POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Laboratory Report
Spring Chair

Advanced Measurement Systems for Control Application

Author:

Pietro Messina

Veronica Occhinero

Serena Palmieri

Student ID: 10663556 10676233 10660784

Academic Year: 2023/24

Contents

1 | Introduction

1.1. Project Objective

The objective of this project is to analyze the movements of a spring chair designed for children with disabilities. The application of this data acquisition is twofold:

- Evaluate the balance abilities of a patient, by analyzing the data
- Create an interactive game that aims to test and improve the coordination of the patient by setting challenges that involve controlling the chair's movements without exceeding certain inclination limits.

Figure 1.1: The Spring Chair

Proposed Solution

To achieve these objectives, the proposed solution involves determining the inclination of the chair and setting inclination limits. If these limits are exceeded, an LED lights up to indicate that the threshold has been surpassed. The selection of these thresholds is entirely up to the medical personnel in charge of the session with the patient and can be easily adjusted in any application presented in this project. For mere illustrative purposes, the thresholds have initially been set to 15 and 30 degrees.

1.2. Data Acquisition Systems

To implement this solution, two primary data acquisition instruments are systems. An MPU-6050 sensor, connected to an Arduino, provides precise real-time measurements of the chair's inclination. This real-time data is crucial for accurately assessing the immediate movements and positions of the chair. Additionally, a camera is used for computer vision. The camera captures video data of the chair's movements, which allows for the extraction of Euler angles: roll, pitch, and yaw. These angles help analyze the orientation of the chair over time, giving a comprehensive view of its motion.

The specific equipment utilized includes:

- an MPU6050 accelerometer/gyroscope
- an Arduino Uno
- a breadboard
- jumper wires and cables
- an IDS 3060CP Rev2 camera

1.3. System Assumptions

Several assumptions were made about the system to facilitate the analysis:

- The chair cannot twist on its own axis, hence the yaw angle can be disregarded.
- The entire seat-spring assembly is considered a rigid body: any compression along the vertical axis, just as any deformation in general, is considered negligible.

2 | Accelerometer/Gyroscope

This chapter outlines the methodology for measuring the inclination of the rocking chair using the MPU-6050 sensor. Initially, various attempted approaches will be described, detailing their respective flaws and the reasons for their rejection. This will provide a clear rationale for the chosen approach. Subsequently, a more in-depth analysis of its implementation will be provided.

2.1. System Setup

The breadboard with the MPU-6050 and the Arduino have been assembled in a box to improve stability. In order to capture the oscillations of the chair, they have been placed in the small cavity located under the base of the chair. The distance between the box and the platform on which the seat stands has been measured to be 40cm, with an uncertainty of 3cm. The precision of this measurement is not crucial, as this project focuses on angle measurements. This distance represents the radius of the sphere that contains all possible positions of the chair and will be helpful for more convenient visualization of the scene, as it will be shown.

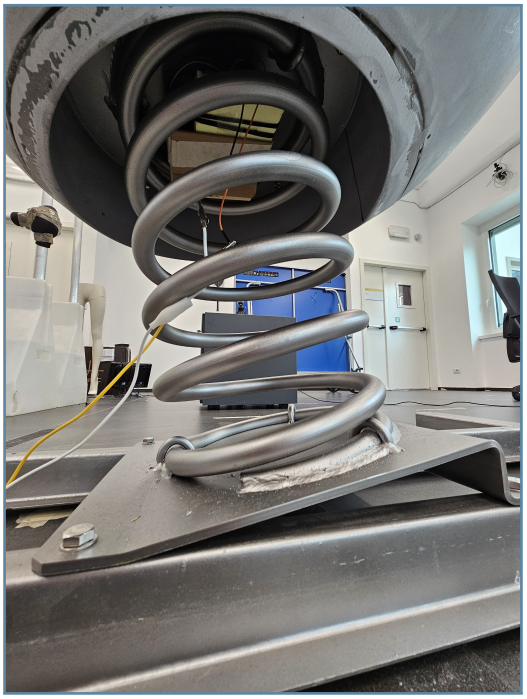

Figure 2.1: Accelerometer Setup

2.2. Methodology for orientation acquisition

Integration of angular velocities

The first approach contemplated consists in using the gyroscope to measure angular velocities around the three axes: $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$. Then the orientation (angle $\theta(t)$) is obtained integrating the angular velocity over time:

$$\theta(t) = \theta(0) + \int_0^t \omega(\tau)\, d\tau$$

However, gyroscopes have inherent noise due to limitations on sensor precision and to a bias, which is a constant offset in the angular velocity measurement. This can be modeled as:

$$\omega_{\text{measured}}(t) = \omega_{\text{true}}(t) + n(t) + b$$

where $n(t)$ represents the random noise and b represents the constant bias.
When integrating this noisy and biased signal, the noise $n(t)$ and the bias b accumulate over time. As a result, the angle estimate $\omega(t)$ tends to drift significantly, making the orientation estimate increasingly inaccurate over time.
To implement this in code, it is necessary to discretize the integration process.

This method allows for the real-time calculation of the orientation angles using the gyroscope data.

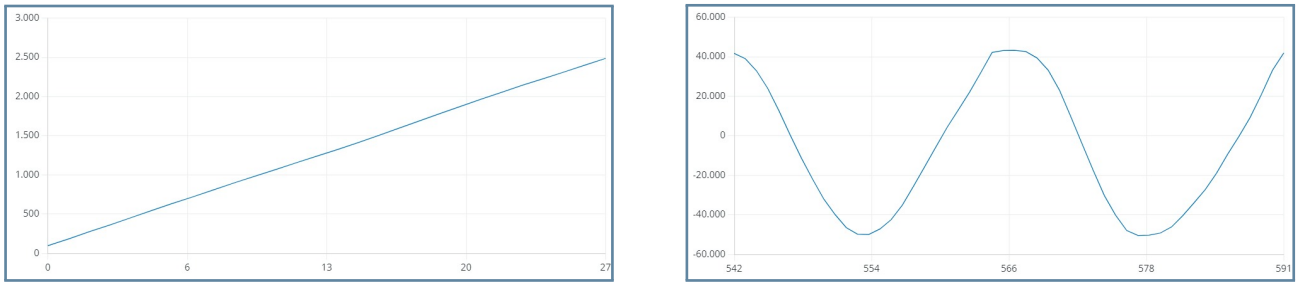

Figure 2.2: Pitch angle obtained from integration of angular velocities
X-axis: time (ms), Y-axis: angle (degree)

In figure 2.2, the left image shows the behavior of the angle at steady state, highlighting the divergence over time due to noise and bias.
In the right image, the behavior of the angle is shown when the accelerometer's position is varied from 0 to 90 degrees. The difference between the two peaks is exactly 90 degrees, indicating that the gyroscope captures the relative change in angle accurately. However, the absolute values of the angles are offset due to the divergence observed at steady state. Although the signals are relatively undisturbed, which is a positive aspect, the divergence makes the measurements unreliable over time. Therefore, it is necessary to consider alternative methods.

Gravitational Accelerations Decomposition

This approch consists in obtaining Euler angles, Roll and Pitch, from the accelerations a_x, a_y, and a_z measured by the accelerometer. The angles can be calculated from the accelerometer

data using the following equations:

$$Roll = \arctan\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right), \quad Pitch = \arctan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right),$$

It is not possible to calculate the Yaw angle from accelerometer data alone. As discussed in Section 1.3 of the introduction, this limitation is acceptable for the current application because the chair is assumed not able to twist on its own axis. Therefore, the Roll and Pitch angles are sufficient for the analysis.

Since the relationship between the accelerations and the Euler angles is nonlinear, any error in the measurement of the accelerations leads to disproportionately large errors in the angle estimations. What must be taken into consideration, though, is the high sensitivity of the accelerometer. Even in steady state condition and even without external movement, small mechanical vibrations, thermal noise, and sensor drift can cause the accelerometer readings to fluctuate, leading to unstable and inaccurate angle estimations.

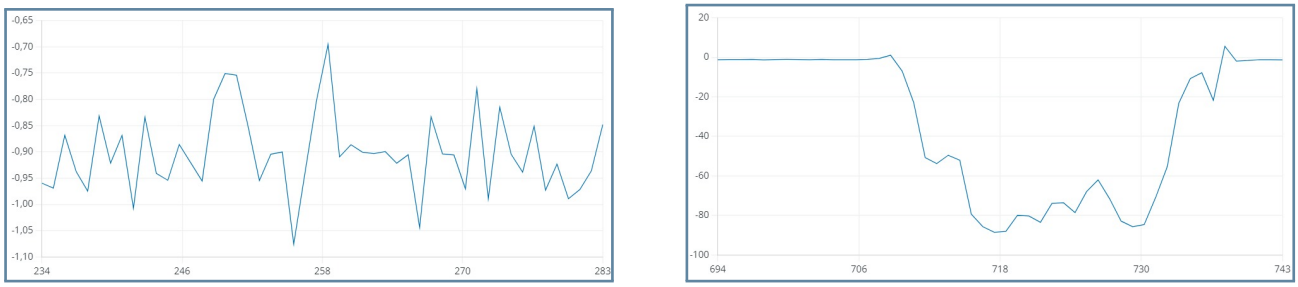

Figure 2.3: Pitch angle obtained from gravitational accelerations decomposition
X-axis: time (ms), Y-axis: angle (degree)

In figure 2.3, the left image shows the behavior of the angle at steady state. By obtaining the Euler angles from the accelerations, the divergence issue observed with the gyroscope method has been eliminated.
The right image illustrates the transient response when the accelerometer is rotated to 90 degrees and then back to zero. Unlike the gyroscope method, the absolute angles obtained from the accelerometer are accurate. However, these images also reveal a significant amount of noise, which can exceed 20 degrees. This level of noise is non-negligible for our scope.
The presence of such high noise levels requires the exploration of alternative methods for angle measurement. Filtering techniques were considered as potential solutions.

Kalman Filter

Despite the individual flaws in both gyroscope and accelerometer measurements, combining these imperfect measures using a Kalman filter results in more accurate orientation estimates. The Kalman filter operates by continuously updating its predictions of the system's state based on both the current measurements and the previous state estimates.
Initially, the angle prediction is made using the gyroscope data, which provides the rate of change in orientation. This prediction is then refined by incorporating the accelerometer data to correct any drift.

The recursive nature of the Kalman filter makes it highly effective in reducing noise and bias. However, it also requires continuous updates involving multiple operations at each time step. This computational demand can significantly slow down the response time.

For our type of experiment, where it is crucial to observe small and rapid oscillations, such as those occurring when a person tries to maintain balance on the chair, a high response time is essential. The computational intensity of the Kalman filter effectively slows down the system, making it less suitable for real-time applications where quick responses are crucial.

Complementary Filter

Similarly to the Kalman filter, the complementary filter is designed to fuse the data from an accelerometer and a gyroscope to accurately estimate the inclination or orientation of an object. The complementary filter structure consists of two filters:

- a Low-Pass Filter applied to the accelerometer data to smooth out high-frequency noise and provide a stable long-term orientation estimate.
- a High-Pass Filter applied to the gyroscope data to remove low-frequency drift, providing an accurate short-term orientation estimate.

The two filtered signals are then combined to produce a stable orientation estimate.

The mathematical representation is as follows:

$$\theta_{\text{angle}} = \alpha \cdot (\theta_{\text{angle}} + \omega_{\text{gyro}} \cdot dt) + (1 - \alpha) \cdot \theta_{\text{acc}}$$

Where:

- θ_{angle} is the estimated angle.
- α is the filter coefficient.
- ω_{gyro} is the angular velocity from the gyroscope.
- dt is the time interval.
- θ_{acc} is the angle derived from the accelerometer data.

In particular, the coefficient α is determined based on the desired balance between the accelerometer and gyroscope data:

$$\alpha = \frac{\tau}{\tau + dt}$$

Where τ is the filter's time constant.

With respect to the Kalman filter, the complementary filter is simpler, and therefore easier to implement, but above all its main advantage lies in its low computational overhead, which makes it the most suitable option for real-time applications.

In the following figures 2.4, the effectiveness of the complementary filter is demonstrated. The first image shows its behavior at steady state. It is evident that there is no divergence, and the error due to noise is negligible for our type of application, being on the order of 0.01 degrees. The second image shows the angle behavior when the accelerometer is moved from 0 degrees

to 90 degrees, repeatedly. It is immediately noticeable from the image that the issues observed with the other methods are no longer present.

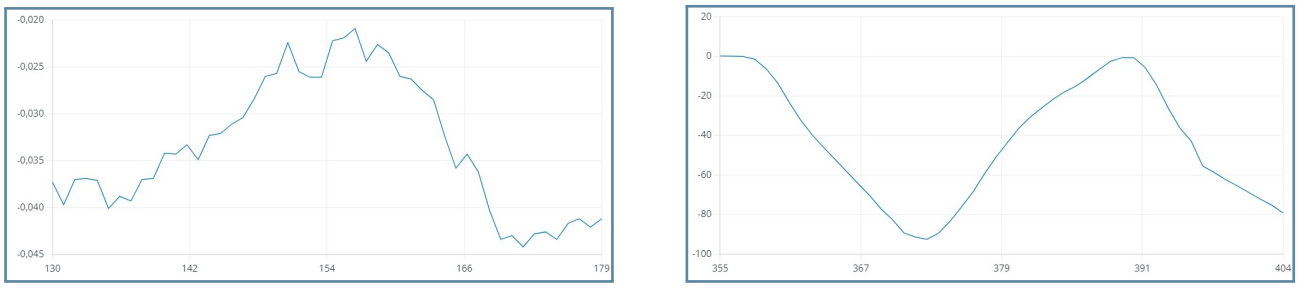

Figure 2.4: Pitch angle obtained from Complementary Filter
X-axis: time (ms), Y-axis: angle (degree)

2.3. Complementary Filter Code Implementation

Once the I2C communication with the MPU6050 sensor has been initialized and the MPU has been woken up from sleep mode through the power management register, the accelerometer and the gyroscope are configured with specific settings by writing to their respective configuration registers.

Then, the accelerometer and gyroscope are calibrated through the computation of their respective errors, which are calculated by averaging a sample of 500 readings.

The following extract of code outlines the steps to follow for the accelerometer. The proceedings for the gyroscope will be analogue.

```
    accRaw[0] = (Wire.read()<<8|Wire.read())/4096.0;
    accRaw[1] = (Wire.read()<<8|Wire.read())/4096.0;
    accRaw[2] = (Wire.read()<<8|Wire.read())/4096.0;

    accError[0] = accError[0] + ((atan((accRaw[1])/sqrt(pow((accRaw[0]),2) +
        pow((accRaw[2]),2)))*RAD2DEG));
    accError[1] = accError[1] + ((atan(-1*(accRaw[0])/sqrt(pow((accRaw[1])
       ,2) + pow((accRaw[2]),2)))*RAD2DEG));
    accError[2] = accError[2] + ((atan((accRaw[2])/sqrt(pow((accRaw[0]),2) +
        pow((accRaw[1]),2)))*RAD2DEG));
  }
  for(int i = 0; i<3; i++){
    accError[i] = accError[i]/500.0;
  }
```

In order to implement the complementary filter, the filter coefficient has to be chosen. It determines the balance between the gyroscope and accelerometer data, and it has been set 0.98. By doing so, it gives more weight to the gyroscope for its short-term accuracy while allowing the accelerometer to correct any long-term drift, providing a stable and accurate orientation estimate.

```
  roll = 0.98 * (roll + gyrAngle[0]) + 0.02*accAngle[0];
  pitch = 0.98 * (pitch + gyrAngle[1]) + 0.02*accAngle[1];
  yaw = 0.98 * (yaw + gyrAngle[2]) + 0.02*accAngle[2];
```

As previously mentioned, the complementary filter is computationally efficient. This is reflected in the time interval between each output, which is approximately 3 milliseconds, resulting in a frequency of approximately 333 Hz.

2.4. Live Visualization

In line with the practical applications of this project, a live visualization of the scene was created. This was made possible through the use of VPython, a Python library application that combines Python programming language with a 3D graphics module called Visual.
The simulation comprises the seat, the spring and the angular displacement thresholds represented by circles. The values of Roll and Pitch angles are clearly displayed as received from the Arduino code for a more comprehensive visualization.
For an even easier interpretation of the scene, when the seat exceeds each threshold, it changes colour.
In order to track perfectly the the chair, its position is iteratively adjourned at every sample coming from Arduino, using the Rodrigues' rotation formula:

```
newSensorPos = fixedPoint + rodriguesRotation(sensorOffset, vector(0, 0, 1),
    0)  # no yaw rotation
newSensorPos = fixedPoint + rodriguesRotation(newSensorPos - fixedPoint, x,
   pitch)
newSensorPos = fixedPoint + rodriguesRotation(newSensorPos - fixedPoint, y,
   roll)
```

Where:

```
fixedPoint = (0, 0, 0)
sensorOffset = (0, 0, 0.4)
```

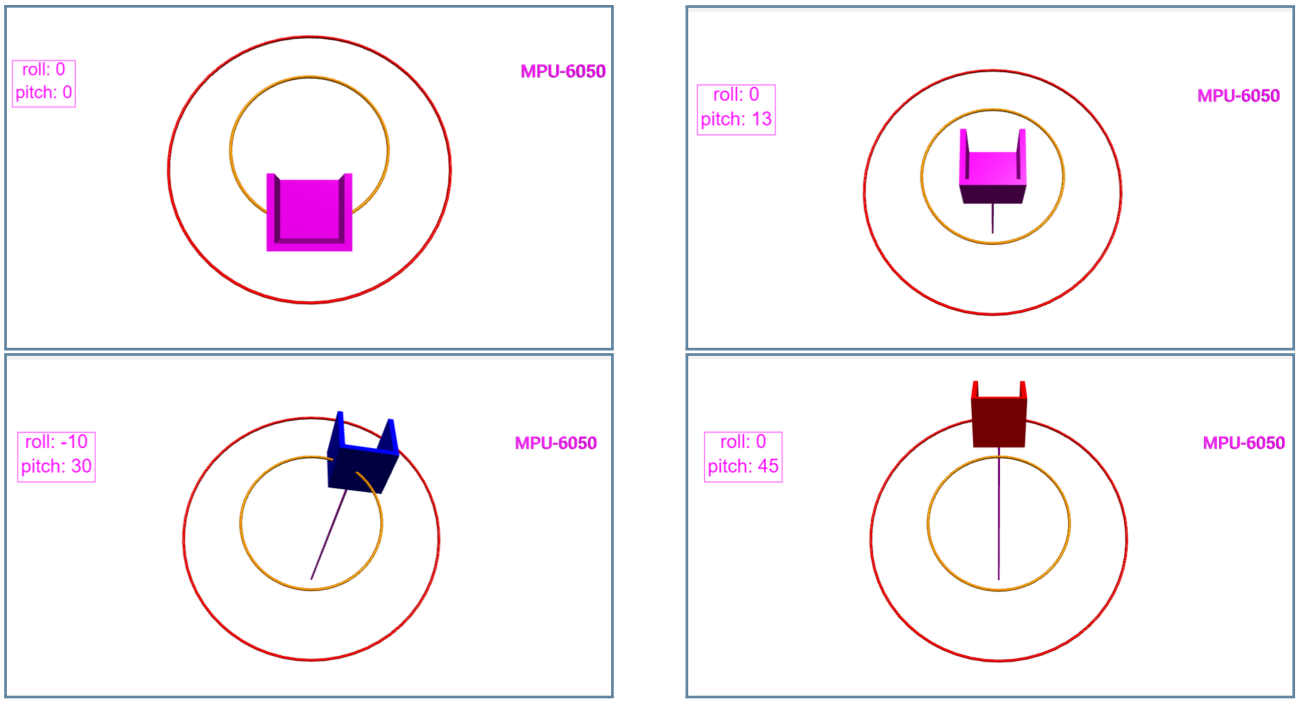

Figure 2.5: Screenshots from the 3D Live visualization using VPython

From the analysis of the images of Figure 2.5 it's evident that the seat is not centered in the circular thresholds when both Roll and Pitch angles are null. The reason for this will be object of a more in-depth discussion later in this report.

3 Computer Vision

In this chapter, the approach based on Computer Vision for measuring the inclination of the rocking chair will be described. The objective is to illustrate the data acquisition process and the extraction of Euler angles.

3.1. System Setup

Configuration of the Checkerboard and Camera Positioning

The setup for the computer vision analysis involves a checkerboard pattern attached to the backrest of the rocking chair. The checkerboard is centrally positioned to accurately calibrate and track the chair's movements. The distance between the middle of the marker and the ground is approximately 1 meter, measured with an uncertainty of 5 centimeters. This measurement doesn't require much precision, as it does not affect any calculation.

Figure 3.1: Checkerboard and Camera Setup for Computer Vision Analysis

As shown in Figure 3.1, the camera is placed at a strategic location directly facing the checkerboard. The camera's orientation and distance from the checkerboard are chosen to maximize the visibility of the pattern and minimize distortion. The distance between the camera and the marker is about 1 meter. The same considerations regarding the distance between the marker

and the ground apply in this case, too. The alignment ensures that the entire checkerboard is within the camera's field of view during any oscillation, allowing for precise extraction of the Euler angles.

The camera is mounted on a stable surface to prevent any movement that could affect the accuracy of the data. The height of this surface is 75 centimeters, measured again following the same considerations as before.

3.2. Calibration of the Checkerboard

The calibration of the checkerboards was performed using Python in the PyCharm IDE. A total of 16 images were captured and analyzed by the software.

The overall calibration process involves the following steps:

- Image Acquisition: 16 images of the checkerboard are captured from different angles and positions.
- Corner Detection: OpenCV functions are used to detect the corners in each image.
- Corner Refinement: cv2.cornerSubPix() is applied to refine the detected corners and improve their accuracy.
- 3D Coordinate Mapping: the 2D coordinates of the detected corners are mapped to the corresponding 3D coordinates previously generated.

First, the coordinates of the corners of the checkerboard in 3D space were created. This was done by initializing a list to store these coordinates and using a nested loop to generate the positions:

```
objp = []
for i in range(CHECKERBOARD[1]):
    for j in range(CHECKERBOARD[0]):
        objp.append([j*25.0, i*25.0, 0.0])
objp = np.array(objp, np.float32)
```

In this code, the coordinates are generated assuming that the checkerboard lies flat on the XY plane, with the Z coordinate being zero. The multiplication by 25.0 corresponds to the distance between adjacent corners, which is 25 mm.

After generating the 3D coordinates, the 'cv2.findChessboardCorners' function is used to find the corners on the images. This function detects the corners of the checkerboard pattern in the grayscale images captured earlier.
Once the corners are detected, their precision is improved using the 'cv2.cornerSubPix()' function provided by the OpenCV library.
Finally, cv.calibrateCamera() is used to perform the calibration. This function performs the actual calibration, returning several outputs, such as:

- ret: A boolean variable indicating if the calibration was successful.
- intrinsic: The intrinsic camera matrix, which contains the internal parameters of the camera such as the focal length and the optical center.

- r_{vecs}: The rotation vectors for each image used in calibration.
- t_{vecs}: The translation vectors for each image used in calibration.

This detailed calibration process ensures the precision and reliability of the data collected through the computer vision system, forming the basis for accurate measurement and analysis of the rocking chair's movements.

3.3. Data Acquisition with Computer Vision

Description of the Video Acquisition Method

The video acquisition process was conducted using the IDS peak Cockpit software. To achieve better quality, the video was set to black and white. The frame rate was set as high as possible, resulting in a value of 49 frames per second. This high frame rate ensures that the movements of the rocking chair are captured accurately and in detail.

Code Implementation

After obtaining the video, PyCharm was used once again, within the same script as the calibration, to extract the desired information. In the code, the first step was to open the video file and read frames in a loop as long as they were available. Then, after the checkerboard corners were detected, the cv.solvePnP() function was used to obtain the rotation and translation vectors. The cv.solvePnP() function takes several inputs:

- objp: The 3D coordinates of the checkerboard corners.
- corners2: The 2D coordinates of the detected checkerboard corners in the image plane.
- intrinsic: The intrinsic camera matrix obtained from the calibration process, which contains the internal parameters of the camera such as the focal length and the optical center.
- dist: The distortion coefficients that describe the lens distortion.

The following extract of the code will show how to proceed to obtain the Euler angles:

```
ret, r_vec, t_vec = cv.solvePnP(objp, corners2, intrinsic, dist)

r_mat, _ = cv.Rodrigues(r_vec)  # Convert the rotation vector (r_vec) into a
    rotation matrix (r_mat).
roll = np.arctan2(r_mat[2, 1], r_mat[2, 2])
pitch = np.arctan2(-r_mat[2, 0], np.sqrt(r_mat[2, 1]**2 + r_mat[2, 2]**2))
yaw = np.arctan2(r_mat[1, 0], r_mat[0, 0])

roll_history.append(roll)
pitch_history.append(pitch)
yaw_history.append(yaw)

roll_history_deg = np.degrees(roll_history)
pitch_history_deg = np.degrees(pitch_history)
yaw_history_deg = np.degrees(yaw_history)
```

This process ensures that each frame of the video is analyzed to determine the orientation of the checkerboard, allowing for accurate tracking of the rocking chair's movements over time.

4 | Comparison of Accelerometer Data and Computer Vision Data

4.1. Positioning of the Accelerometer with respect to the Checkerboard

For the purposes of this analysis, the accelerometer was placed in such a way that its pitch angle corresponds to the roll angle of the checkerboard. Consequently, the roll angle of the accelerometer aligns with the yaw angle of the checkerboard.

This specific positioning ensures that the data collected from the accelerometer can be accurately compared with the data derived from the checkerboard through computer vision.
From this point onward, references will be made only to the angles measured by the accelerometer.

4.2. Experiments and Results

In this section, the various experiments conducted to compare the accelerometer data with the computer vision data are discussed.

The data acquired with the Arduino are used for real-time feedback, helping to monitor the balance and movements of the rocking chair thanks to the implementation of an LED.
The data acquired from computer vision are used to validate the data collected by the Arduino. This comparison ensures that the Arduino-based measurements are accurate and that the feedback mechanism operates correctly.

Experiment 1: Study of Balance

The first experiment focused on studying the balance of the chair. This step was essential as it helps determine the inclination around which the chair maintains equilibrium. The equilibrium position does not coincide with zero (zero being the chair's position when unoccupied), but depends on the backrest's inclination, which is influenced by the weight of the person seated. In the experiments shown, the balance point was calibrated for a person weighing 55 kg.
The average values during the oscillation period were calculated to determine the equilibrium position, resulting in an equilibrium angle of approximately 13 degrees for the Pitch (Figure 4.1, from 5 to 15 seconds) and 0 degrees for the Roll (Figure 4.2, from 5 to 15 seconds).
This process is easily repeatable using a MATLAB code that calculates the average angular position maintained during a given period. By doing so, the equilibrium position can be adjusted in all the data analysis codes. It should also be noted that the precision on the equilibrium

position simply affects how accurately the thresholds are centered relatively to the equilibrium position itself.

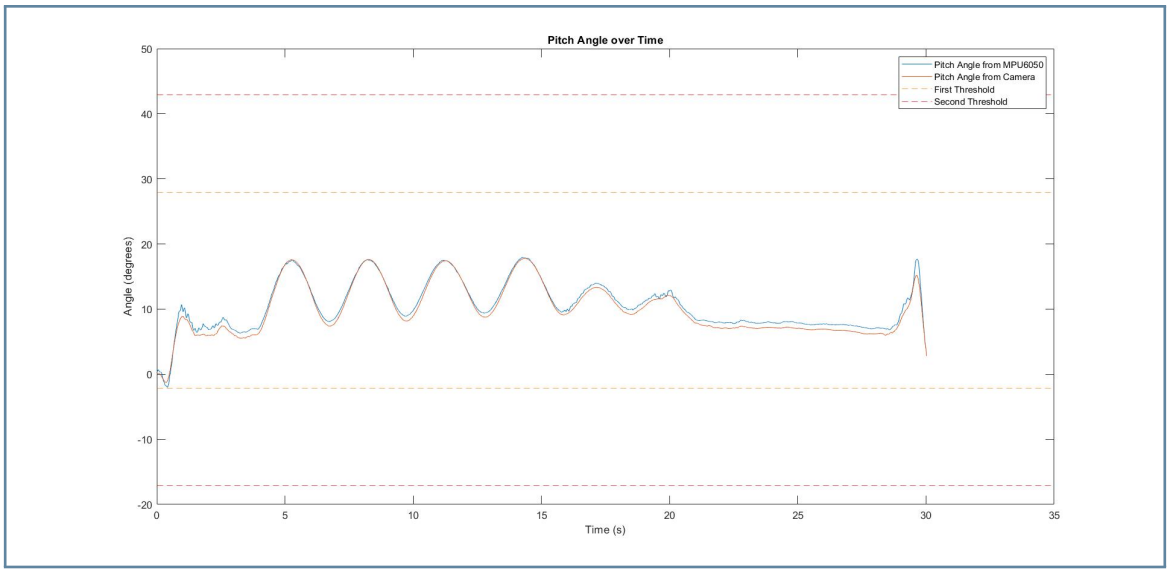

Figure 4.1: Pitch angle, experiment 1

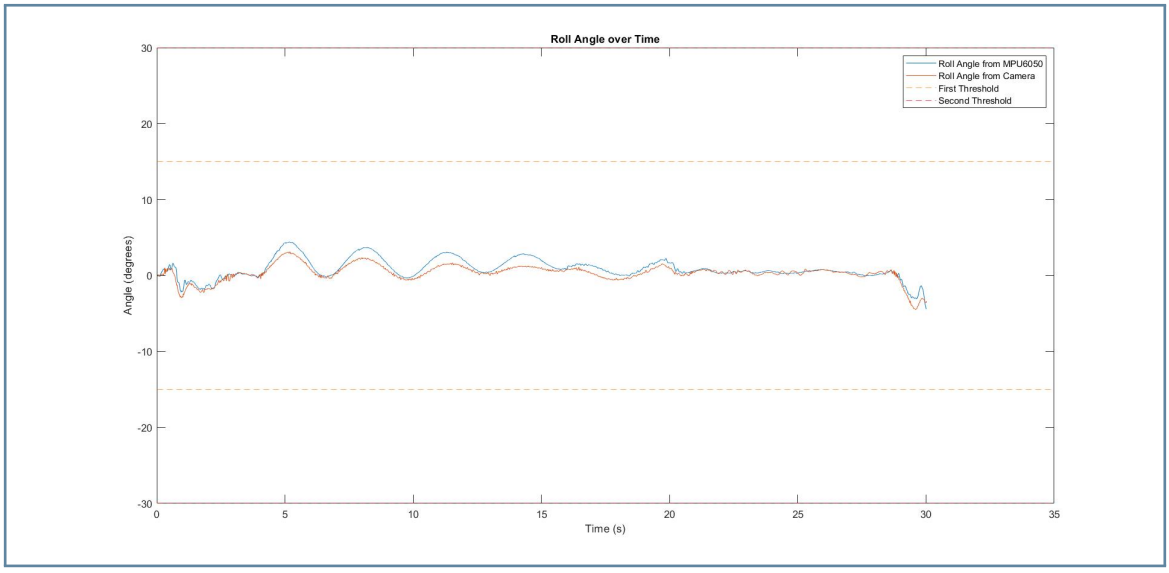

Figure 4.2: Roll angle, experiment 1

The results from this experiment show the validity of the data collected by the Arduino. The graphs 4.1 and 4.2 illustrate the pitch and roll angles over time, comparing data from the MPU6050 sensor and the camera.

The following metrics highlight the accuracy of the measurements:

- NRMSE between roll angles measures: 8.4%
- NRMSE between pitch angles measures: 3.6%

- Maximum error on roll measure: 1.9 degrees
- Maximum error on pitch measure: 2.8 degrees

where NRMSE stands Normalized Root Mean Square Error, and is computed as:

$$NRMSE = \sqrt{\frac{\frac{1}{n}\Sigma_{i=1}^{n}(y_{real} - y_{sim})^2}{y_{max} - y_{min}}} \tag{4.1}$$

Experiment 2: Simple Movements

In this experiment, simple movements were performed to verify the precision of the roll and pitch angles. The chair was tilted once to the right, affecting mostly the roll angle (Figure 4.3), and once backward, affecting mostly the pitch angle (Figure 4.4). As seen in the graphs, the pitch angle shows an oscillation even though the experiment consisted of a single backward movement. This occurs because once the person stands up, the equilibrium position shifts from 13 degrees to zero degrees.

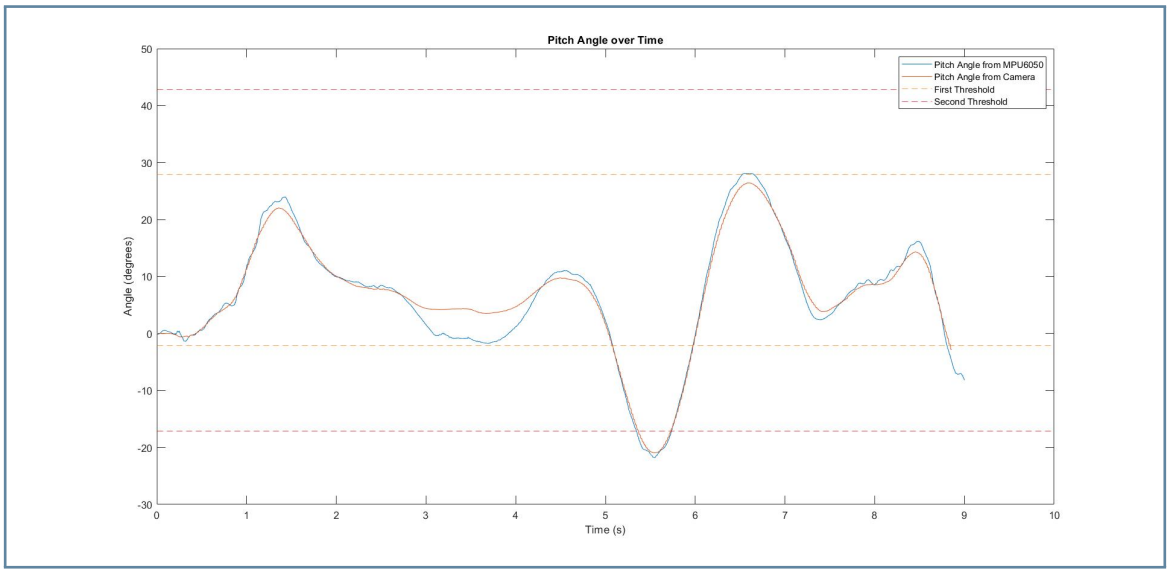

Figure 4.3: Pitch angle, experiment 2

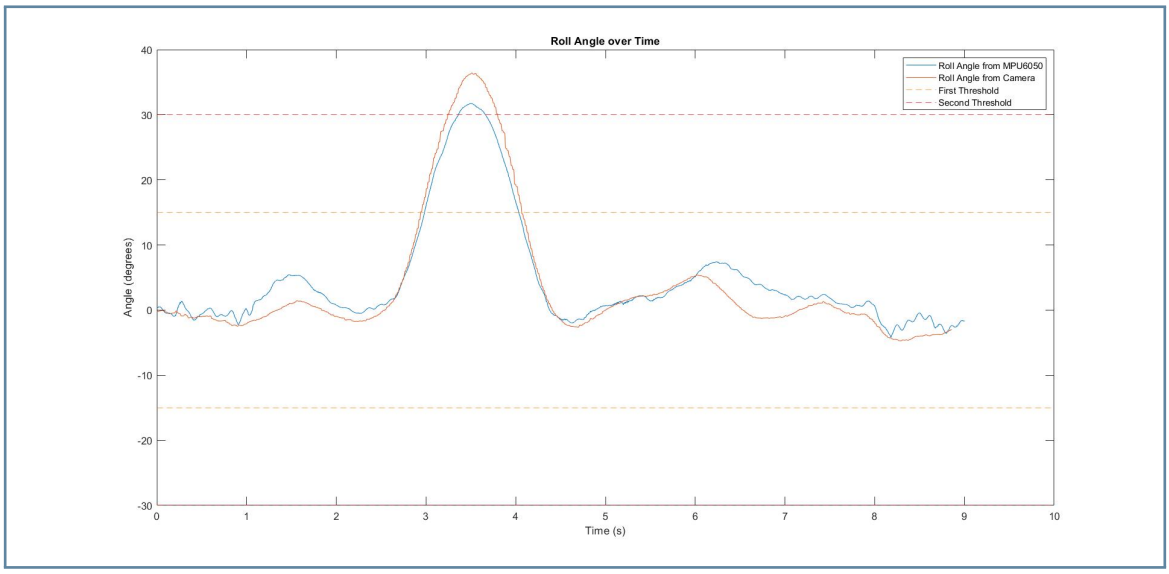

Figure 4.4: Roll angle, experiment 2

The following metrics highlight the accuracy of the measurements:

- NRMSE between roll angle measures: 6.8%
- NRMSE between pitch angle measures: 3.8%
- Maximum error on roll measure: 5.6 degrees
- Maximum error on pitch measure: 5.3 degrees

These results confirm the high level of agreement between the two measurement methods.

Experiment 3: Random Movements

In the final experiment, the person seated was asked to perform random movements. Figures 4.5 and 4.6 show the results and the differences in measurements acquired with the two different methods.

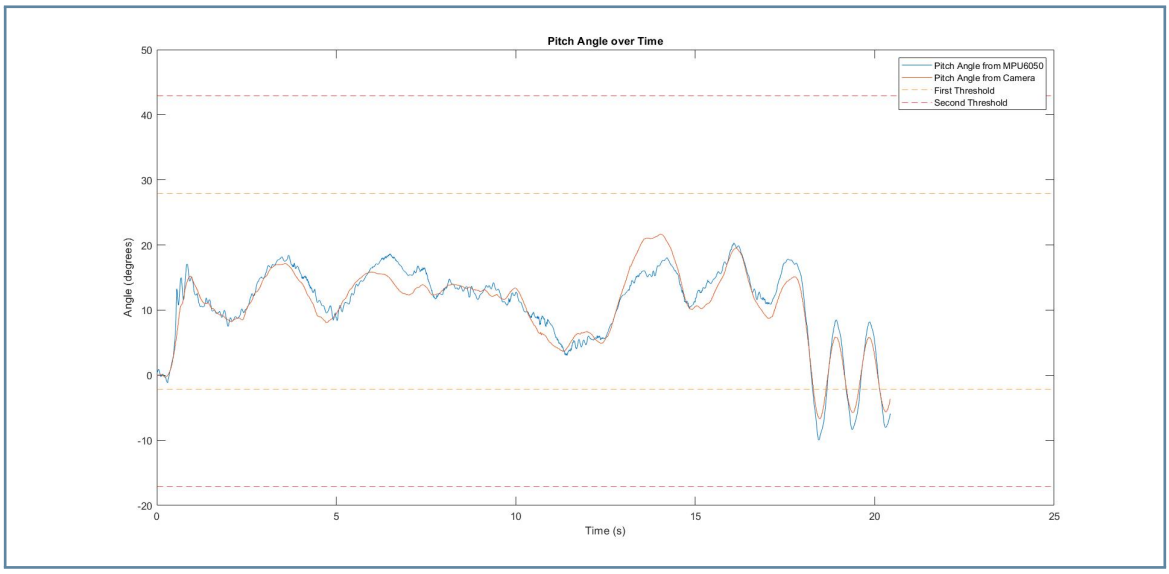

Figure 4.5: Pitch angle, experiment 3

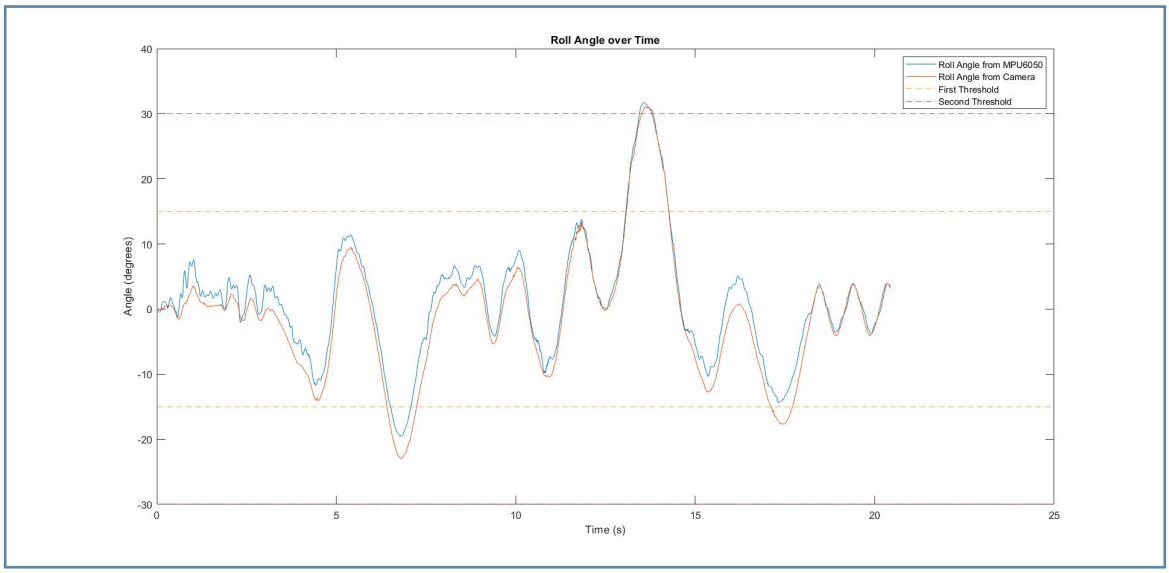

Figure 4.6: Roll angle, experiment 3

The following metrics highlight the accuracy of the measurements:

- NRMSE between roll angle measures: 4.7%
- NRMSE between pitch angle measures: 6.3%
- Maximum error on roll measure: 5.8 degrees
- Maximum error on pitch measure: 7.2 degrees

4.3. Observations

From the experiments conducted, it is evident that the accelerometer data is more susceptible to noise compared to the data from the computer vision system.
For the purposes of this study, this noise is negligible and does not significantly impact the overall accuracy of the measurements. The accelerometer's real-time feedback is reliable and sufficient for monitoring the chair's movements and balance.

On the other hand, the computer vision system, while less affected by noise, has its own limitations. It is less reliable due to the potential loss of frames during data acquisition, which can lead to inaccuracies. Additionally, optimal lighting conditions are essential for obtaining accurate and consistent data. Without proper illumination, the accuracy of the computer vision measurements can be compromised.

The integration of both systems can offer a comprehensive solution to provide accurate and reliable measurements of the rocking chair's movements and balance.

5 | Post Processing

5.1. 3D Visualization

Using the measurements of Roll and Pitch angles and the measure of the radius of the rotation, it's possible to create an interactive 3D plot of the positions assumed by the chair.

```
for i = 1:length(roll)
    R = angle2dcm(deg2rad(roll(i)), deg2rad(pitch(i)), 0, 'XYZ'); % Rotation
        matrix
    position = r * [0; 0; spring_length];

    px(i) = position(1);
    py(i) = position(2);
    pz(i) = position(3);
end

figure;
Traj = plot3(px, py, pz, 'DisplayName', 'Trajectory');
```

The thresholds are represented by portions of the spherical surface centered in (0,0,0) and having radius 0.4 meters. The circular borders of these spherical caps correspond to 15 and 30 degrees dislocations with respect to the equilibrium position. For these reasons, just as for the case of the circles in the VPython view, they appear tilted and are not parallel to the ground.

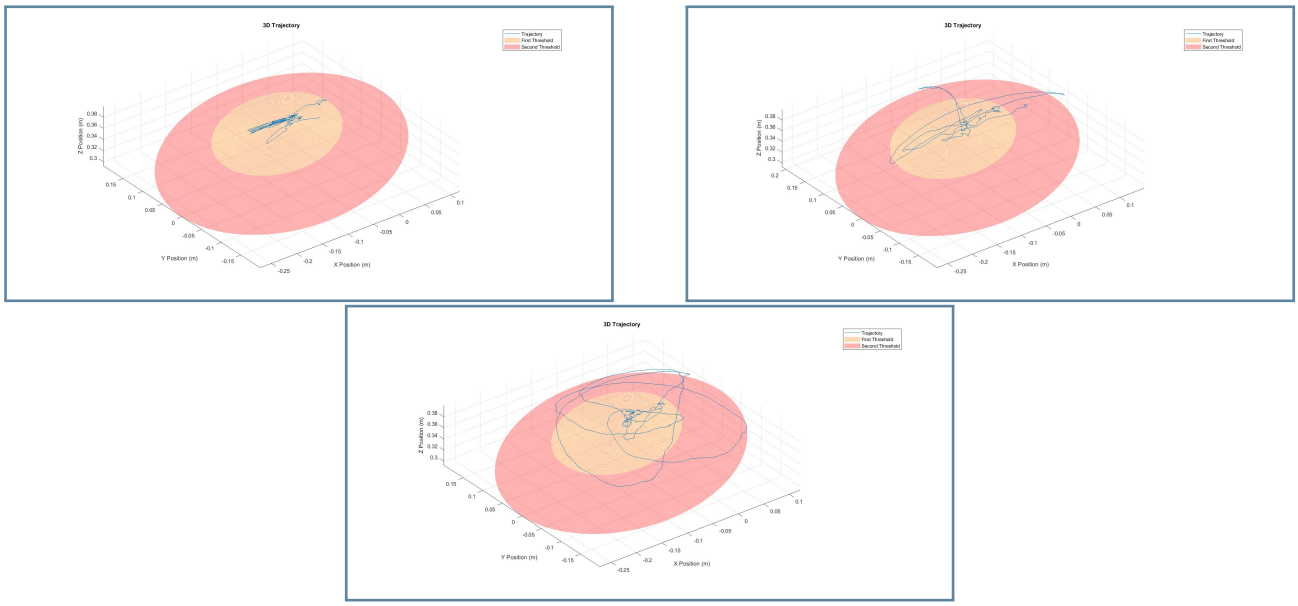

Figure 5.1: Interactive plots of the seat position in space: experiments 1, 2 and 3

The interactive nature of these plots allows for a full personalization of the viewing perspective, including the ability to switch to a 2D view from above.

5.2. Data Analysis

From the gathered data, it is possible to perform various analyses, the choice of which is left to the professionals handling the sessions. For illustrative purposes, some of the possible parameters that can be studied and their applications will be shown.

```
Threshold = 15.0;
timeCount = 0.0;
timeBalanced = [];
totTime = 0.0;
finishedOutsideThreshold = 0;

for i = 1 : length(roll)
    if abs(roll(i)-equilibrium) < Threshold && abs(pitch(i)-equilibrium) <
       Threshold
        timeCount = timeCount + timeStep(i);
    end
    if (abs(roll(i)-equilibrium) >= Threshold || abs(pitch(i)-equilibrium)
       >= Threshold) && timeCount ~= 0.0
        timeBalanced = [timeBalanced timeCount];
        timeCount = 0;
    end
    totTime = totTime + timeStep(i);

if timeCount ~= 0
    timeBalanced = [timeBalanced timeCount];
else
    finishedOutsideThreshold = 1;
end
```

This snipped of code successfully calculates the duration of each time interval during which the position is maintained within the threshold. As long as this condition holds, the time counter is incremented. When the condition is eventually violated, the last value of the counter is stored in an array and then reset, ready to be incremented again when the position returns within the threshold.
Some possible applications of these parameters are shown below, after implementing the second threshold in the previous code. These are taken directly from the MATLAB command window and refer to experiments 1, 2 and 3 respectively.

```
The session lasted 30.0 seconds
The position has been maintained inside the 15 threshold for as long as 30.0
    consecutive seconds
The position has been maintained inside the 30 threshold for as long as 30.0
    consecutive seconds
The position has been maintained inside the 15 threshold for a total of 30.0
    seconds, corresponding to 100% of the whole session
The position has been maintained inside the 30 threshold for a total of 30.0
    seconds, corresponding to 100% of the whole session
The 15 threshold has been exceeded 0 times
The 30 threshold has been exceeded 0 times
```

```
The session lasted 9.0 seconds
The position has been maintained inside the 15 threshold for as long as 3.0
   consecutive seconds
The position has been maintained inside the 30 threshold for as long as 3.4
   consecutive seconds
The position has been maintained inside the 15 threshold for a total of 6.7
   seconds, corresponding to 75% of the whole session
The position has been maintained inside the 30 threshold for a total of 8.3
   seconds, corresponding to 92% of the whole session
The 15 threshold has been exceeded 4 times
The 30 threshold has been exceeded 2 times
```

```
The session lasted 13.0 seconds
The position has been maintained inside the 15 threshold for as long as 2.7
   consecutive seconds
The position has been maintained inside the 30 threshold for as long as 7.1
   consecutive seconds
The position has been maintained inside the 15 threshold for a total of 7.9
   seconds, corresponding to 61% of the whole session
The position has been maintained inside the 30 threshold for a total of 12.4
    seconds, corresponding to 96% of the whole session
The 15 threshold has been exceeded 5 times
The 30 threshold has been exceeded 3 times
```

6 | Bibliography

[1] Carbon Aeronautics. 2022. Carbon Aeronautics Quadcopter Build and Programming Manual.

[2] Hibit.dev. Complementary Filter and Relative Orientation with MPU6050. https://www.hibit.dev/posts/92/complementary-filter-and-relative-orientation-with-mpu6050

[3] OpenCV. Open Source Computer Vision Library. https://opencv.org/

[4] Python Software Foundation, Python Language Reference, version 3.x, 2023. https://www.python.org/

[5] David Scherer, Bruce Sherwood, Ruth Chabay, et al., VPython: 3D Programming for Ordinary Mortals, 2023. ttp://vpython.org/

[6] MathWorks. MathWorks - Makers of MATLAB and Simulink. https://it.mathworks.com/?s_tid=gn_logo