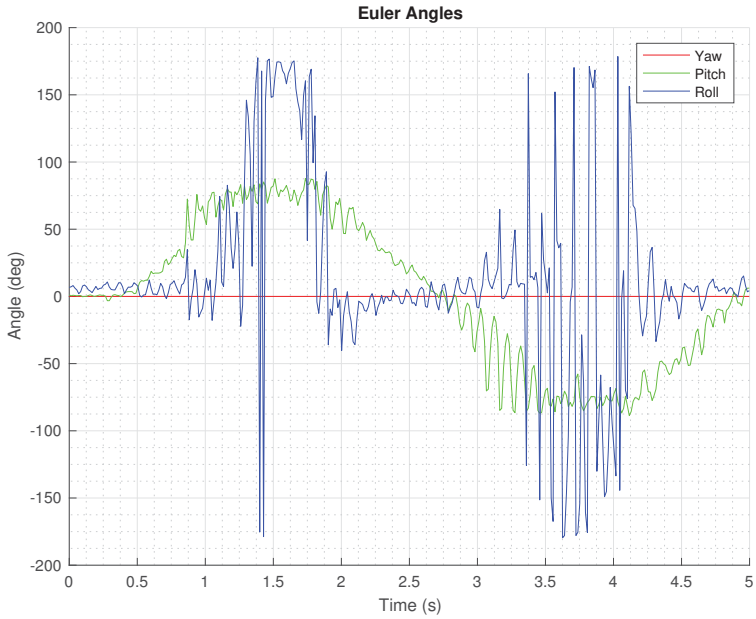

Figure 6.4: Euler estimation with accelerometer, testing the pitch.

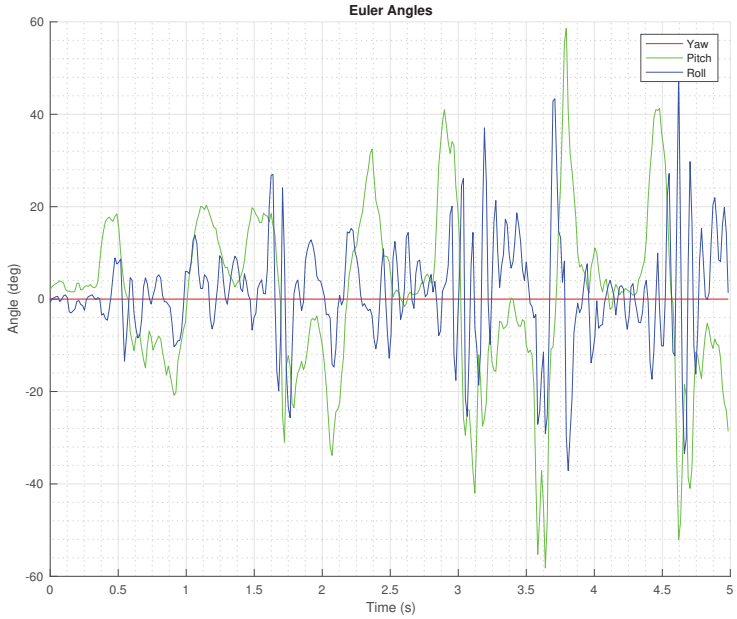

Figure 6.5: Euler estimation with accelerometer, moving the sensor forward and backwards.

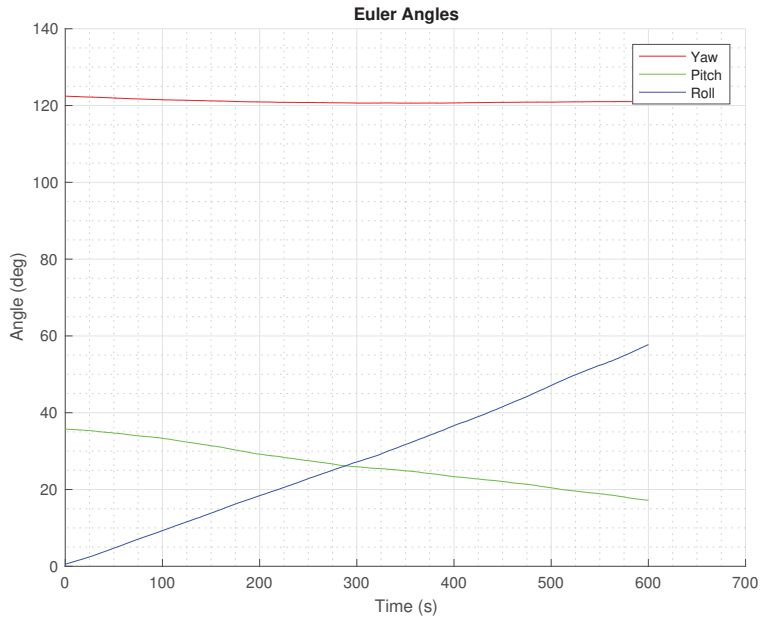

Figure 6.6: Gyroscope drift over time.

6.3 Gyroscope

As described with Eq. 3.16 the gyroscope can be used by itself to estimate the orientation of a body in terms of pitch, roll and yaw with simple integration. However, cheap MEMS sensors are not very accurate nor precise, so the integral is feed with the measurement x plus a small amount of noise n. After performing the integration, a small error is being added to the angle estimation in each iteration, which makes it to drift over time. It is refered as drift, the error over time that the gyro is accumulating. In Fig 6.6 and 6.7 the device was left during ten minutes in a static position. The expected output would be a constant value of pitch, roll and yaw, but due to the explained effect, there is a drift over time. For the yaw it is $1/10$ deg/min, for the pitch 2 deg/min and for the yaw 6 deg/min. This drift makes MEMS cheap gyroscope not suitable for many applications, specially when they are more than just few seconds.
Note that there are techniques that can estimate as a state the drift of the gyroscope and compensate, like Kalman filtering, but its computational complexity is much higher, and not always suitable for low cost microprocessors.

6.4 Complimentary filter

After the results using the accelerometer and the gyroscope, the complimentary filter technique is presented. It is a simple sensor fusion technique that allows

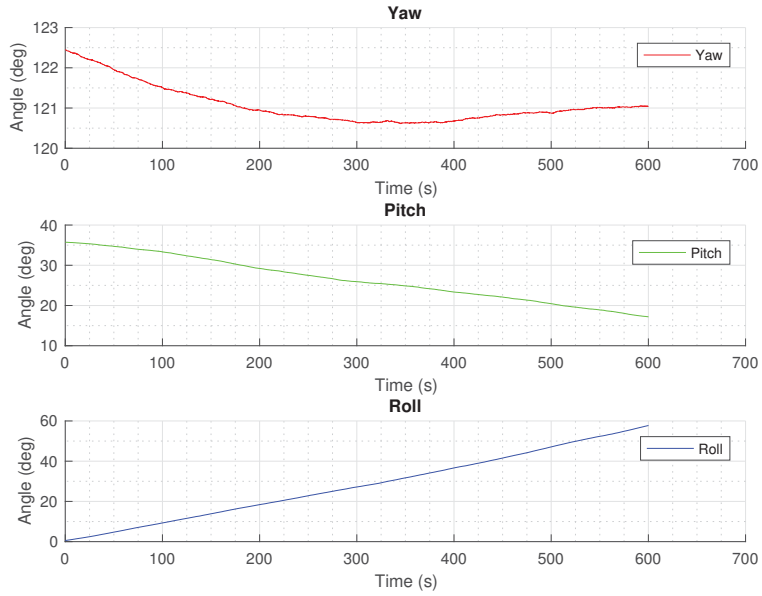

Figure 6.7: Gyroscope drift over time.

to take the best of each sensor. Note that this method can estimate the pitch and roll as a fusion but not the yaw, which is taken only from the gyroscope and which defects are already known.

The complimentary filter explained in Eq. 4.1 can be easily implemented in low cost microprocessors like the Arduino, and its computational complexity is very low. This technique takes the best of each sensor. It is known that the accelerometer is not accurate when external accelerations are applied, but it can be assumed that the accelerations are not always present, only sometimes. So it can be said that the accelerometer is accurate in long term measurements. However, the gyro can provide a very good estimation of the orientation in short periods of time, but not in longer ones because of the drift. The complimentary filter performs a kind of average between the accelerometer and gyroscope estimations. In Fig 6.8 an example using the complimentary filter is presented. For that test the sensor was moved along one axis forward and backward, to test the resilience to external accelerations. The angles should not variate because the orientation was not being changed, but however they are. It can be concluded that the complimentary filter is a good approach for estimating the orientation when there are not strong external accelerations, but even if there are, the estimation can give an order of magnitude of the position of the sensor. It is also important to note that the yaw is estimated only with the gyroscope, so it will drift.

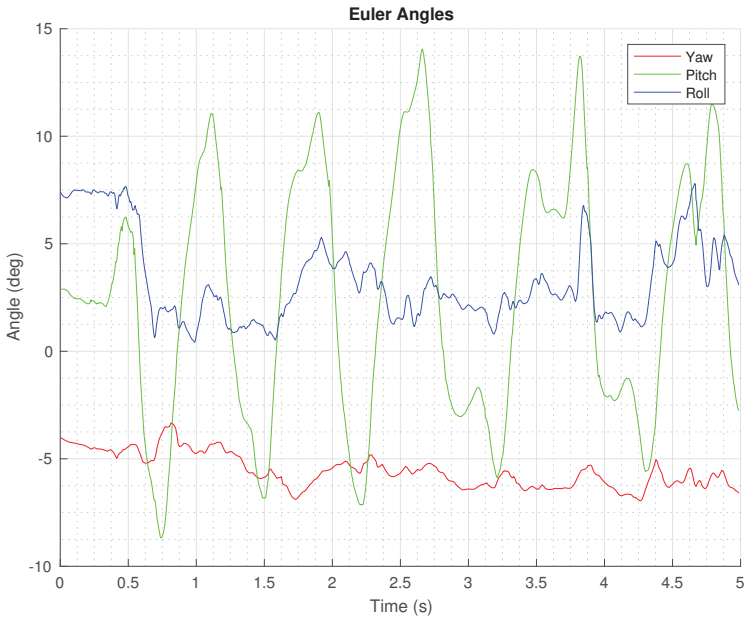

Figure 6.8: Euler estimation with complimentary filter, testing resilence to external accelerations.

6.5 DCM and Madgwick comparative

Previosly, in Ch. 4 DCM and Madgwick sensor fusion algorithms were explained from a mathematical point of view. In this section a comparative of both algorithms is provided, comparing them in terms of resilience to external accelerations, which is the major concern of this project. Three major tests were done:

- Static performance
- Slow movements performance
- Fast movements performance

In the first test both algorithms were tested with the device static and left in the table. In Fig 6.9 the output Euler angles of each sensor are displayed and in Fig. 6.10 the difference is plotted. Both algorithms seems to perform equally, and the DCM one looks a little bit more noisy. Note that neither of those algorithms have been post processed with filtering.

The second test will determine the behavior when the device is moved slowly, without applying big accelerations. This is equivalent to a person with the sensor attached walking around the room. In Fig. 6.11 the Euler angles over time are represented and in Fig. 6.12 the difference is plotted. The error is bigger than in

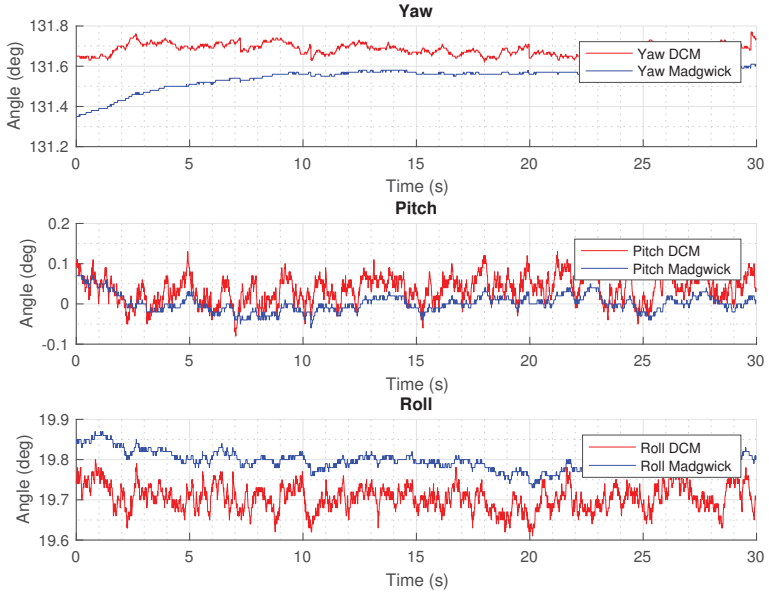

Figure 6.9: Static comparative between DCM and Madgwick.

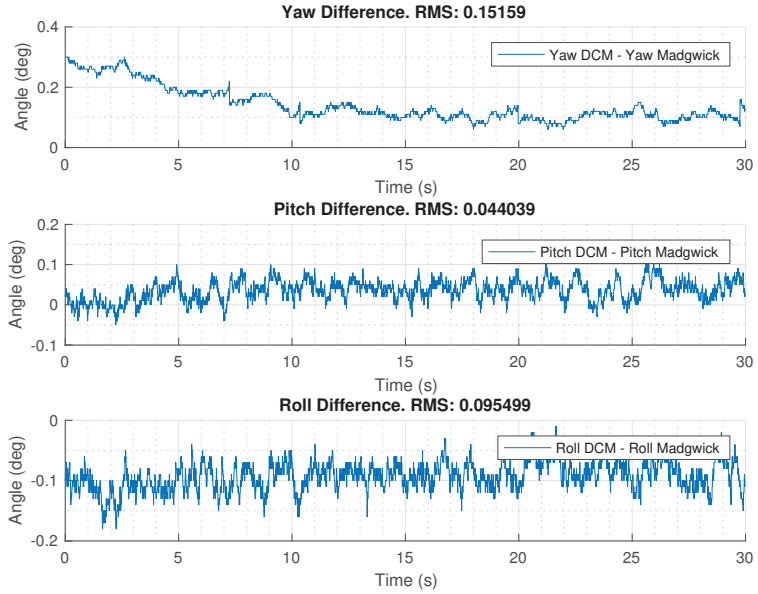

Figure 6.10: Static error between DCM and Madgwick.

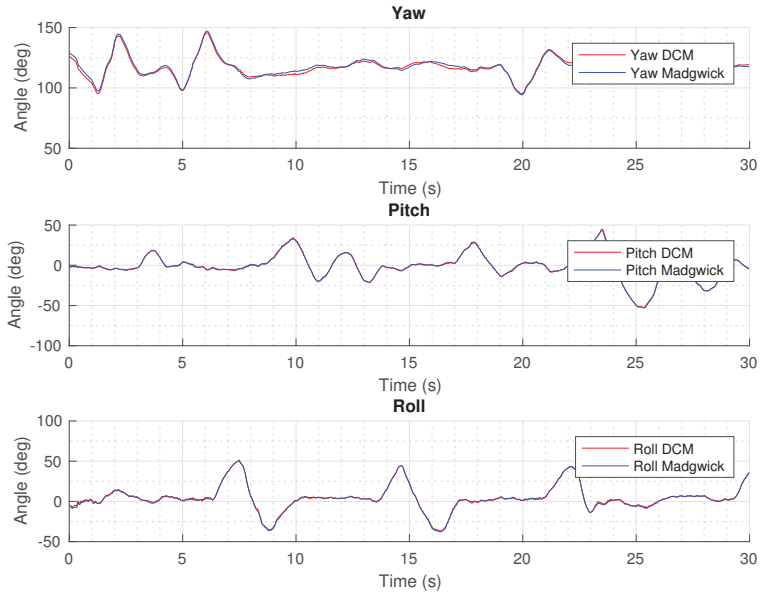

Figure 6.11: Slow movement comparative between DCM and Madgwick.

Table 6.3: RMS error comparing DCM and Madgwick.

	Yaw	Pitch	Roll
Static RMS	0.1516	0.0440	0.0955
Slow RMS	1.4888	0.5050	0.7327
Fast RMS	5.2659	3.3381	6.7603

the static scenario, but both algorithms perform similar.

In the last test, the sensor is moved very fast, so the accelerometer also measures external accelerations. The Euler angles calculated by each algorithm are shown in Fig. 6.13 and in Fig. 6.14 the difference between both algorithms is represented. In this case both algorithms perform quite different. The Madgwick algorithm seems to be much more accurate when external acceleration is present. This conclusion was taken by watching in real time the performance of both algorithms (see attached video *dcmmadgwick.mov*).

In Table. 6.3 RMS values of the error between both algorithms can be seen. It can be seen that in the scenarios where the device is moving, the error is bigger. It is also important to note that the difference in the yaw estimation is bigger than in the pitch and roll.

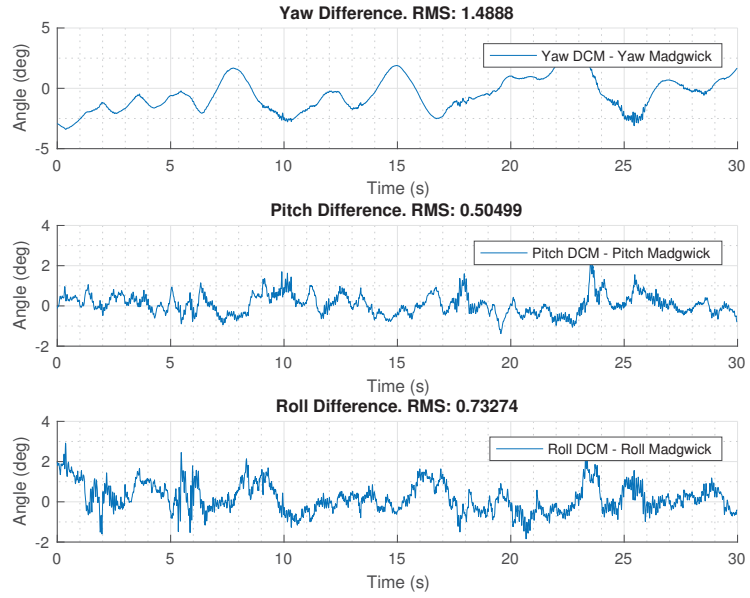

Figure 6.12: Slow movements error between DCM and Madgwick.

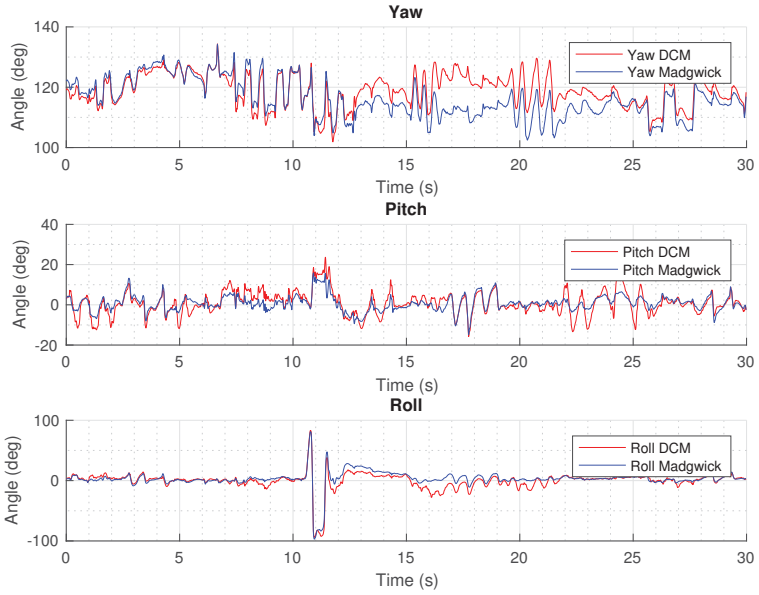

Figure 6.13: Fast movement comparative between DCM and Madgwick.

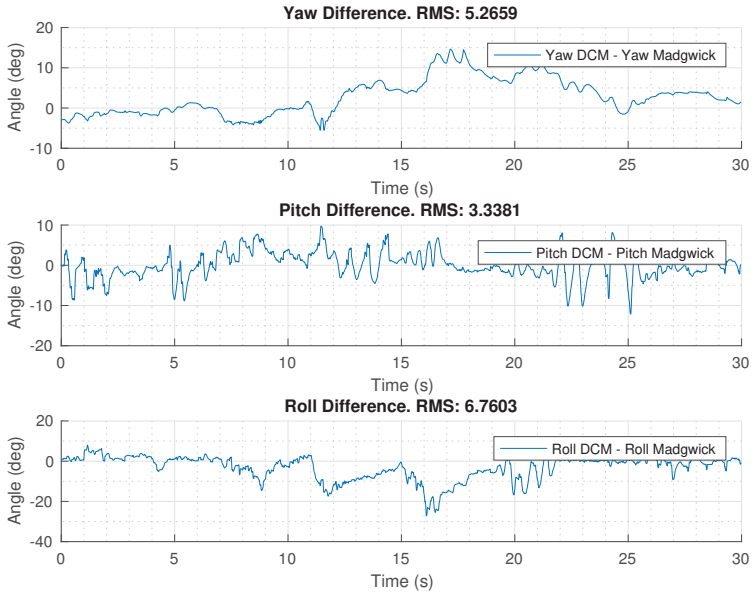

Figure 6.14: Fast movements error between DCM and Madgwick.

6.6 Execution times

Using a simple built in timer, the different tasks displayed in Fig. 5.7 were timed. In Table. 6.4 these times are shown in seconds (period) and Hertz (frequency). Some important aspects to note:

- Convert to the sensor units and apply the calibration are tasks that not require much time.
- Transmit the data over the serial port takes much less time that sending a UDP packet.
- Read data through I2C takes significant time.
- Madgwick algorithm is very optimized, and it takes less time that the DCM, in spite of being more complicated.

It is also very important to determine the upper bound of the device (Sensor + Arduino MKR1000). Assuming that the device has to read data from the sensors, apply the calibration, run an algorithm, and send the data, the sampling frequency will never be more than 160-200 Hz, depending on more specific information.

Table 6.4: Execution times of different operations in the Arduino MKR1000.

Operation	**Time (s)**	**Frequency (Hz)**
Read data	0.00304	330
Convert and calibrate	0.00014	7150
TX data over serial port	0.00088	1200
TX data over UDP	0.00361	280
Euler gyroscope + TX serial	0.00407	245
Euler complimentary + TX serial	0.00487	205
Euler DCM + TX serial	0.00667	150
Euler Madgwick + TX serial	0.00554	180
Euler DCM + TX UDP	0.00921	110
Euler Madgwick + TX UDP	0.00802	125

Chapter 7

Study Case

In previous chapters different sensor fusion techniques have been studied and compared, seeing their advantages and disadvantages. Due to the low execution time and performance, Madgwick algorithm has been chosen. In this chapter a practical scenario is presented, where a movement recognition algorithm is presented in Sec. 7.1 and evaluated in Sec. 7.2.

The proposed algorithm runs in the hardware setup explained in Ch. 5, using Madgwick algorithm for Euler angles calculations and sending all the data through a UDP port to Matlab. Once the data is received, the proposed algorithm is executed and different movements are recognized. One of the practical applications of this scenario, would be to attach the sensor to a bike, skate or snowboard to recognize the different movements that the user is doing.

In order to provide a more practical application, Plotly [31] libraries has been used. Plotly is a web based app that offers different API to interact with its content written in Matlab or Python. Real time data can be streamed to a Plotly plot from Matlab and that content can be seen in any web browser with a simple URL.

7.1 Introduction

The algorithm is fed with Euler angles and accelerations along each axis. Magnetometer and gyroscope data is also sent and available but not used. The algorithm parameters can be modified with different thresholds to be adapted to different scenarios (i.e. bike or skateboard). These are the movements that are recognized by default:

- $\theta = 360$: A 360 degrees turn around the θ axis, which corresponds to the roll.
- $\theta = -360$: Same as previous movement but in the other direction.

- $\psi = 180$: A 180 degrees turn around the ψ axis, which corresponds to the yaw.
- $\psi = -180$: Turn of 180 degrees in the other direction.
- $\psi = 360$: Turn of 360 around the ψ axis.
- $\psi = -360$: Same turn but in the other direction.

In Fig. 7.1, 7.2 the execution flow of the algorithm is displayed. Note that this section runs in Matlab and its input are the Euler angles and acceleration values: Important sections to take into account are:

- Unwrap: Note that Euler angles are inside the range -180/+180 for roll, -90/+90 for pitch and -180/+180 for yaw. So if two turns are performed i.e. in the roll axis the angle will go from -180 to 180 in the first turn, then it will jump to -180 again and go to 180. A similar approach as the one used in signal processing with the phase is applied in the algorithm. The idea is to avoid that jumps, but a continuous angle. So after two turns the output after the unwrap will be 720 degrees.
- Recognition: In the recognition phase there are two steps. First one is to detect when the user has finished a trick. In the case of a skate this is detected when an impulsive acceleration is detected in the z-axis. This means that the skate has "landed". After that, the movement has been performed is detected, if any.
- Yaw offset: When the skater starts going to a specific direction (yaw) that direction is set as zero. This is detected when a strong acceleration is detected in the y-axis. This threshold value can be tuned for different scenarios.

7.2 Tests and Results

Once the algorithm has been explained, now it is time to run it in real time. For the tests described in this section the device has been moved with the hand, without attaching it to a real skate/bike. In a real scenario some threshold would need to be adjusted to a better performance.

A first test is shown in Fig. 7.3. In this example the yaw is reset at $t = 3.6s$ and a 360 turn is done in the roll axis. It can be seen that after the trick, the unwrap is reset. In Fig. 7.4 raw data from the acceleroemter, magnetometer and gyroscope is shown. It can be clearly seen that the recognition is much more

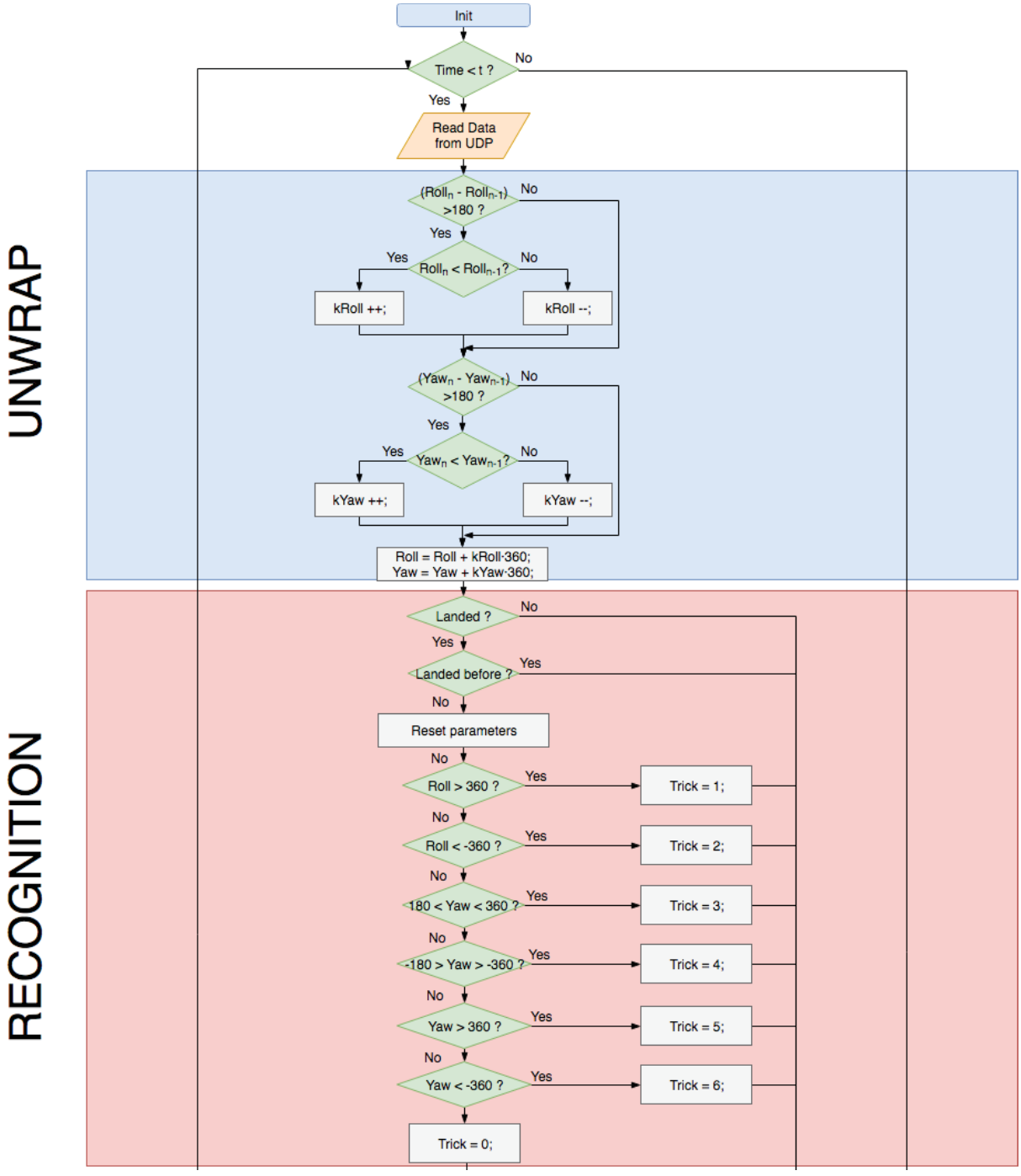

Figure 7.1: Flow control for movement recognition (1/2).

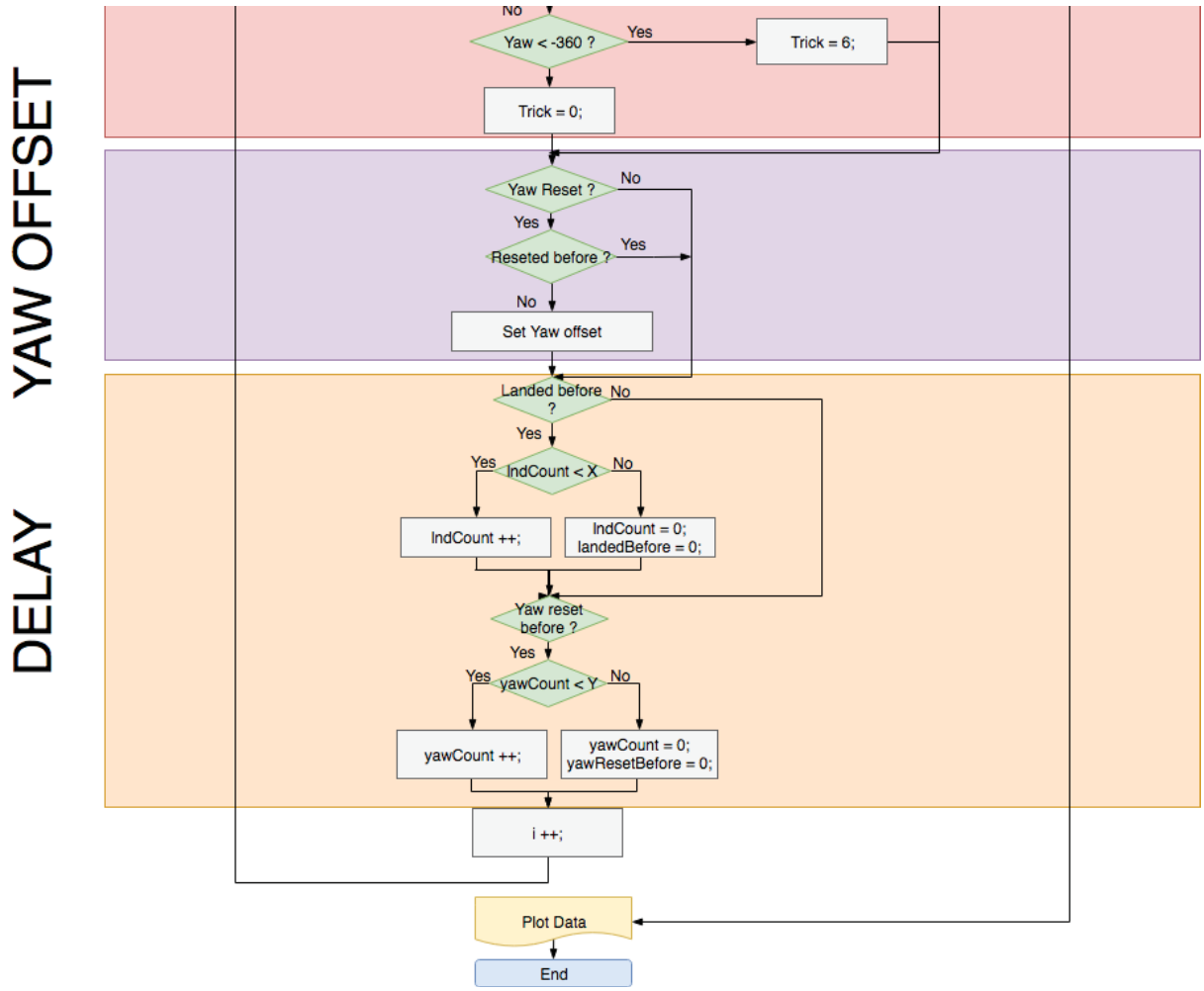

Figure 7.2: Flow control for movement recognition (2/2).

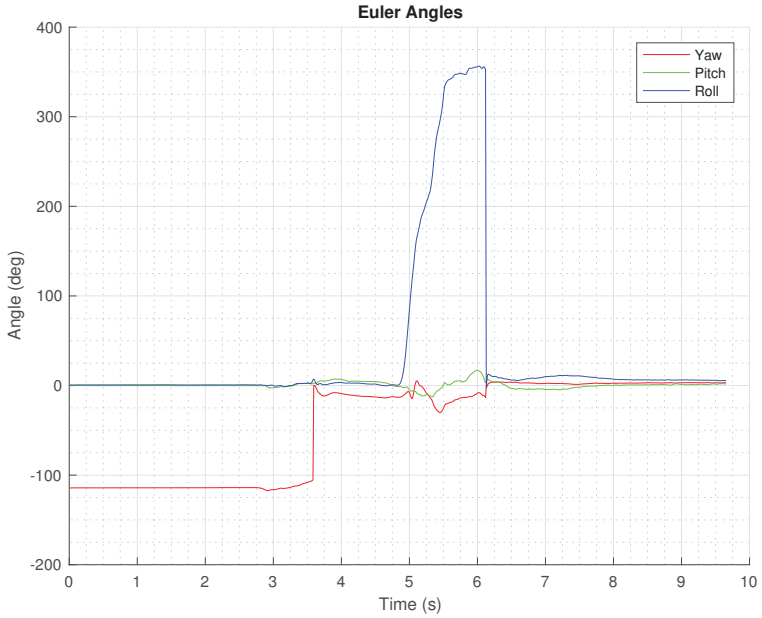

Figure 7.3: Time and Euler angles plot for a simple flip of 360 degrees.

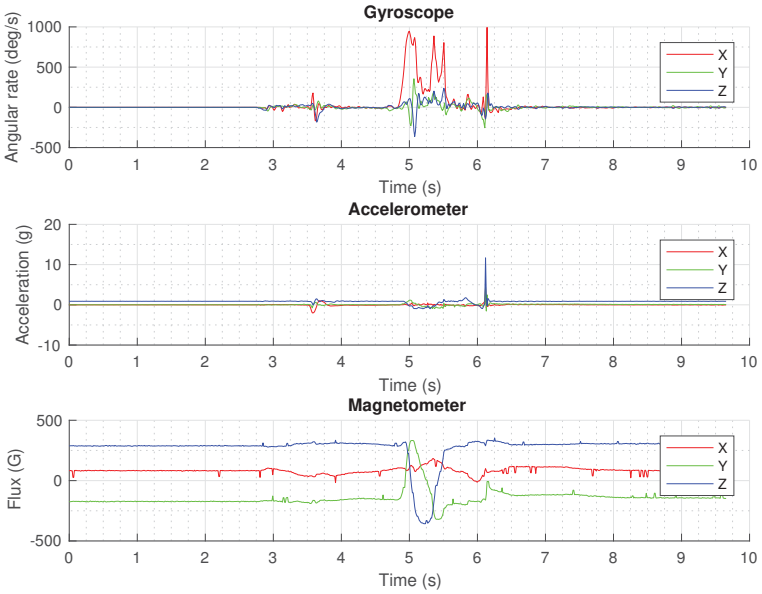

Figure 7.4: Sensors measurements for a simple flip of 360 degrees.

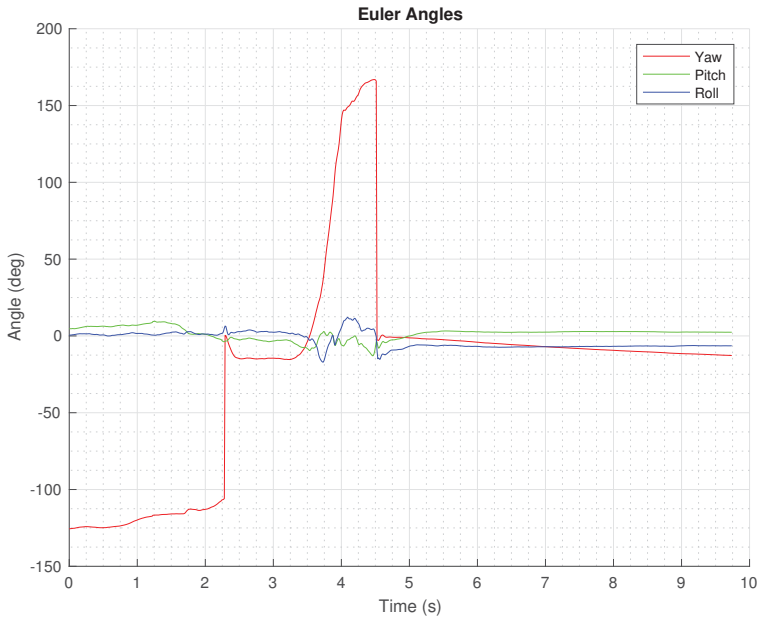

Figure 7.5: Time and Euler angles plot for a 180 degrees flip along the z (yaw axis).

intuitive using the processed Euler angles.

A similar test is shown in Fig. 7.5 where the movement that was performed was a rotation of 180 degrees in the yaw axis. In 7.6 the raw data of the sensors is also shown.

In Fig. 7.7 the advantages of performing the unwrap are seen. In this case the performed movement was several turns in the roll axis. The exact number of turns can be calculated by dividing by 360 the final angle.

Now that some basic examples of the performance of the algorithm were described, it is turn to run a test of around one minute, where different movements were done. In Fig. 7.8 a time vs trick plot is shown, so for example at time $t = 19.8$ the trick one was done, which corresponds to a turn of 360 in the roll axis θ. Processed Euler angles data is shown in Fig. 7.9.

Note that Fig. 7.8 and 7.9 are Matlab plots that are only available at the computer that is receiving and processing the data. Moreover, the program streams real time data to a Plotly plot. This allows anyone with the related address to watch in real time the tricks that has been done in any platform, just having a simple URL direction. In Fig. 7.10 an example is shown in a smart phone, where

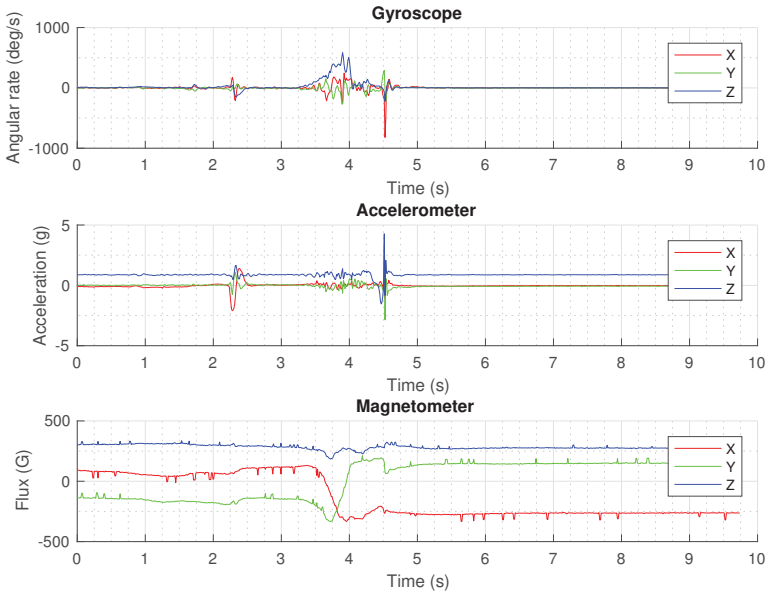

Figure 7.6: Sensor measurements for a 180 degrees flip along the z (yaw axis).

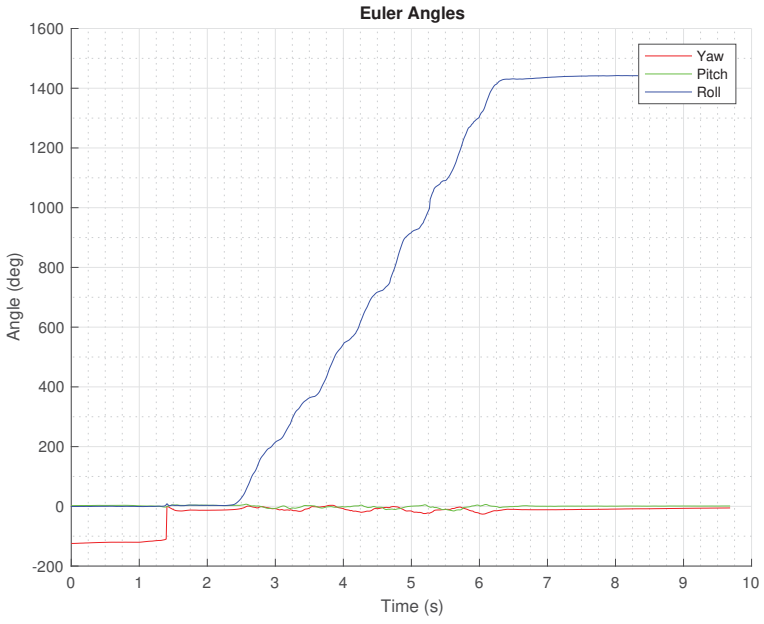

Figure 7.7: Time and Euler angles plot for several flips to illustrate the unwrap.

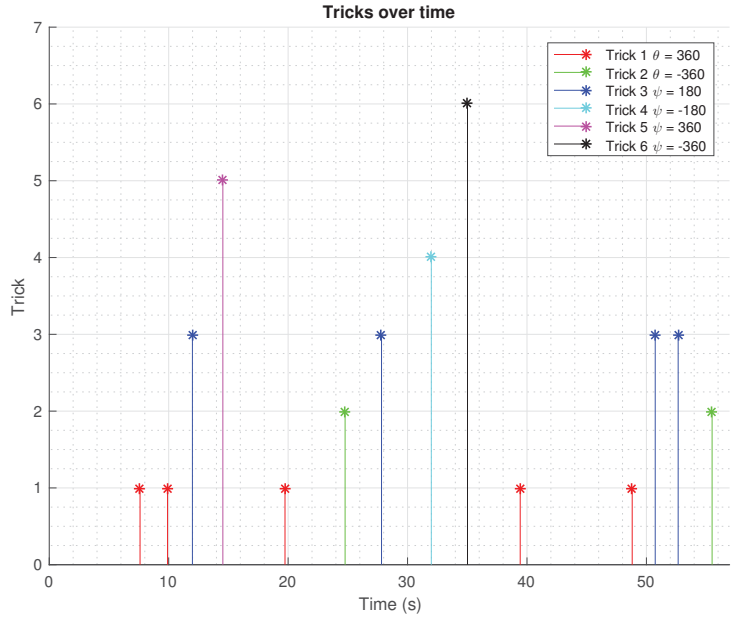

Figure 7.8: Tricks identified by an id over the time

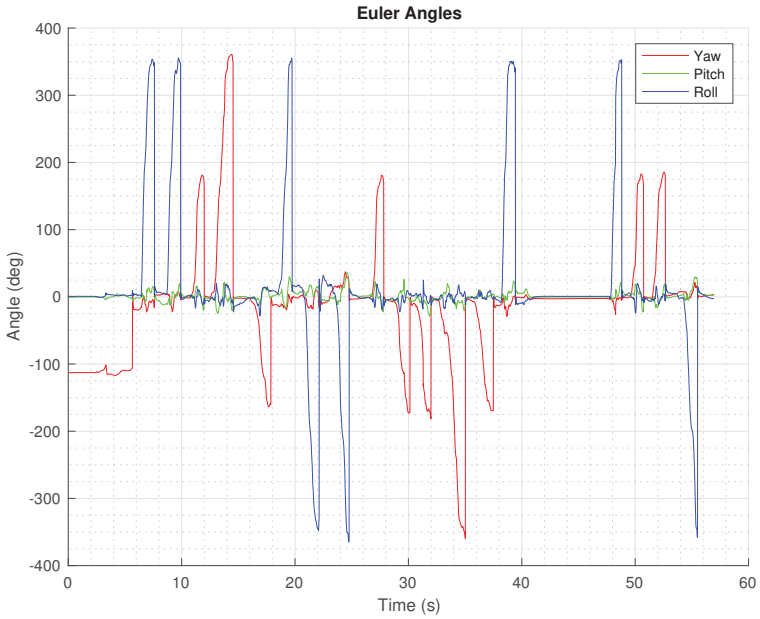

Figure 7.9: Euler angles over time when different tricks were made.

the x-axis is time and y-axis is the movement that was detected from 1 to n. Streaming the data to Plotly is very convenient because it can be visualized in many platforms concurrently and it is stored in the cloud, which means that can be accessed at any time.

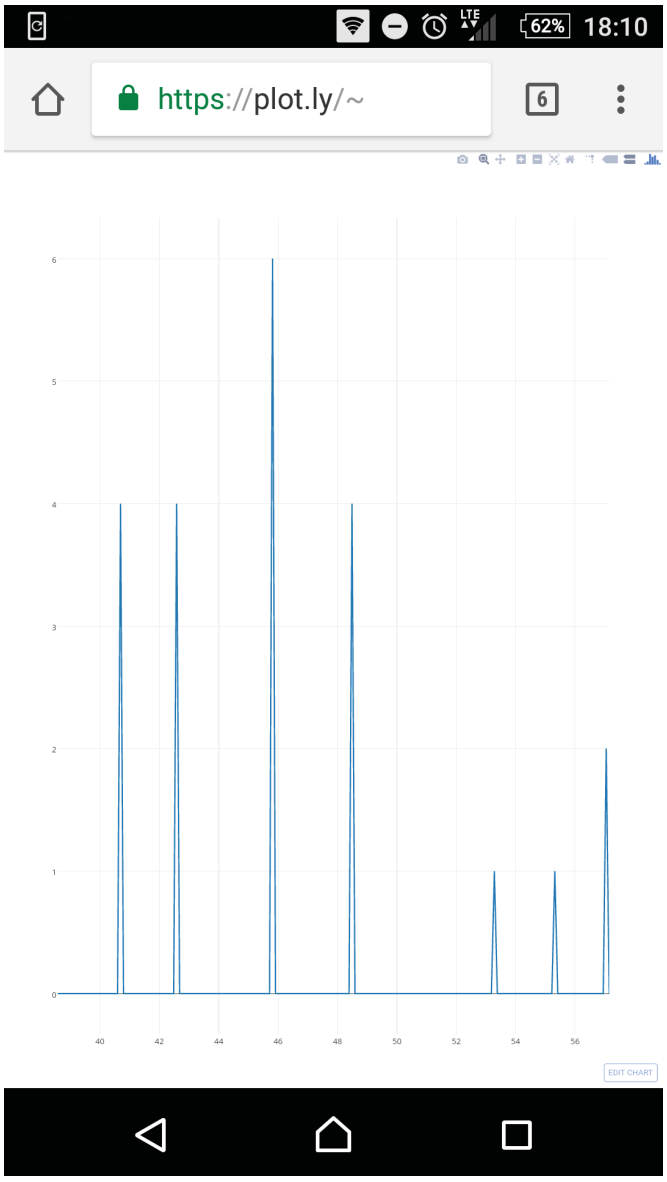

Figure 7.10: Plotly figure shown in a smart phone

Chapter 8

Summary and Conclusions

During this thesis, the existing orientation techniques have been described in Ch. 2, which are: quaternions, Euler angles and rotation matrices. Later on in Ch. 3 and Ch. 4 different methods to calculate the orientation were presented using low cost sensors such as accelerometer, magnetometer and gyroscope. In Ch. 5 the experimental setup for testing the algorithms was described and in Ch. 6 this setup was used to empirically test the performance of different orientation algorithms. After determining the most suitable algorithm for the study case, which was the Madgwick one due to its accuracy and computational cost, in Ch. 7 an application of the thesis is presented. In this application the device is attached to a moving object, like a skate board or bike, and through some data processing in Matlab, different movements are detected.

Through the different tests that were performed, these are the conclusions that were reached:

- The accelerometer by itself can't be used to determine the full orientation of a body. It may be useful to determine the pitch and roll in scenarios where there are not high accelerations, but when external forces are applied to the device, the measurements become unreliable. Yaw calculations are not possible due to the nature of gravity. When the pitch reaches angles close to 90 degrees, the pitch and roll estimation becomes unreliable. This can be deducted from Eq. 3.12 ,3.13 adding some noise.

- Gyroscopes can provide really good estimation of the orientation even when strong external accelerations are applied. However, low cost gyroscopes are quite noisy and after the integration used to estimate the angle from the angular speed, they tend to drift at high rates. So this make the gyroscopes only valid in short time periods (i.e. very few minutes).

- Complimentary filter algorithm is a very basic method of sensor fusion, where data from the accelerometer and gyroscope are fused to obtain the best of each one. With this method there won't be any drift for pitch and roll, and low external accelerations do not compromise the performance.

However, yaw estimation has not any fusion and is taken directly from the gyroscope, so the drift is still present.

- Calibration is extremely important to archive a good performance. Accelerometer and gyroscope calibration is quite easy to apply, with only a scale/offset and an offset respectively, and is not as critical as the magnetometer one. There are several ways of calibrating the magnetometer, but the one used for this thesis known as ellipsoid fit is fine and not very computational demanding.

- Accelerometer and gyroscope can be calibrated just once and they will not drift a lot with pressure or temperature. If they do a bit, it does not comprise the performance of the algorithm at all.

- The magnetometer has to be before every use calibrated because it is very influenced by its surroundings. Ferromagnetic materials or other external magnetic fields make the measurements totally unreliable. In the device used in the experimental setup, the WiFi module was placed very close to the magnetometer and this was taken into account for calibration.

- The DCM sensor fusion algorithm is able to estimate pitch roll and yaw without any drift using the accelerometer, magnetometer and gyroscope but is very influenced by external accelerations. It is also quite computational efficient so it can run in low cost micro processors such as Arduino.

- Madgwick algorithm was marked as the most suitable algorithm for the application of the thesis, where the orientation has to be estimated in high acceleration scenarios. The computational cost is slightly higher than the DCM so it can also run in low cost micro processors.

- Kalman Filter and its variations like the Extended Kalman Filter are very well known techniques that can perform sensor fusion, or estimate the drift of the gyroscope as an additional state. However its computational cost is quite high and can't be implemented in an Arduino.

- Low cost sensors can provide a good estimation of the orientation but note that the error can be of some degrees. Aerospace IMU or AHRS can cost from some hundreds to several thousand euros, so they can't be compared with a few euros one.

- As said before, accelerometers, magnetometers or gyroscopes by themselves can't provide a full estimation of orientation. Accelerometers can't estimate yaw, magnetometers can't estimate pitch, and a gyroscopes can estimate all angles but with drift. Sensor fusion techniques are very useful in scenarios like this, where each sensor provides some information that the other doesn't.

Chapter 9

Future Work

In this chapter some future work ideas and research lines are presented, both for hardware and software. Regarding to hardware, these are some ideas that can be improved for future versions of the device:

- The actual device used for running the experiments is based on an Arduino MKR1000 plus a nine degrees of freedom sensor. As seen in the budget, it is above 100 euros, so for a final product it is important to keep it cheap. Using other micro processors and sensors bought directly from the supplier, a price of less than 20 euros can be achieved.
- The device has three different ICs, one for each sensor (accelerometer, magnetometer, gyroscope). There are options in the market that gather all of them into a single IC. This will allow to reduce the size.
- Depending on the application, the battery size would need to be changed. In, in the application related in this thesis a battery life of 2 or 3 hours shall be enough, so smaller batteries can be used in order to reduce the weight.
- The device is using UDP sockets to send data to the receiver over a WiFi network. This means that an access point is needed, and this might be a drawback. A Bluetooth module can be another option instead of the WiFi one.

Related to software, sensor fusion algorithms and data processing:

- Madgwick and DCM fusion algorithms worked quite well for estimating the orientation of the sensor. However, the DCM implementation is not as optimized as the Madgwick one, so this can be a future work.
- Sensor fusion using Kalman filters in low cost micro processors is a complex task because of the lack of computational resources. Studying its performance in an Arduino can be an interesting project.
- In this project, a simple IoT application was tested. An interesting project would be to use a lot of sensors like the one presented in this project for a particular application.

References

[1] *Madgwick Sensor fusion on LSM9DS0*. [Online forum comment], 2014. Available at: Link.

[2] National Aeronautics and Space Administration. *Euler Angles, Quaternions, and Transformation Matrices*. Mission Planning and Analysis Division, 1977. Available at: Link.

[3] Arduino. *Arduino MKR1000 Specifications*. 2016. Available at: Link.

[4] Peter Bartz. *Razor AHRS Arduino Code*. [Computer program], 2013. Available at: Link.

[5] Peter Bartz. *DCM Implementation on Arduino in a AHRS Razor 9DOF Board*. [Computer program], 2016. Available at: Link.

[6] Peter Bartz. Razor 9dof attitude and heading reference system, 2016. Available at: Link.

[7] Mona Berciu. *Rotations and the Euler angles*. 2014. Available at: Link.

[8] Jose-Luis Blanco. *A tutorial on SE Transformation Parameterizations and On-Manifold Optimization*. Málaga University, 2013. Available at: Link.

[9] Majid Dadafshar. *Accelerometer And Gyroscopes Sensores: Operation, Sensing, And Applications*. 2014. Available at: Link.

[10] Pieter-Jan Van de Maele. *Reading a IMU Without Kalman: The Complementary Filter*. 2013. Available at: Link.

[11] Analog Devices. *ADXL345 Digital Accelerometer Datasheet*. 2009. Available at: Link.

[12] James Diebel. *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. 2006. Available at: Link.

[13] SparkFun Electronics. *SparkFun 9DOF Sensor Stick Specifications*. Available at: Link.

[14] Aleš Janota et *al.* *Improving the Precision and Speed of Euler Angles Computation from Low-Cost Rotation Sensor Data.* University of Žilina, 2015. Available at: Link.

[15] Howard Haber. *Three-Dimensional Rotation Matrices.* 2012. Available at: Link.

[16] John Safko Herbert Goldstein, Charles Poole. *Classical Mechanics.* Addison Wesley, 3 edition. Available at: Link.

[17] Honeywell. *3-Axis Digital Compass IC HMC5883L Datasheet.* 2011. Available at: Link.

[18] Noel H. Hughes. *Quaternion to Euler Angle Conversion for Arbitrary Rotation Sequence Using Geometric Methods.* Available at: Link.

[19] Invensense. *3-Axis MEMS Gyroscope ITG-3200 Datasheet.* 2010. Available at: Link.

[20] Keller Kindt. *Arduino Wire Library.* [Computer program], 2016. Available at: Link.

[21] Christopher Konvalin. *Compensating for Tilt, Hard-Iron, and Soft-Iron Effects.* 2009. Available at: Link.

[22] Jack B. Kuipers. *Quaternions and Rotation Sequences.* Calvin College, 2000. Available at: Link.

[23] Sebastian O. H. Madgwick. *An efficient orientation filter for inertial andinertial/magnetic sensor arrays.* 2010. Available at: Link.

[24] Sebastian O. H. Madgwick. *Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm.* IEEE, 2011. Available at: Link.

[25] Sebastian O. H. Madgwick. *Madgwick Algorithm Implementation in C.* [Computer program], 2011. Available at: Link.

[26] Sebastian O. H. Madgwick. *Madgwick Algorithm Implementation in Matlab.* [Computer program], 2011. Available at: Link.

[27] Sebastian O. H. Madgwick. *Open Source AHRS with x-IMU.* 2013. Available at: Link.

[28] Sandeep Mistry. *Wifi Library for the Arduino WiFi 101 Shield.* [Computer program], 2016. Available at: Link.

[29] Talat Ozyagcilar. *Calibrating an eCompass in the Presence of Hard- and Soft-Iron Interference.* Freescale Semiconductor, 2013. Available at: Link.

[30] Mark Pedley. *Tilt Sensing Using a Three-AxisAccelerometer*. Freescale Semiconductor, 2013. Available at: Link.

[31] Plotly. *Plotly Library for Matlab*. [Computer program], 2017. Available at: Link.

[32] William Premerlani and Paul Bizard. *Direction Cosine Matrix IMU: Theory*. 2009. Available at: Link.

[33] Arduino Projects. *Tilt Compensation in HMC5843 Magnetometer*. Available at: Link.

[34] Tobias Simon. *Madgwick IMU/AHRS and Fast Inverse Square Root*. [Online forum comment], 2012. Available at: Link.

[35] Starlino. *A Guide to Using IMU (Accelerometers and Gyroscope Devices) in Embedded Applications*. 2009. Available at: Link.

[36] Starlino. *DCM Tutorial: An Introduction To Orientation Kinematics*. 2011. Available at: Link.

[37] Erick Macias *et al.* *Nine-Axis Sensor Fusion Using the Direction Cosine Matrix Algorithm on the MSP430F5xx Family*. Texas Instruments, 2012. Available at: Link.

[38] Manon Kok *et al.* *Calibration of a Magnetometer in Combination With Inertial Sensors*. 2012. Available at: Link.

[39] Xia Yan *et al.* *Research on Random Drift Modeling and A Kalman Filter Based on The Differential Signal of MEMS Gyroscope*. IEEE, 2013. Available at: Link.

[40] Yong Shangguan *et al.* *Research on the Compensation in MEMS Gyroscope Random Drift Based on Time-Series Analysis and Kalman Filtering*. IEEE, 2015. Available at: Link.

[41] Federico Vanzati. *Arduino implementation of the MadgwickAHRS algorithm*. [Computer program], 2016. Available at: Link.

[42] Andrea Vitali. *Ellipsoid Orspherefitting for Sensor Calibration*. 2016. Available at: Link.

[43] Gordon Wetzstein. *Inertial Measurement Units II: Sensor Fusion and Quaternions*. Standford. Available at: Link.