به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

دانشکده مهندسی مکانیک

رباتیک و مکاترونیک

مینی پروژه شماره ۱

نام و نام خانوادگی	محیا شهشهانی-شیرین جمشیدی
شماره دانشجویی	۸۱۰۱۹۹۵۹۸-۸۱۰۱۹۹۵۷۰
تاریخ ارسال گزارش	۱۲ فروردین ۱۴۰۳

Table of Contents

Problem 1 - Angle Recording using MPU-6050

We utilized the MPU6050 sensor with an Arduino Mega 2560 for our project. The initial step after connecting the device to the computer involved calibration.

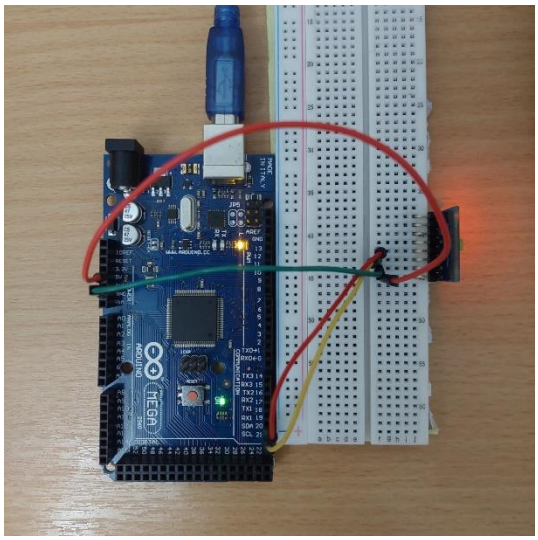

Figure 1-1: Arduino Mega2560 and MPU6050 sensor wired.

1. Placing the sensor horizontally, we utilized a provided code to establish the initial position as zero.

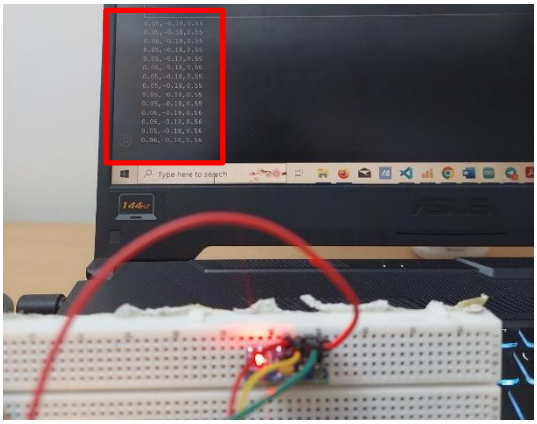

Figure 1-2: Calibrating view

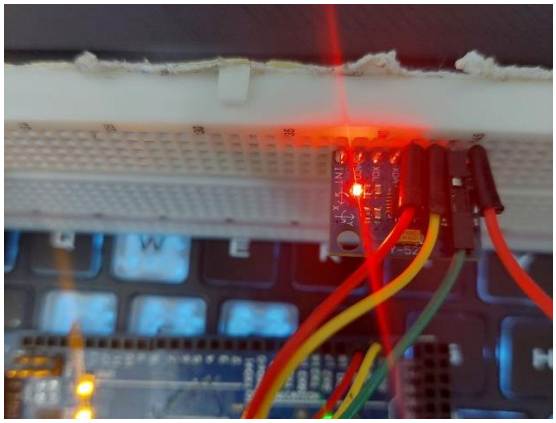

Figure 1-3: Sensor calibration up view

By rotating the sensor, we could observe roll, pitch, and yaw angles, as well as quaternion values. This foundational calibration ensured accurate and reliable data readings, setting the stage for optimal performance in our system.

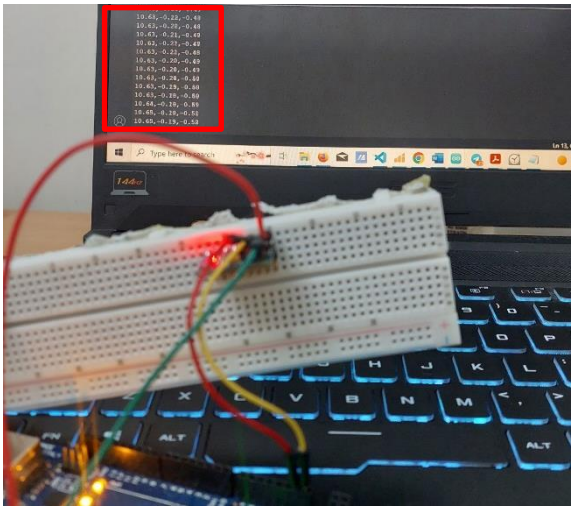

Figure 1-4: 10 degrees roll.

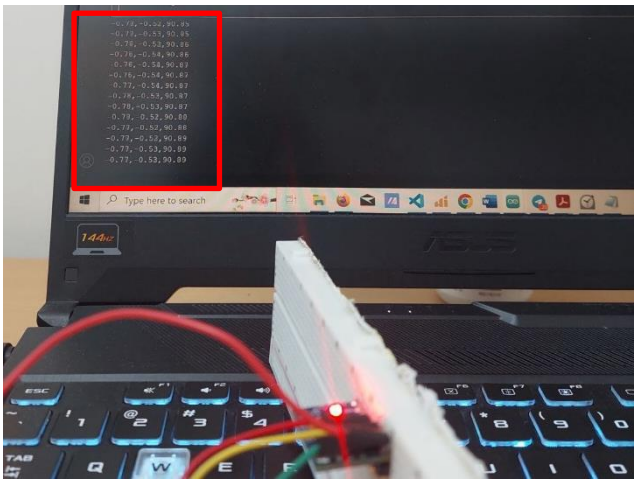

Figure 1-5: 90 degrees yaw.

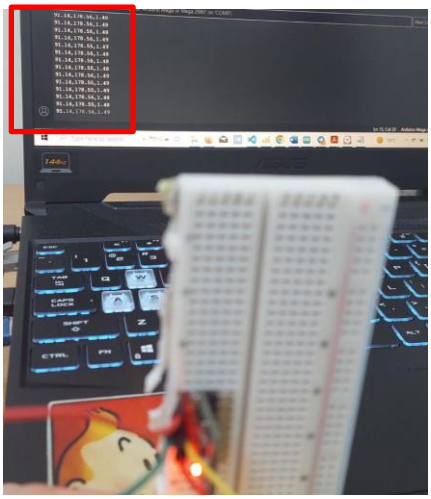

Figure 1-6: 90 degrees roll.

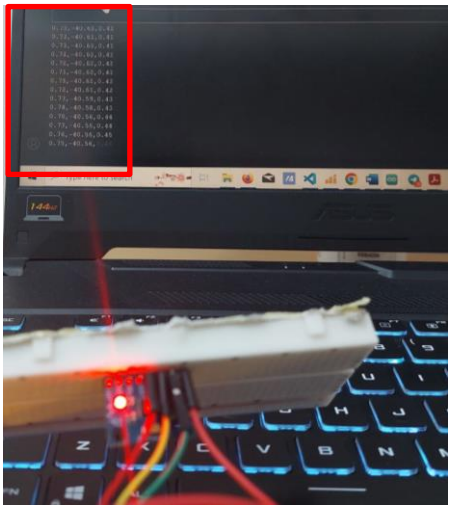

Figure 1-7: 40 degrees pitch.

We captured a video of this part's functionality, and it is available by "roll, pitch, yaw.mp4" in the uploaded folder.

By changing the code, we observed the quaternion value as well. The output resembled the following table.

Quaternion	Axis-Angle	Description
$(1,0,0,0)$	$(\text{undefined}, 0)$	Identity rotation
$(0,1,0,0)$	$((1,0,0),\pi)$	Pitch by π
$(0,0,1,0)$	$((0,1,0),\pi)$	Yaw by π
$(0,0,0,1)$	$((0,0,1),\pi)$	Roll by π
$(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}},0,0)$	$((1,0,0),\pi/2)$	Pitch by $\pi/2$
$(\frac{1}{\sqrt{2}},0,\frac{1}{\sqrt{2}},0)$	$((0,1,0),\pi/2)$	Yaw by $\pi/2$
$(\frac{1}{\sqrt{2}},0,0,\frac{1}{\sqrt{2}})$	$((0,0,1),\pi/2)$	Roll by $\pi/2$

Table 1-1: Quaternion Description

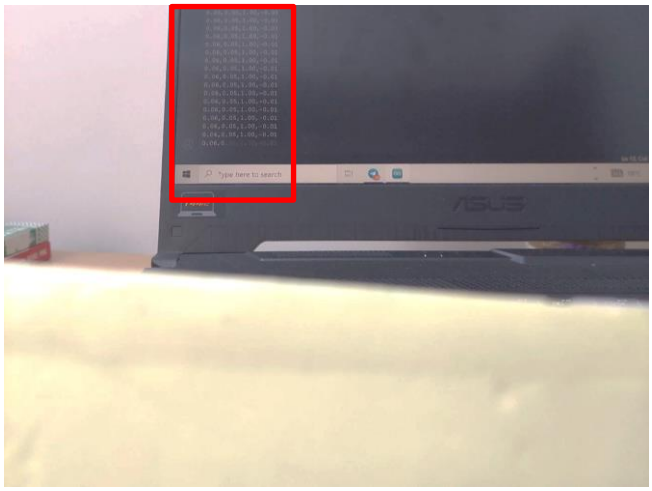

Figure 1-8: Quaternion for 90 degrees yaw.

We captured a video of this part's functionality as well, and it is available by “Quaternion.mp4” in the uploaded folder.

2. Using the Serial-Plot application, we captured a video demonstrating the sensor's functionality, which is also available by “SerialPlot.mp4” in the uploaded folder.

Problem 2: Angle Filtering

Filters serve as essential tools in signal processing, vital for extracting meaningful information from noisy data streams. By employing filters such as first-order and Kalman filters, we aim to enhance data accuracy and reliability by removing unwanted noise and disturbances.

1. In this part we implemented the First Order filter for each angle. We utilized a complementary filter with 'alpha' (0.98) weighing gyroscope data and 'beta' (0.02) weighing accelerometer data, ensuring a balanced fusion of sensor inputs for optimal filtering. The video can be found in the uploaded folder.

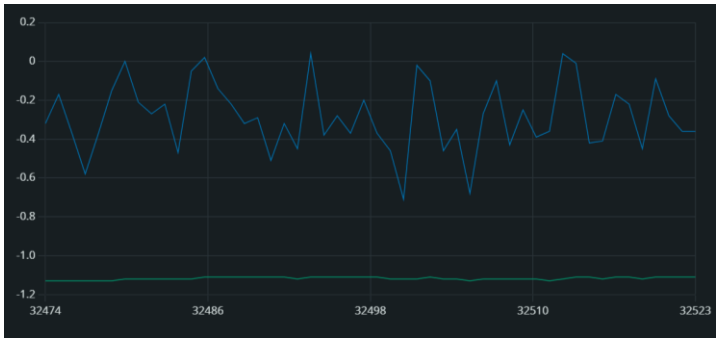

Figure 2-1: First-order filter output (green) vs original signal (blue)

2. We can see the Kalman filter effect in the following picture clearly. This filter smooths the graph and reduces noise, resulting in a more refined and stable output.

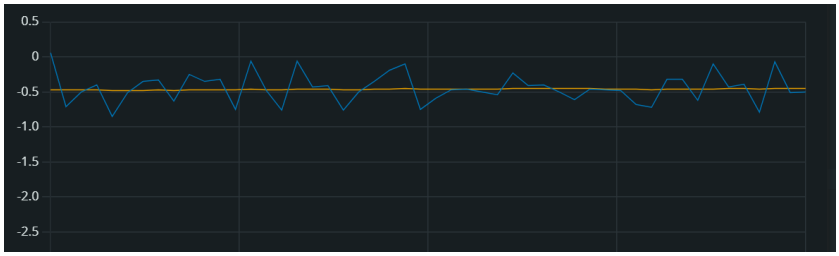

Figure 2-2: Kalman filter output (yellow) vs original signal (blue)

3. As we observed the Kalman filter and the first-order filter, we noticed some important differences. The Kalman filter constantly adjusts and corrects its estimates based on new information. It gives us accurate results even when there's a lot of noise or changes happening. On the other hand, the first-order filter is simpler but less reliable over time. It tends to drift away from the correct answer because it doesn't adjust itself like the Kalman filter does. As we can see these filters output on the following graph, the first-order filter (green) starts off close to the original signal (blue), but it gradually moves away as time goes on and has a drift. The Kalman filter(yellow), however, stays much closer to the true signal over time. So, if we need really accurate and stable results, especially over a long time, the Kalman filter is usually the better choice.

Figure 2-3: First-order output(green), Kalman filter output (yellow) and original signal (blue)

4. Videos can be found in the uploaded folder.

Problem 3: MPU-6050 as a Game Controller

As we shifted from using keyboard controls to sensor-based control in our maze game, we set up conditions based on sensor data, as shown below. Additionally, drawing from our previous part experiences, we integrated data filtered with a Kalman filter to reduce any potential drift errors that could occur over time. To improve gameplay responsiveness, we optimized the frames per second (FPS) to 500, ensuring smoother operation of the game.

The conditions are redefined as follows: (wall = 0 means there isn't any wall on the way)

If roll > 0 and wall = 0: move right

If roll < 0 wall = 0: move left

If pitch > 0 wall = 0: move up

If pitch < 0 wall = 0: move down

Figure 3-1: sensor-based control maze game

The video showcasing the code in action has been included as "Maze.mp4". For the Arduino code, please refer to the files located in the " MPU6050 for_part3" folder, while the Python code can be found by "Maze-Game.py".

Problem 4: Motion Capturing Using MediaPipe

In This Problem, we recorded two videos from the lateral and frontal views of ourselves, doing some random rotational motions in 11 seconds. Both videos are captured with 30 fps cameras. By using MediaPipe and cv2 libraries we can open the videos in Python, detect knees and ankles, and find angles in two dimensions: theta and phi.

The angles were found by the math library and using arc tangent of the differences of the knee and ankle coordinates in each frame. First, we calculate the x and y rotations for each lateral and frontal video (the outputs have been provided as frontal_output.mp4 and lateral_outputs.mp4), but for better estimates, we use the frontal video for the x rotation calculation and the lateral video for the y rotation calculation. By using these angles, we calculate the e vector and the value of quaternion at each frame(the output has been provided as output.mp4).

The results can be seen as follows:

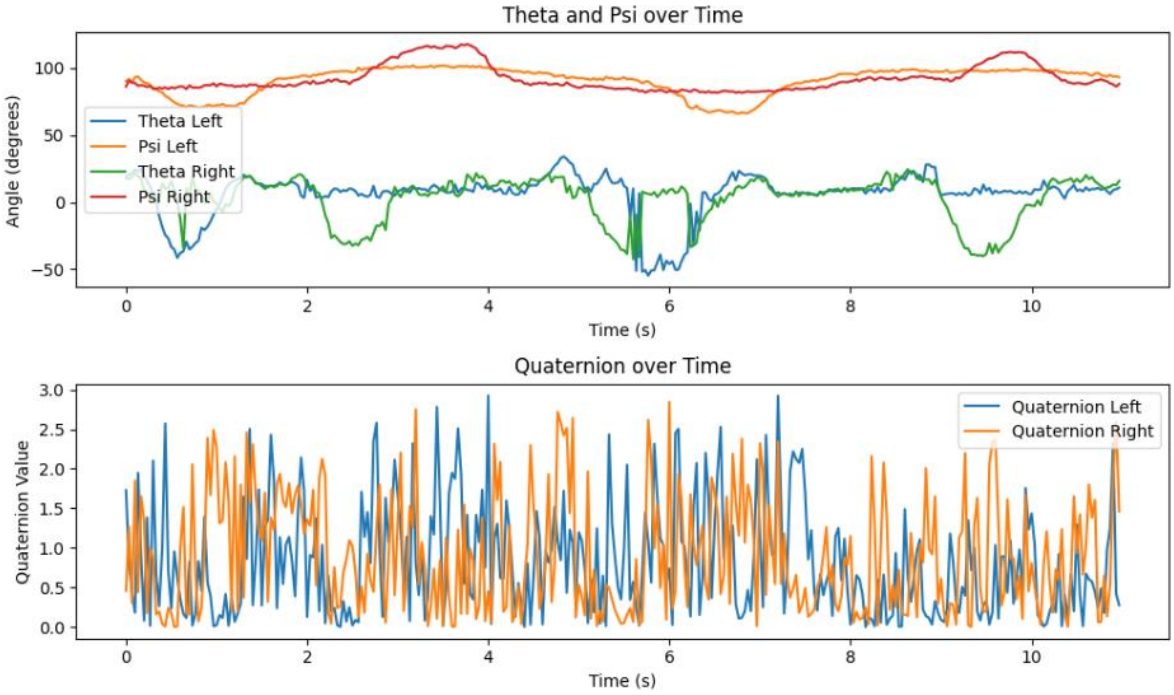

Figure 4-1: Angles and Quaternion visualization

As we can see, the plots are a little noisy. To solve this issue, we use an offline smoothening method which is “moving average”. The “moving average” is a simple and widely used technique for smoothing data. It works by replacing each data point with the average of neighboring points within a specified window size.

By using the “convolve” function in the “numpy” library, we have:

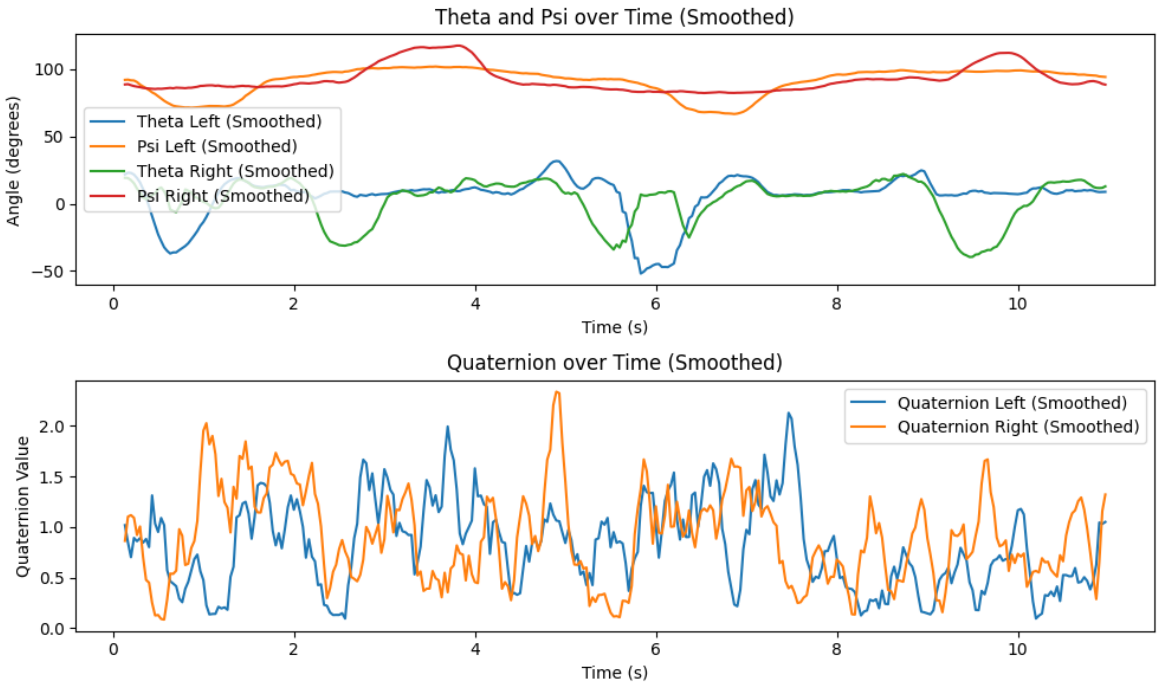

Figure 4-2: Smoothed Angles and Quaternion value visualization

We can see that the noise of the plots has been reduced significantly.

P.S: As we saw, the quaternion value oscillates in a very small range. The reason for this incident is that we’re calculating the e-vector (rotation axis) in each frame; So, it is natural to have an insignificant rotation angle in each frame.