به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

دانشکده مهندسی مکانیک

رباتیک و مکاترونیک

مینی پروژه شماره 1

نام و نام خانوادگی	نورا زارعی – عرفان عسگری
شماره دانشجویی	810199460 – 810199433
تاریخ ارسال گزارش	1403/01/13

Question report list:

Problem 1 – Angle Recording using MPU-6050

In this problem, we want to find Euler angles and quaternion values using an MPU-6050 sensor, which is a sensor with 6 degree of freedom (3-axis gyroscope and 3-axis accelerometer).

1. At first, we should implement the circuit using MPU-6050 and Arduino-UNO. As illustrated in Figure 1, we need to connect the VCC pin of the sensor to the 5V pin of the Arduino, GND to GND, SCL to A5, and SDA to A4 using jumper wires.

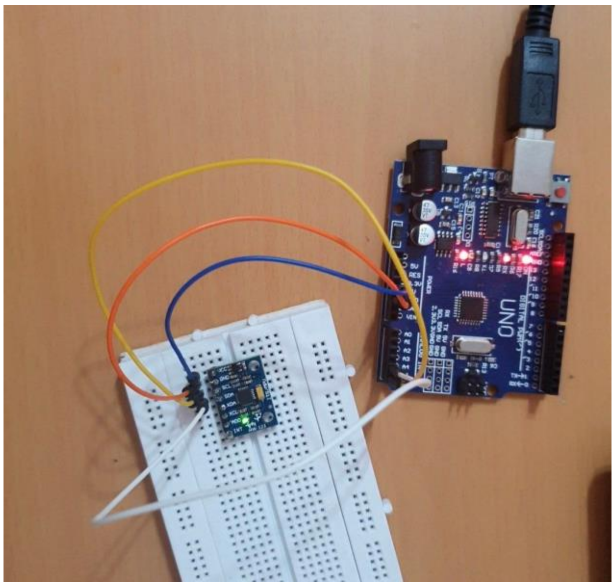
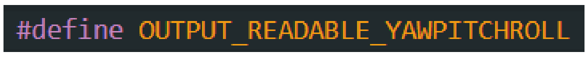


Figure 1 Implemented circuit

2. Now, we connect the Arduino to the laptop and upload the provided code onto it. Then, we run the code in two modes: one to display Euler angles and the other to display quaternion values. Finally, we view the output in the Serial-Plot application.

Euler angles are displayed first by uncommenting the part that shows in Figure 2.



```
#define OUTPUT_READABLE_YAWPITCHROLL
```

Figure 2 Uncomment this line to show Euler angles

As depicted in Figure 3, angles are accurately displayed with sensor movement. This phenomenon is also observable in the video titled “Problem1_EulerAngles”.

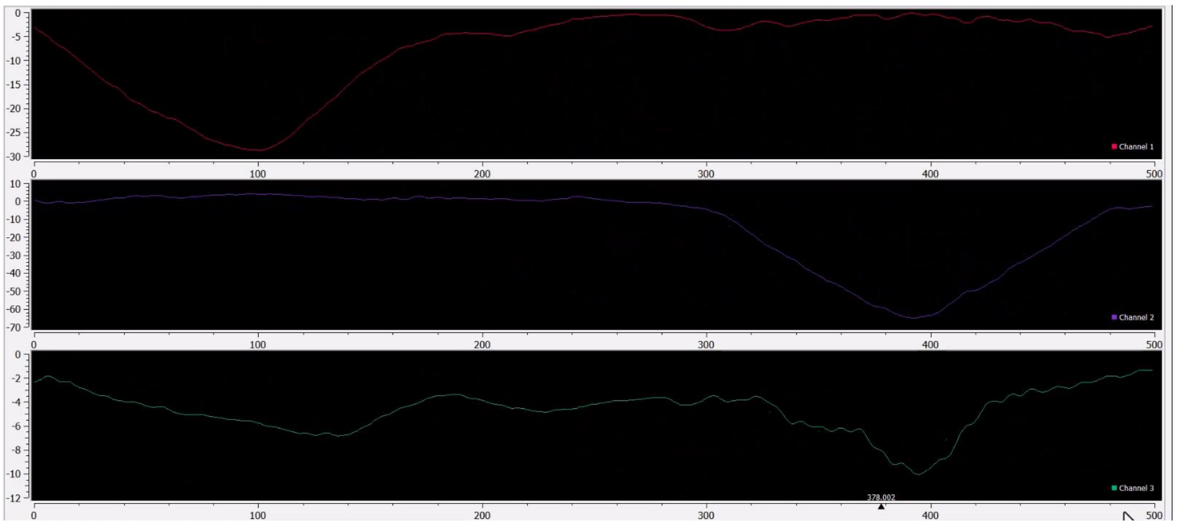
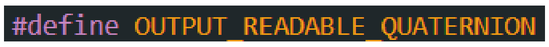


Figure 3 changes in Euler angles on the Serial-Plot

Quaternion values are then displayed by uncommenting the part that shows in Figure 4.



```
#define OUTPUT_READABLE_QUATERNION
```

Figure 4 Uncomment this line to show quaternion values

As illustrated in Figure 5, values are accurately displayed with sensor movement. This phenomenon is also observable in the video titled “Problem1_Quarernion”.

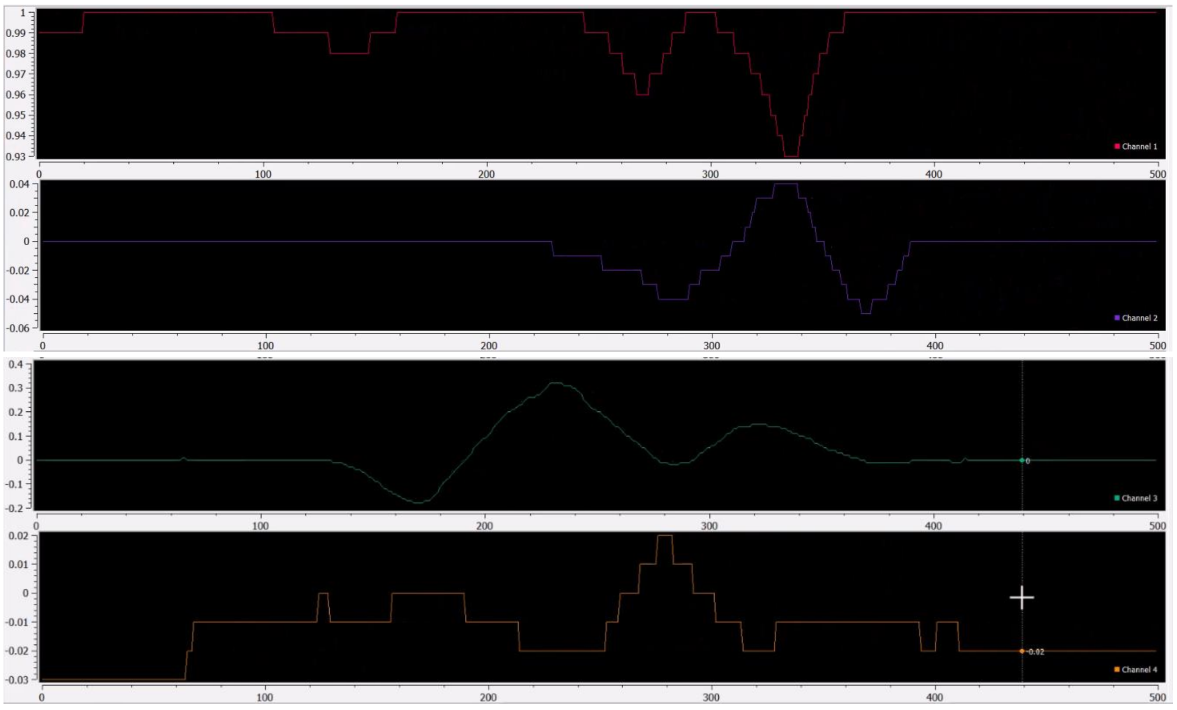


Figure 5 changes in quaternion values on the Serial-Plot

Euler angles represent a way to describe the orientation of an object in three-dimensional space using three angles: roll, pitch, and yaw. Roll is the rotation around x-axis, pitch is the rotation around y-axis, and yaw is the rotation around the z-axis. Euler angles are intuitive and easy to understand, but they suffer from a problem known as gimbal lock where one degree of freedom is lost under certain conditions.

On the other hand, quaternions are a mathematical way to represent rotations in three-dimensional space using four numbers. They provide a compact and efficient representation without suffering from gimbal lock. Each quaternion can be interpreted as a rotation from initial reference frame to a final frame. The scalar part represents the cosine of half the rotation angle, and the part represents the axis of rotation multiplied by the sine of half the rotation angle.

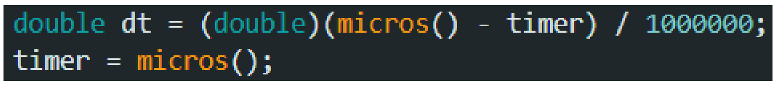
Problem 2 – Angle Filtering

In this part, we aim to evaluate the performance of adding filters to the system. Therefore, we'll examine the complementary filter once and the Kalman filter another time, comparing their performance with the unfiltered system over time.

Generally, we use filters for sensor data for several reasons: Noise reduction, Improved accuracy, Stability, Compensating for sensor limitations, Consistent output. Overall, the use of filters can help us extract meaningful information from noisy or imperfect sensor measurements, leading to more reliable and accurate results in various applications such as navigation, robotics, motion tracking, and environmental monitoring.

1. Based on the formula below, we first need to determine dt. To do this, we need to find the time when we receive new data. In other words, we need to find the time interval during which the loop is executed. To achieve this, we consider two points in the code and, using the provided code snippet in Figure 6 and with the help of the micros() function, we then convert it to seconds to obtain the time interval. It is observed that 0.01 is obtained.

$$Angle = \alpha_1 * (angle + gyroscope * dt) + \alpha_2 * accelerometer$$



```
double dt = (double)(micros() - timer) / 1000000;
timer = micros();
```

Figure 6 part of code that provide dt

Then, we need to obtain the sensor's position based on all six axes. Additionally, for correct sensor operation when determining angles and sensor position, the weight of the gyroscope should be greater than the accelerometer, and the sum of α_1 and α_2 should be equal to 1. Therefore, by experimenting with different values for α_1 and α_2, we eventually arrived at the desired values of 0.93 for α_1 and 0.07 for α_2.

2. For the Kalman filter, we utilize a ready-made algorithm available in the Arduino library. Initially, we set the initial angle using the setAngle() function provided in the respective library. Then, we obtain the angle after passing through the filter using the getAngle() function. For this filter as well, we require the angle that found in this loop, the new rate, and dt.

3.

- Complementary filter:
  - The complementary filter is a simple and effective way to combine data from multiple sensors, such as gyroscope and accelerometer, to obtain accurate orientation information.

- It works by blending the outputs of the gyroscope (which provides fast but noisy data) with those of the accelerometer (which provides slow but accurate data).
- The complementary filter calculates a weighted average of the gyroscope and accelerometer data, where the weight of each sensor's output depends on the update rate and noise characteristics of the sensor.
- By adjusting the filter coefficients, you can control the trade-off between responsiveness and accuracy of the orientation estimation.

▪ Kalman filter:
  - The Kalman filter is a more advanced filtering technique that optimally combines noisy sensor measurements over time to estimate the true state of the system.
  - It models the system dynamics and sensor noise probabilistically and updates its estimate using a recursive algorithm.
  - The Kalman filter can handle non-linearities and uncertainties in the sensor measurements better than the complementary filter.
  - However, it requires more computational resources and tuning compared to the complementary filter.

4. Images illustrating changes in angles with and without filters are presented below. Additionally, to demonstrate the performance of these filters, videos named “Problem2_roll”, “Problem2_pitch”, and “Problem2_yaw” are available.

In the images, the first plot corresponds to the system’s performance without a filter, the second plot with the Kalman filter, and the third plot with complementary filter.

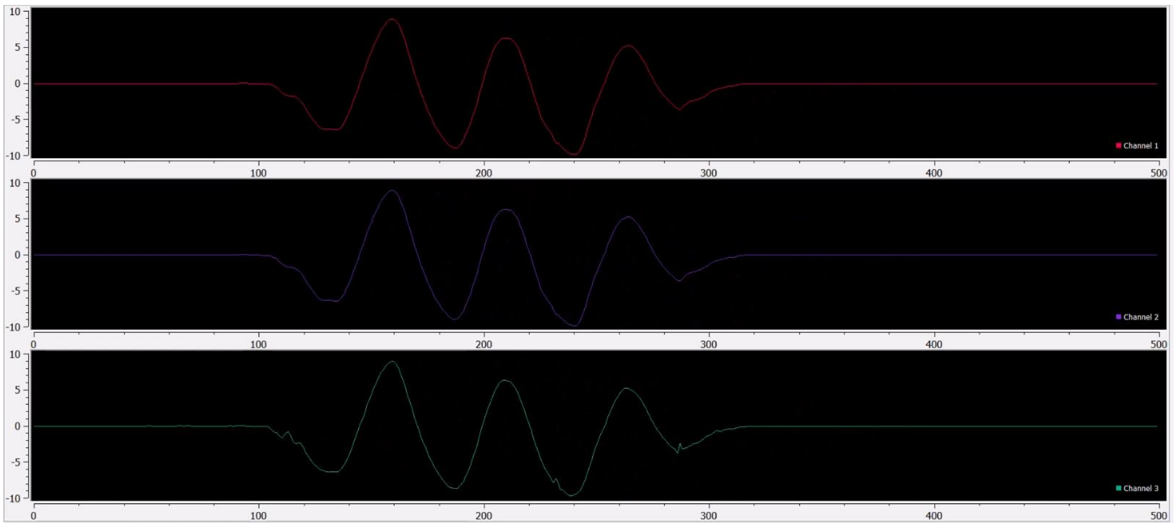


Figure 7 changes in roll angle

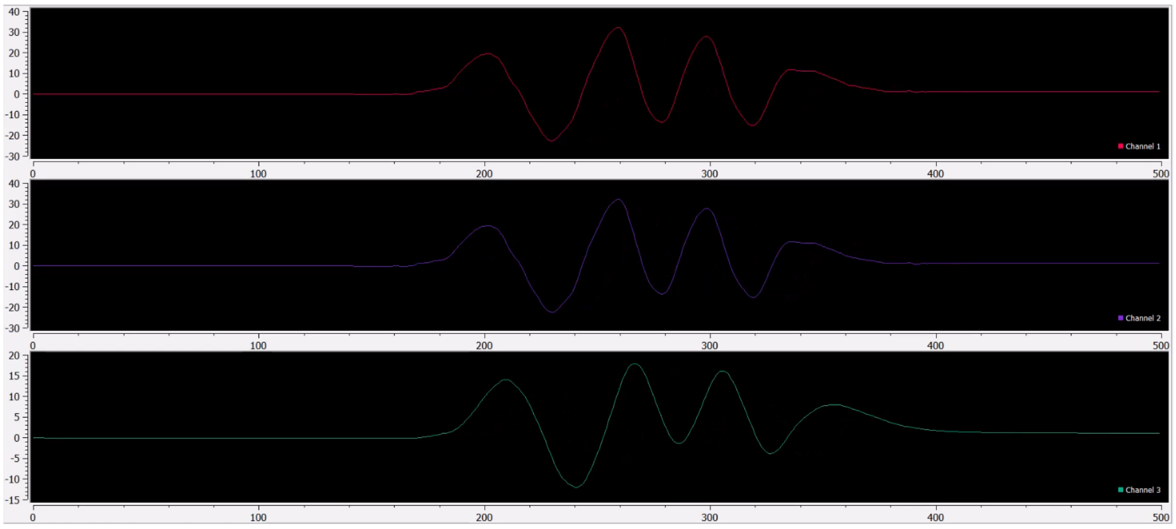


Figure 8 changes in pitch angle

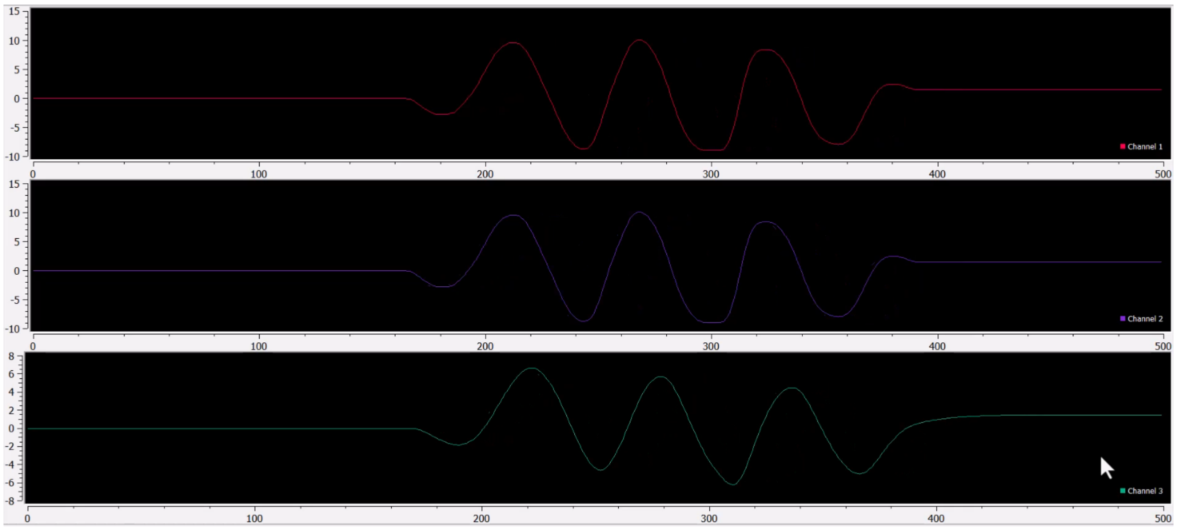
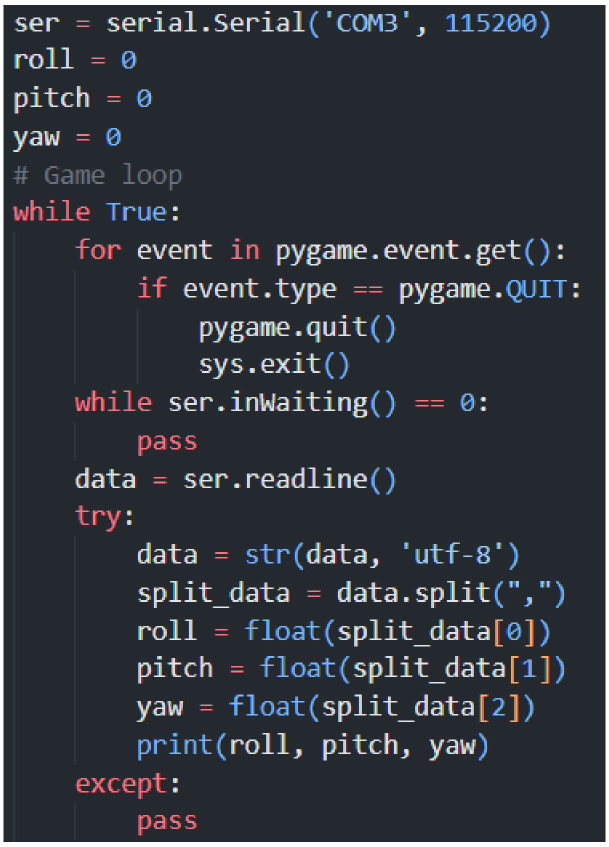


Figure 9 changes in yaw angle

Problem 3 – MPU-6050 as a Game Controller

In this section, we are asked to control and play the maze game using MPU-6050.

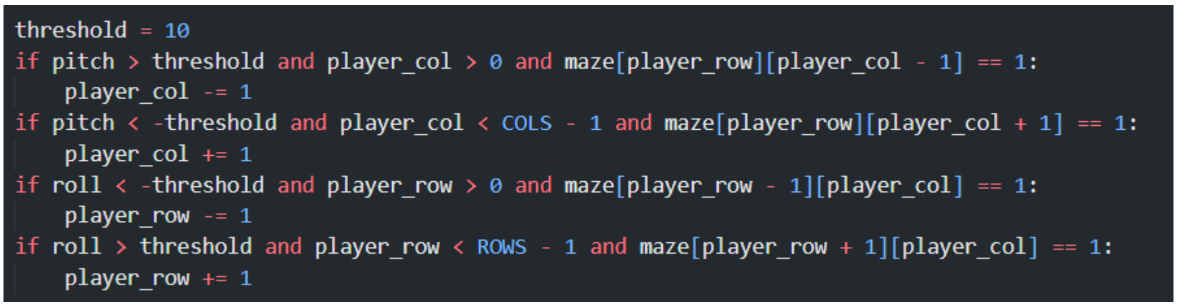
1. First, we need to establish communication between the Arduino and Python. To do this, we upload the initial code onto the Arduino, which includes reading data and obtaining Euler angles. Then, in Python, we establish the connection via serial and read the received data.



```python
ser = serial.Serial('COM3', 115200)
roll = 0
pitch = 0
yaw = 0
# Game loop
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    while ser.inWaiting() == 0:
        pass
    data = ser.readline()
    try:
        data = str(data, 'utf-8')
        split_data = data.split(",")
        roll = float(split_data[0])
        pitch = float(split_data[1])
        yaw = float(split_data[2])
        print(roll, pitch, yaw)
    except:
        pass
```

Figure 10 read the data using python

2. Now, using the data read and stored in the split_data array, we want to control the game. To move the ball based on the Euler angles obtained from the sensor. Since we only have four directions (up, down, left, and right), we only use the roll and pitch angles. If we want to move up, we move the sensor in a direction where roll becomes negative; for moving down; roll becomes positive. Similarly, for moving right, pitch becomes negative, and for moving left, pitch becomes positive. Additionally, to facilitate ball movement and better game control, we set a condition on the angle for ball movement. If the angle obtained exceeds this condition, the ball moves.



```
threshold = 10
if pitch > threshold and player_col > 0 and maze[player_row][player_col - 1] == 1:
    player_col -= 1
if pitch < -threshold and player_col < COLS - 1 and maze[player_row][player_col + 1] == 1:
    player_col += 1
if roll < -threshold and player_row > 0 and maze[player_row - 1][player_col] == 1:
    player_row -= 1
if roll > threshold and player_row < ROWS - 1 and maze[player_row + 1][player_col] == 1:
    player_row += 1
```

Figure 11 Code for controlling ball with receive serial input from the Arduino

3. A video titled “Problem3” is available to demonstrate ball routing with the sensor.

Problem 4- Motion Capturing using MediaPipe

1. Two videos with duration of 15 seconds were recorded from frontal and lateral views of the subject moving his leg. One of the videos had 30 fps and the other had 60 fps framerate, therefore the latter was down sampled as a preprocess to match the framerates.

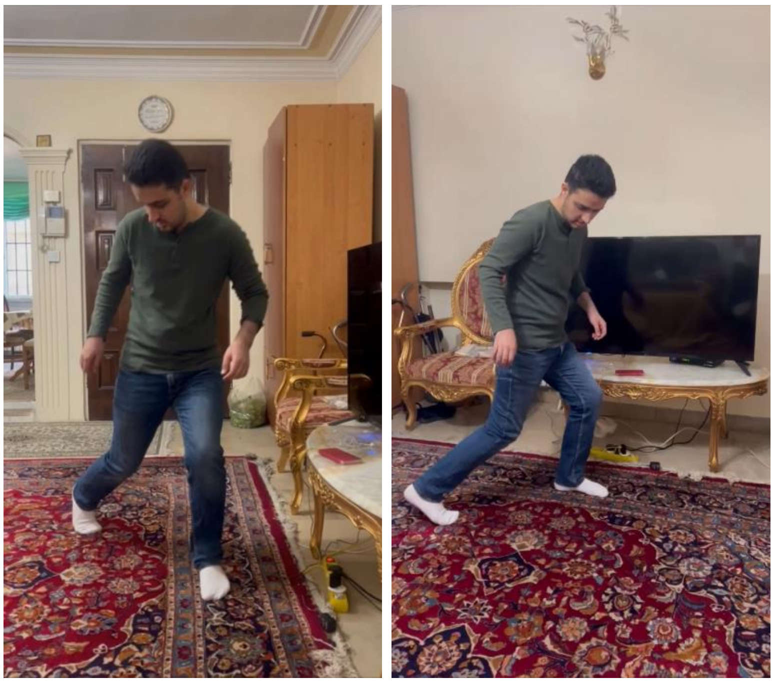


Figure 12- Sample frames of the recorded videos - Frontal (Left), Lateral (Right)

2. The pose detection task was performed using MediaPipe and the annotated videos are attached with "[view]_annotated.mp4" naming format.

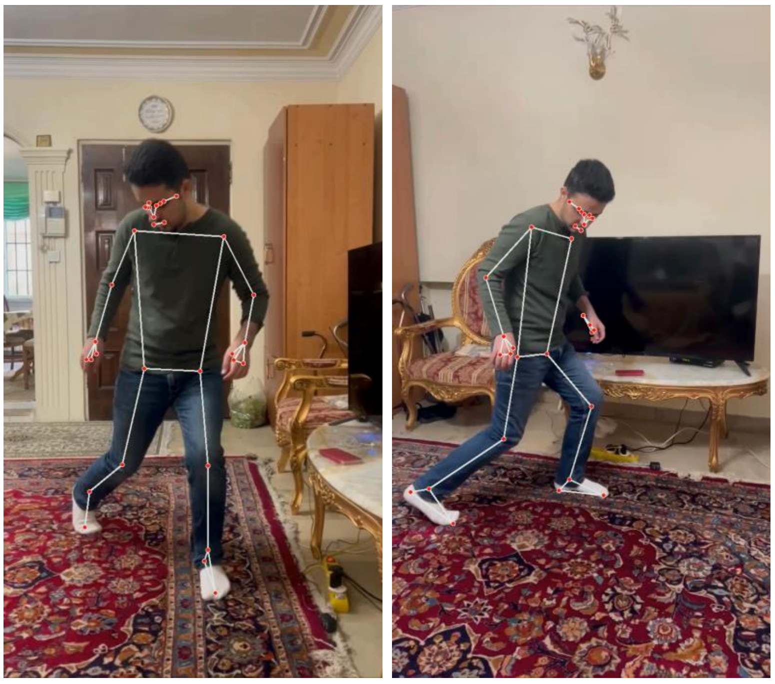


Figure 13- Sample frames of the annotated videos - Frontal (Left), Lateral (Right

Values of θ_x and θ_y are calculated from pose landmarks of the frontal and lateral views respectively and stored in "Raw_Angles.csv". They are technically the angle between the link between the right ankle and knee and vector $\hat{k} = [0\ 0\ 1]$.

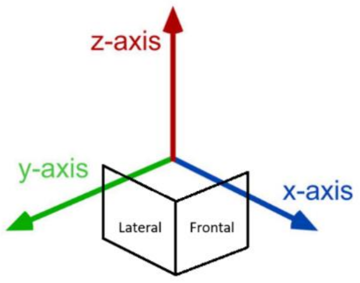


Figure 14- Demonstration of the frontal and lateral views in the global coordinates

The rotation matrix is formed by considering rotations along x-axis and y-axis within the global coordinates, therefor as product of R_x and R_y:

$$R = R_x R_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$$

3. To calculate the vector e and the quaternion at each frame, the process of extraction of natural parameters was performed on the rotation matrix of each frame:

$$\phi = acos\left(\frac{\mathrm{tr(R)} - 1}{2}\right), \qquad \vec{e} = \frac{1}{2\sin\phi}\begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix}$$

4.

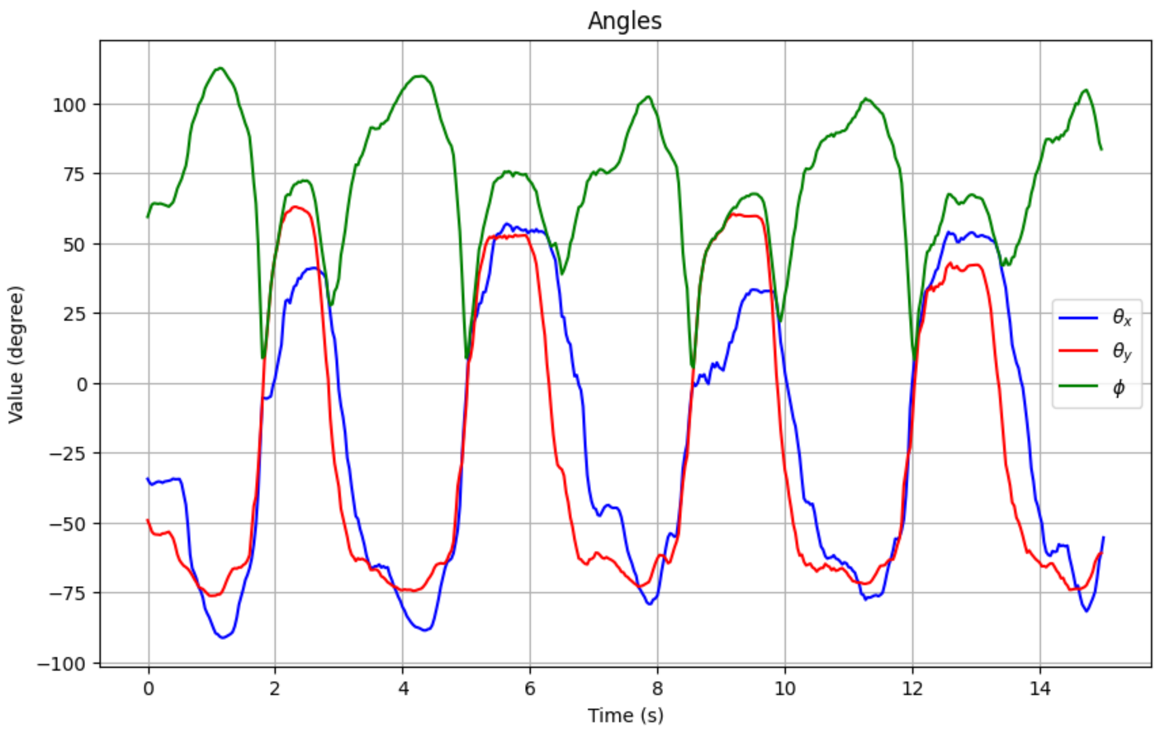


Figure 15- Plot of the angles with respect to time

5. In order to achieve a better smoothness on the plot, an offline smoothing process was performed in MATLAB. The technique used for smoothing is called the Savitzky-Golay filter. This filter fits successive sub-sets of adjacent data points with a low-degree polynomial by the method of linear least squares. The result is stored in “Smooth_Angles.csv”.

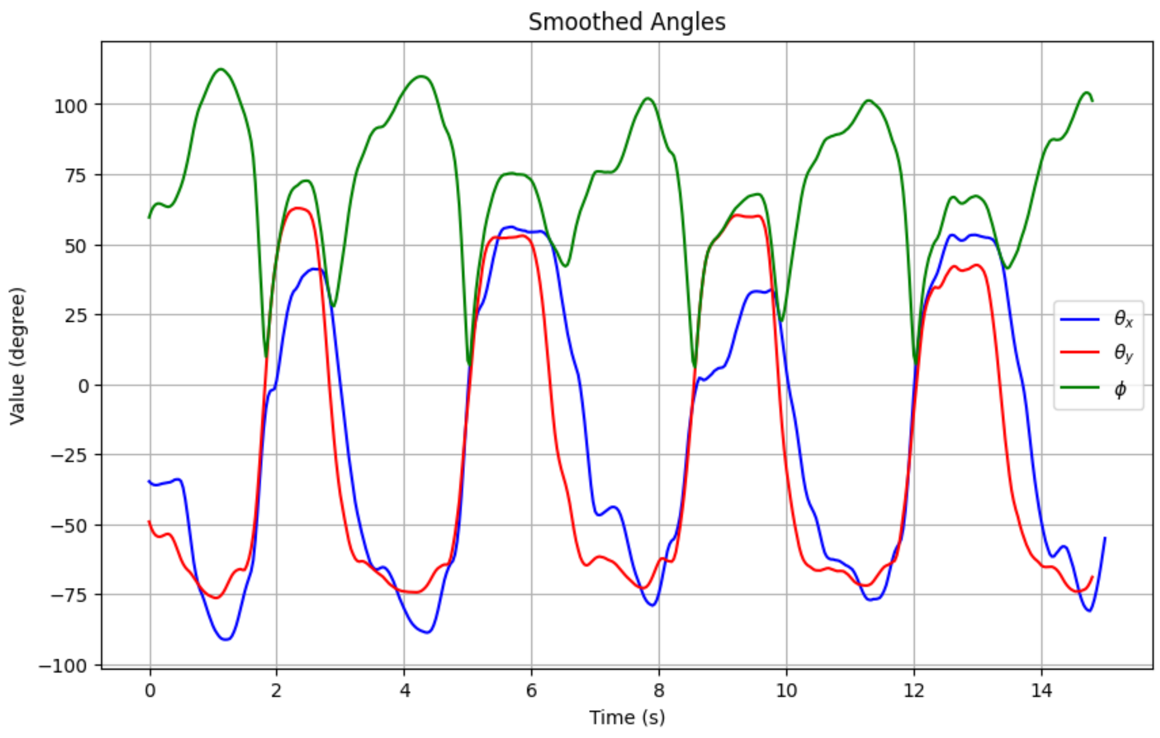


Figure 16- Plot of the smoothed angles with respect to time

The smoothing process was only applied on θ_x and θ_y but as it can be observed in Figure 16, ϕ is also smoothed after being calculated using smoothed theta angles. The filter managed to smooth the turbulances roughly shorter than 0.3 seconds (10 frames) comparing to Figure 15, as it was the window size of the filter.