Mini Project 1

1st Amir Ali Pakatchian
Electrical and Computer Engineering (ECE)
Student Number: 810199389

2nd Seyedeh Diba Ravanshid Shirazi
Electrical and Computer Engineering (ECE)
Student Number: 810199431

CONTENTS

***Abstract*—In this project we will use our knowledge from Mechatronics and Robotics course to complete the problems discussed. At first we get familiar with MPU-6050 and try to record angles and rotation matrix using Arduino. Then we see that the noisy data can be smoothed using Kalman Filter. Then we try to control a Maze Game using MPU-6050. At last we use an open-source framework named Mediapipe. It's a machine learning base solution for capturing motions and tracking body poses.We use that to find the angles and rotation of a person's leg .**

***Index Terms*—MPU-6050, Mediapipe**

I. Problem 1: Angle Recording using MPU-6050

A. Task 1

At first we try to connect Arduino to MPU-6050. We have Arduino ATMega so we have to connect SCL and SCA to the same pins on board and the power of $5V$ to the VCC (shown in Figure 1).

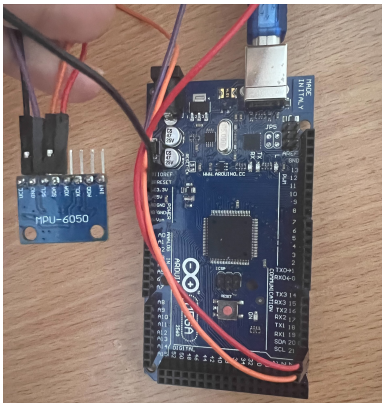

Fig. 1. The setup

Then we upload the provided code on the board. The code at first calibrates the sensor by offsets and we have to keep the sensor on a flat surface. Then we have a while loop to keep reading the angles for roll, pitch, and yaw and also the quaternion values(4 numbers). The angles are in radian so we have to convert them to degrees($*180/(2\pi)$) for better understanding. Every time we run the code we can get the angles or quaternion values. Then we move on to the python code which connects the arduino to the python world. We print the data online and see the results (Check video 1-1-1 and 1-1-2).As expected, when we rotate the sensor around X-axis the first number (roll), Y-axis the second number (pitch), and Z-axis the third number (yaw) changes respectively. As shown in the videos (Figure 2) our sensor is working properly.

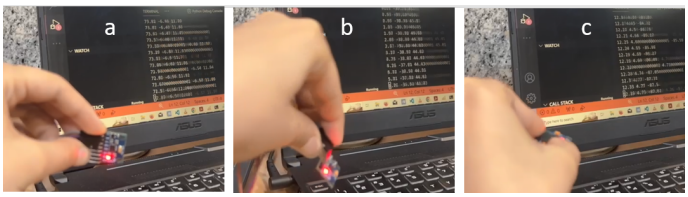

Fig. 2. a) roll, b) pitch, c) yaw

Then we see the quaternion values and we expect the values to change according to Table I.

Quaternion	Description
$(1, 0, 0, 0)$	Identity rotation
$(0, 1, 0, 0)$	roll by π
$(0, 0, 1, 0)$	pitch by π
$(0, 0, 0, 1)$	yaw by π

TABLE I
QUATERNION VALUES

So we roll, pitch, and yaw to get the results in Figure 3.

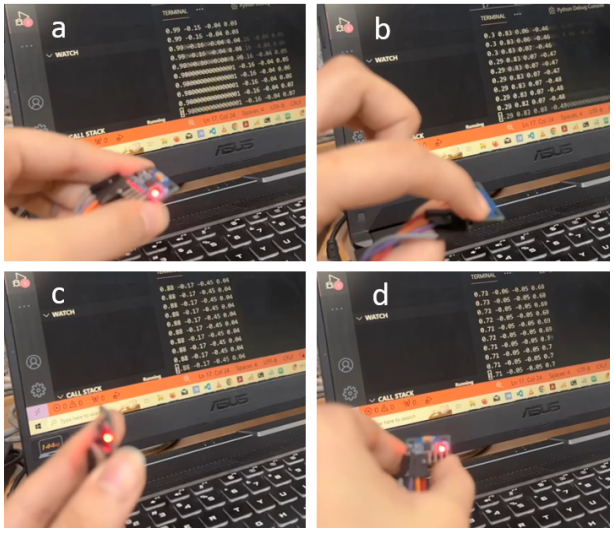

Fig. 3. a) identity b) roll, c) pitch, d) yaw

B. task 2

For better understanding and visualizing the values, we use serial plotter (Check video 1-2-1 and 1-2-2).

II. PROBLEM 2: ANGLE FILTERING

We need filtering in signal processing because of the noisy nature of the signals. Filters will help us improve accuracy, stability and get a consistent output. Here we use two filters:$FirstOrderComplementaryFilter$ and $KalmanFilter$ and compare their performance with the unfiltered signal.

A. Task 1

For the $FirstOrderComplementaryFilter$ we have to combine the data from accelerometer and gyroscope with these 2 equations:

$$Angle = \alpha_1(angle + (gyro)dt) + \alpha_2(accelerometer) \quad (1)$$

$$\alpha_1 + \alpha_2 = 1 \quad (2)$$

For finding dt in the formula we use $micros()$ function in 2 parts of our code like in Figure 4.

```
double dt = (double)(micros()-time)/1000000
time = micros()
```

Fig. 4. Code for finding dt

We can see that we get 0.01 approximately so we set$dt = 0.01$. We change α_1 and α_2 for a better result. We see that setting them $\alpha_1 = 0.94$ and $\alpha_2 = 0.06$ gives the best result. We only show results for $X - Axis(roll)$ for a better representation. We change the code so it gets data using $MPU6050_tockn.h$ library and the result is shown in Figure 5.

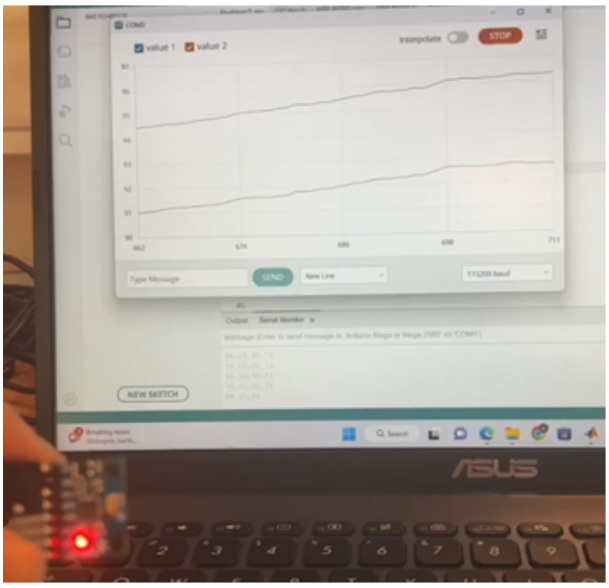

Fig. 5. Unfiltered(value 1) and filtered(value 2) signals using Complementary Filter

B. Task 2

Here we use $Kalmanfilter$ which is a powerful recursive algorithm. We used the example from the $KalmanFilterLibrary$. Which provides the libraries and the code we need for this part. Also it provides complementary filter. The results can be seen in Figure 6

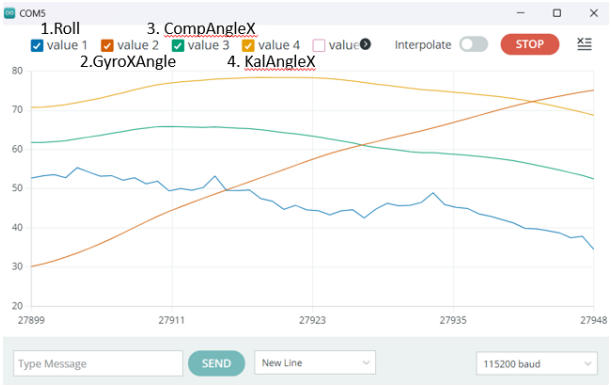

Fig. 6. Unfiltered and filtered signals using Kalman Filter and Complimentary Filter

C. Task 3

Complementary filter:

- It is pretty much easier than the other the other filter.

- It works by combining the output of gyroscope and accelerometer that have different characteristics for a better result.
- We can change the weights of this filter specific to our need for accuracy. of the sensor.
- It didn't have any significant noise reduction on our data. But it made the instant turbulence smoother.

Kalman filter:

- This filter is more advanced but takes more space and time to execute because it uses a recursive algorithm.
- It can handle non-linearity and makes the data smooth and reduces noise.

D. Task 4

The videos can be found in "video" directory (Check kalman-filter and Complementary-filter).

III. Problem 3: MPU-6050 as a Game Controller

A. Task 1

To give an overview of the code, it first imports the libraries and initialize pygame.It goes on by setting the size and color of the maze. Then it initializes the game's clock and serial communication with Arduino. The maze generator function uses stack to build a maze. The game now needs to be initialized and in the while loop of the game we add the code from problem 1 to get the roll, pitch, and yaw angles. Then in the same loop we can set rules on the received data. And also the maze need to be drawn and where we stand is shown as a circle.

B. Task 2

At first we set FPS to 300 for a faster gaming experience. Then we change the first controlling rule by simply checking whether the angle of rotations exceed or is less than a particular number. For going right and left we need to pitch (positive numbers=left,negative numbers=right). For going up or down we need to roll(positive numbers=down, negative numbers=up). To prevent the controller from being too sensitive we set the conditions from $\pm 20 degree$. The other conditions make sure we move on a way that we are supposed to and we don't enter the walls.

```
if pitch>20 and player_col > 0 and maze[player_row][player_col - 1] == 1:
    player_col -= 1
if pitch<-20 and player_col < COLS - 1 and maze[player_row][player_col + 1] == 1:
    player_col += 1
if roll<-20 and player_row > 0 and maze[player_row - 1][player_col] == 1:
    player_row -= 1
if roll>20 and player_row < ROWS - 1 and maze[player_row + 1][player_col] == 1:
    player_row += 1
```

Fig. 7. Mapping motions to directions

C. Task 3

The video of the maze game being controlled by MPU-6050 can be found in the "video" file (Check video maze-control).

IV. Problem 4: Motion Capturing Using MediaPipe

We use MediaPipe's pose detection to track a leg movement. We first have to take two videos from 2 different sides(frontal and lateral). Using pose detection we can find the coordinates of knee and ankle. Using that we will be able to answer the following tasks.

A. Task 1

The videos are provided with in the videos folder (lateral.mp4 and frontal.mp4). The videos are recorded with $30FPS$ and they are synced using Adobe Premiere Pro. The length of each video is approximately $9seconds$.

B. Task 2

To calculate θ_x and θ_y we need to use the lateral and frontal video, find the coordinates of ankle and knee using MediaPipe's pose detection(videos are marked as shown in Figure 8

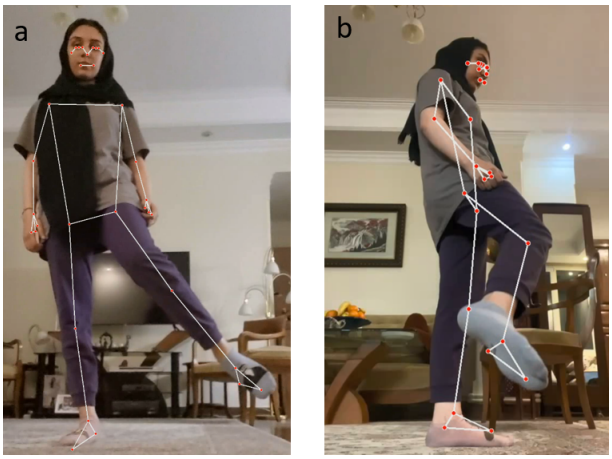

Fig. 8. Marks on Videos

Then we can calculate the angle between the line from left ankle to left knee and $(0, 0, 1)$ using $arctan$. For frontal video the left leg is detected and used to find θ_y. For lateral video the right leg is detected and used to find θ_x. We assume the x rotation was done before y rotation so we form the rotation matrix using $R.from-euler()$ function. We know that the rotation matrix for $XAxis$ and $YAxis$ rotation in global coordinate is formed like in Figure 9

$$Q_{xy} = Q_y Q_x = \begin{bmatrix} cos\theta_y & 0 & sin\theta_y \\ 0 & 1 & 0 \\ -sin\theta_y & 0 & cos\theta_y \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta_x & sin\theta_x \\ 0 & sin\theta_x & cos\theta_x \end{bmatrix} = \begin{bmatrix} cos\theta_y & sin\theta_y sin\theta_x & sin\theta_y cos\theta_x \\ 0 & cos\theta_x & sin\theta_x \\ -sin\theta_y & cos\theta_y sin\theta_x & cos\theta_y cos\theta_x \end{bmatrix}$$

Fig. 9. Rotation Matrix

C. Task 3

We have rotation matrix for each frame now using these two equations in Figure 10 we can find e and quaternion values as well.

$$\emptyset = \operatorname{acos}\left(\frac{tr(Q) - 1}{2}\right)$$

$$\vec{e} = \frac{1}{2\sin(\phi)} \begin{matrix} Q32 - Q23 \\ Q13 - Q31 \\ Q21 - Q12 \end{matrix}$$

Fig. 10. Calculating e and quaternion values

D. Task 4

The Figure 11 shows the x angle and y angle and the q_0 value in a plot over frames in time.

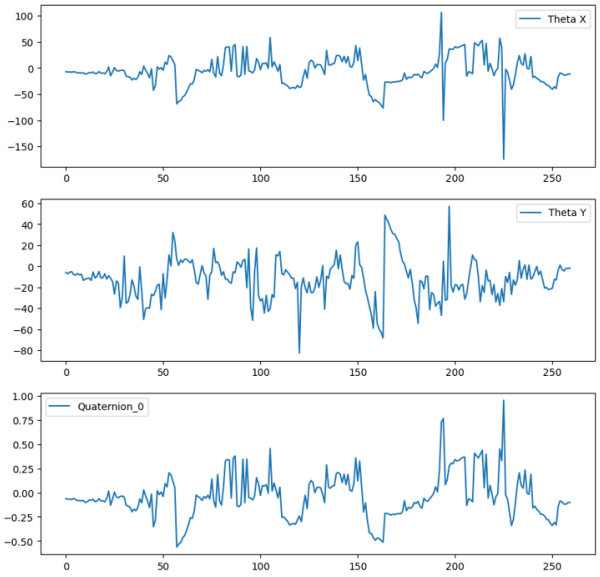

Fig. 11. Plots over frames in time

E. Task 5

For reducing noise on our data we perform moving average with $windowsize = 5$. We observe that if we increase the window size, the data will become even smoother but there is a trade-off here. We may loose our data by increasing the window size, but if the window is too small the signal will stay noisy. the plots change accordingly :

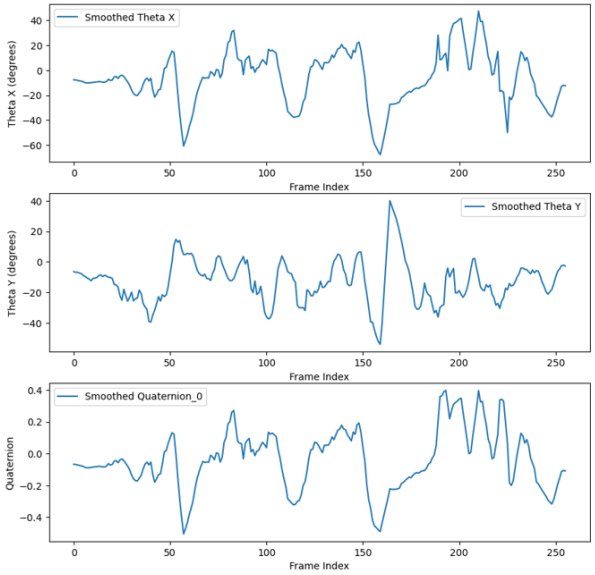

Fig. 12. Filtered plots over frames in time