Mechatronics & Robotics
Dr. Tale Masouleh
MiniProject 1

Alireza Kamali Ardakani
School of Mechanical Engineering
University of Tehran
Tehran, Iran
alirezakamali@ut.ac.ir

Siavash KhorramiNejad
School of Mechanical Engineering
University of Tehran
Tehran, Iran

***Abstract*—This mini-project, conducted as part of a robotics course at the University of Tehran, delves into various facets of motion sensing and control utilizing the MPU-6050 sensor and MediaPipe framework. Spanning four distinct problems, the project encompasses tasks such as angle recording, filtering, game controller application, and motion capturing. Each problem presents unique challenges and opportunities for exploration, from implementing calibration techniques and filtering algorithms to integrating sensor data for game control and precise motion tracking. Through practical implementation and analysis, students gain valuable insights into the practical applications of motion sensing technologies in robotics, reinforcing concepts learned in the classroom with hands-on experimentation and problem-solving.**

***Index Terms*—MPU-6050 sensor, motion sensing, angle recording, filtering algorithms, calibration techniques, game controller application, motion capturing, MediaPipe framework, sensor data integration**

I. Introduction

Let's first introduce the MPU-6050 sensor a little bit. The MPU-6050 is a MEMS (Micro-Electro-Mechanical Systems) device that combines a 3-axis gyroscope and a 3-axis accelerometer in a single chip. It is a popular choice for motion tracking applications due to its small size, low power consumption, and affordable price. The gyroscope component of the MPU-6050 measures the rotational velocity (how fast something is spinning) along the X, Y, and Z axes. This information can be used to track orientation and motion. The accelerometer component of the MPU-6050 measures acceleration due to gravity and motion along the X, Y, and Z axes. This information can be used to track position and movement. The MPU-6050 is a versatile sensor that can be used in a variety of applications, including robotics, drones, virtual reality, wearable devices, and gaming controllers. Now let's talk about the libraries used in the code:

A. MPU6050.h Library

The `MPU6050.h` library serves as a translator for the Arduino and the MPU6050 sensor, facilitating easy communication between them. This library simplifies complex communication details, making the sensor's data more accessible while potentially improving performance compared to writing code from scratch. It comes in various options, allowing users to pick one that suits their needs. Remember to choose a library that works with the Arduino board and sensor, and consult the library's documentation for specific use instructions.

B. I2Cdev.h Library

The `I2Cdev.h` library acts as a translator toolkit for the Arduino and its I2C devices, including the MPU-6050. It simplifies complex I2C communication details, allowing users to focus on using the sensor data. This library serves as a universal adapter, enabling work with various I2C sensors by adding specific device files. With its modular design, it keeps things organized by separating core I2C functions from individual sensor functionalities. While a basic understanding of I2C can be helpful, this library makes I2C communication much easier. Remember to consult the `I2Cdev.h` library documentation for specific use instructions.

C. MPU6050_6Axis_MotionApps20.h Library

The `MPU6050_6Axis_MotionApps20.h` library is a supercharged option for the MPU-6050 sensor on Arduino. It combines the functionality of I2C communication handling with advanced motion calculations using Invensense's MotionApps. This library provides easy-to-use data such as orientation (pitch, roll, yaw) instead of raw numbers, along with improved accuracy through Kalman filters and other techniques. While more complex than simpler libraries,

`MPU6050_6Axis_MotionApps20.h` can be a powerful tool for extracting the most out of the MPU-6050 sensor, provided users are willing to invest the effort in understanding its features.

D. KalmanFilter.h Library

The `KalmanFilter.h` library, while not a single, universal option, refers to libraries that implement the Kalman filter algorithm on various platforms such as Arduino and Raspberry Pi. These libraries help obtain smoother sensor data by removing noise and estimating system state more accurately. Users should consider the learning curve for understanding the Kalman filter itself, choose the library that best suits their platform and needs, and be prepared to tune the filter for optimal performance in their project. By using a `KalmanFilter.h` library, cleaner and more reliable information can be extracted from the sensors.

II. Problem 1

A. Task 1

In the first part of this task, we utilize the provided code from the course teaching assistant's GitHub repository to obtain quaternion values and the roll, pitch, and yaw angles, and output them to the Arduino serial monitor. To print the quaternion, we simply uncomment the line:

```
#define OUTPUT_READABLE_QUATERNION
```

The output on the serial monitor will appear as follows:

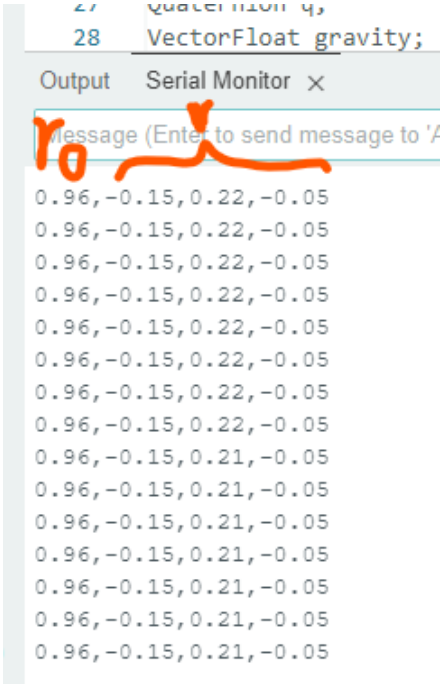

Fig. 1: Quaternion output on Arduino serial monitor.

To print the roll, pitch, and yaw angles, we uncomment the line:

```
#define OUTPUT_READABLE_YAWPITCHROLL
```

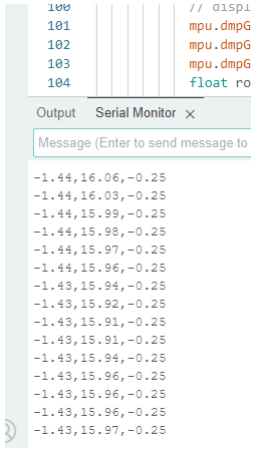

Fig. 2: Yaw, pitch, and roll angles output on Arduino serial monitor.

The output on the serial monitor will be as follows:

The Arduino code written and the demonstration video of the system's operation are attached to this report.

B. Task 2

In the second part of this task, we plot the roll, pitch, and yaw angles using the serial plotter within the Arduino IDE and once using the Serial Plotter software, and compare the results. To utilize the Arduino serial plotter, we select Serial Plotter from the Tools menu. The plotted results will indicate the variation of roll, pitch, and yaw angles over time, where roll refers to the sensor's rotation around the x-axis, pitch around the y-axis, and yaw around the z-axis. It is advisable to compare the results obtained by plotting these angles using both the Arduino serial plotter and the Serial Plotter software to validate their accuracy.

To use the Serial Plot software, follow these steps:

1) After installing, choose the correct port.
2) Adjust the Baud Rate to 115200.
3) In the Data Format tab, select ASCII followed by Comma.
4) Set the Number of Channels to 3.
5) With the Arduino serial monitor closed, click on the Open button to initiate plotting.

The results are demonstrated in the following figures:

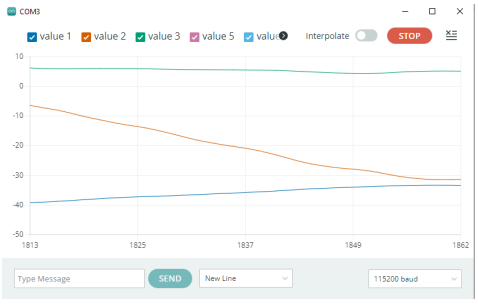

Fig. 3: Plotted angles using Arduino serial plotter.

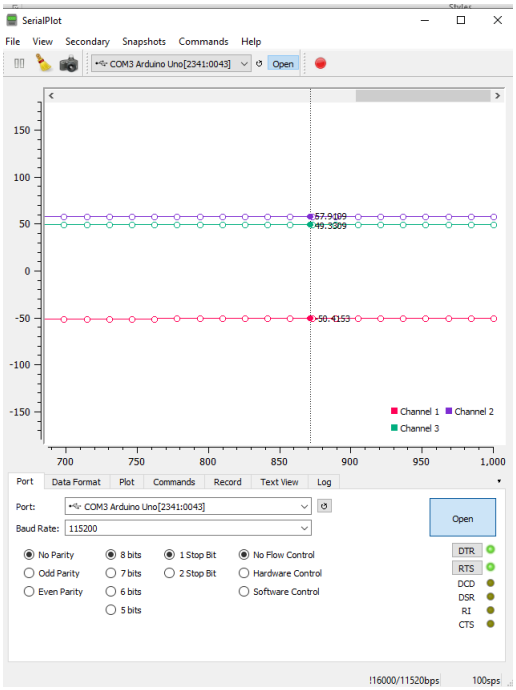

Fig. 4: Plotted angles using Serial Plot software.

Based on the experiment depicted in the video, it can be concluded that both plotting software options exhibit commendable and comparable performance speed. However, the Serial Plot software offers additional functionalities such as customizable legends, data recording abilities, and individual graph displays, setting it apart from the Arduino Serial Plotter.

III. Problem 2

A. What is the problem?

This issue involves the application of two filters, specifically the first-order complementary filter and the Kalman filter, to the output data of both the gyroscope and accelerometer of the MPU-6050 sensor in order to obtain angle data. The initial step of the task requires integrating the implementation of the Kalman filter and the first-order complementary filter into the code used in the previous request, which implies using the FIFO method for data extraction.

Since we need angular velocity data from the gyroscope and acceleration data from the accelerometer to apply the filters, using FIFO is considered impractical. Therefore, both filters are applied to the data extracted through the integration method, with their respective code available in the files labeled Problem3.ino. By running the code and evaluating the performance of the filters, a comparison can be made. The table below provides an assessment of various aspects of these filters.

Hence, it can be concluded that the first-order complementary filter exhibits the most favorable overall performance. Nonetheless, it's crucial to select the appropriate filter for each application based on the specific features needed. The last part of this question asks us to provide a video of the system performance, which is provided in the files.

B. Results

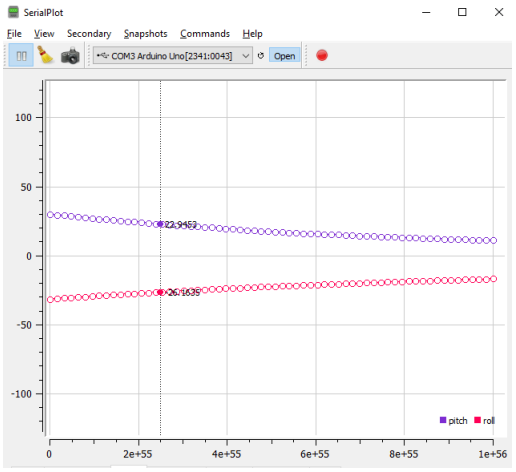

Fig. 5: Complementry Result - Serial Plotter

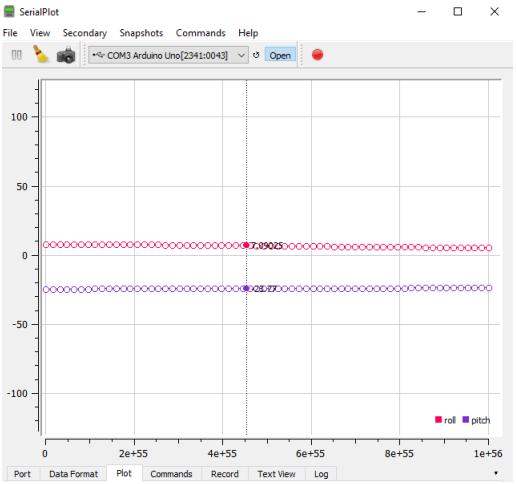

Fig. 6: kalman Result - Serial Plotter

IV. Some Arduino Points

Here are some other points about syntaxes used in Arduino Code: **DMP in MPU6050 Sensor**

- Enables hardware-accelerated motion processing capabilities.
- Reduces computational overhead on the host microcontroller.

Calibrating Sensor Offsets

- Involves compensating for biases or errors in sensor measurements.
- Factors contributing to biases include manufacturing imperfections, environmental conditions, or mounting orientations.
- Ensures accurate and reliable sensor data.

FIFO Buffer

- Utilizes the ”First In, First Out” principle.
- Acts as temporary storage for data packets from the sensor.

Quaternion Components

- **Scalar Component (q.w)** Represents rotation magnitude or angle.
- **Vector Components (q.x, q.y, q.z)** Indicate rotation axis.

Interpreting Quaternion Values

- Typically used alongside other sensor data and algorithms.
- Provides precise orientation determination in 3D space.

Configuration Settings for MPU6050

- Full-scale range and gyroscope sensitivity settings impact sensor measurements.
- Common options for accelerometer and gyroscope settings listed.
- Specific settings typically configured using functions provided by the MPU6050 library.

Wire.h Library

- Facilitates I2C communication.
- Defines necessary variables for data storage.

ADO Pin Functionality

- Determines the I2C address of the module.
- Default address is 0x68HEX when left unconnected.
- Address becomes 0x69HEX when connected to 3.3V.

V. Problem 3

A. What is the Problem?!

The problem entails creating an interactive maze game where the player controls a ball's movement by physically tilting and rotating an MPU-6050 accelerometer/gyroscope sensor. Currently, the game is controlled using keyboard keys (WASD) to navigate the ball through the maze. The task is to replace this control scheme with input from the motion sensor, making the gameplay more physical and interactive. This involves connecting an Arduino board to Python, reading data from the sensor, and modifying the Python game code to interpret the sensor inputs for controlling the ball's movement. Collision detection with maze walls remains a key aspect to prevent illegal passage through walls. Finally, the solution should be demonstrated via a video recording showing the setup and gameplay interaction using the motion sensor, with a maximum duration of 30 seconds.

In this question, firstly, according to the code that was uploaded on GitHub and the many searches that were done, we looked at how the code written to produce the maze game works. According to the comments that were written in the code, the code was written according to the reading of the sensor from the Arduino and the transfer of the information received from the roll and pitch angles. In the numerous searches that were done, it is necessary to mention some points about the code written here. Therefore, the difference between the written code and the raw code of the maze game is pointed out, and a picture of the results of the game can be seen below. In addition, a video of how it works is also attached to the report.

B. Key Differences between 2 Codes

- **Library Import:**
  Code 1 imports basic libraries like pygame, sys, random, and time, which are commonly used for creating games and managing program execution. Code 2 expands the library import list by including additional libraries such as serial, numpy, and vpython. Serial is used for establishing communication with the Arduino, numpy for numerical computations, and vpython for 3D visualizations.
- **Serial Communication:**
  In Code 1, there's no provision for serial communication with the Arduino. However, Code 2 initiates serial communication with the Arduino by setting up a serial port (`arduino_data = serial.Serial('com3', 115200)`) and includes functions like `ignore_data()` and `read_controller()` to handle data communication with the MPU-6050 sensor connected to the Arduino. These functions facilitate the reading of sensor data, particularly the roll and pitch values.
- **Maze Generation:**
  Both codes have a function called `generate_maze()` to create a maze for the game. However, Code 2 introduces an alternative version of this function with a slightly different algorithm, although the core purpose remains the same: to generate a maze layout.
- **Player Movement:**
  Code 1 allows player movement using the keyboard keys W-A-S-D for up, left, down, and right directions respectively. On the other hand, Code 2 integrates sensor input from the MPU-6050 to control player movement. It reads data from the sensor and adjusts the player's position based on the tilt of the sensor. If the tilt exceeds certain thresholds, it moves the player accordingly.
- **Player Position Update:**
  In Code 1, player position updates are triggered by keyboard events. Conversely, in Code 2, player position updates are driven by sensor data. If the sensor detects tilting beyond certain thresholds, the player's position is updated accordingly.
- **Additional Libraries:**
  Code 2 introduces the use of additional libraries such as numpy and vpython. Numpy is utilized for handling numerical operations, which might be required for processing sensor data or performing mathematical calculations related to game mechanics. Vpython is used for visualizations, although its specific purpose in this context is not explicitly mentioned in the provided code snippets.
- **Code Structure:**
  Code 2 is slightly more structured and modular compared to Code 1 due to the inclusion of additional functions (`ignore_data()` and `read_controller()`) to manage serial communication and sensor input. This modular approach enhances code readability and maintainability.

C. Results

Roll and pitch angles are printed on the side of the game to follow the values taken from the Arduino. It should be noted that the movement threshold for the player is 0.3 radians. This means that the size of the angles must be greater than this value for the player to move. But after this amount, there is no limit and the player continues to move. Between two values 0.3 and -0.3 nothing happens and the code continues.

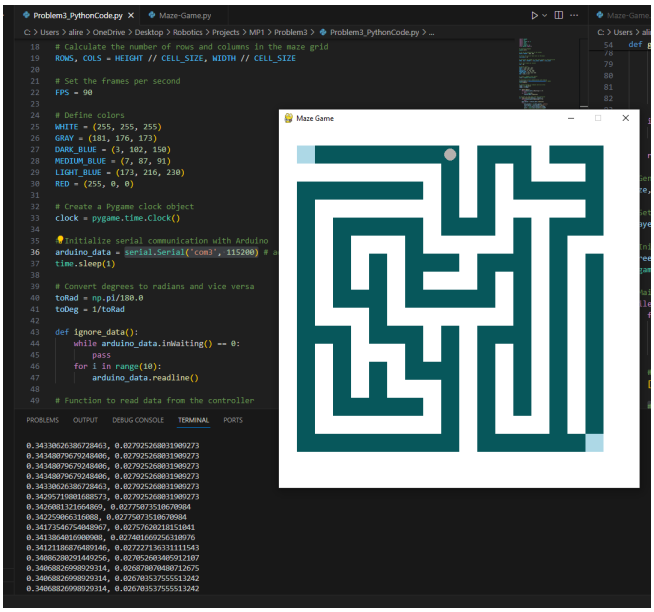

Fig. 7: Maze Game Result

VI. Problem 4

What is the Problem?

The task involves utilizing MediaPipe's pose detection system to capture and analyze the 2D rotational movement of a football player's leg in a video. Because of MediaPipe's limitations in depth estimation, it's necessary to record data from two perspectives simultaneously: the leg's lateral and frontal views. The assignment includes several steps:

- Record two videos capturing the leg's rotational motion, ensuring both cameras have matching frame rates.
- Calculate x and y values for each frame to construct the leg's rotation matrix (between ankle and knee).
- Determine the vector e and quaternion value for each frame.
- Plot x, y, and quaternion values over time.
- Identify noise in the plotted data, conduct an offline smoothing process—potentially using MATLAB functions—and then compare the smoothed plots with the original ones.

This task aims to demonstrate MediaPipe Pose's application in motion capture and analysis, specifically focusing on the rotational motion of a particular body segment.

Writing Code

Initially, it was essential to grasp a basic understanding of how MediaPipe and OpenCV operate, as outlined in the provided references. This involved running various code snippets and examining multiple examples. By referring to [1] and [2] , it becomes evident that processing a video entails placing image processing code within a Python loop. Essentially, this means that Python processes and analyzes video frames one by one, calculating the angular position of the specified body part (in this case, the leg) in each iteration of the loop. At the conclusion of the loop, after performing the necessary calculations and determining the desired values, these values are overlaid onto the output video. While all the updated code explanations are detailed below, the Python file for this task includes comments providing explanations at each step.

Important Points about Code

1) **Installing MediaPipe Library:**
   The `mediapipe` library is installed using `!pip install -q mediapipe`. This package enables pose estimation tasks.
2) **Mounting Google Drive:**
   Google Drive is mounted using Google Colab's `drive.mount()` method. This step allows access to files stored in Google Drive.
3) **Function Definitions:**
   Several functions are defined:
   - `calculate_angle(a, b)`: Calculates the angle between two points using trigonometry.
   - `generate_matrix_Qx(c)`: Generates a rotation matrix about the x-axis.
   - `generate_matrix_Qy(c)`: Generates a rotation matrix about the y-axis.
   - `vect(Q)`: Computes the vector part of a quaternion from a rotation matrix.
4) **Importing Libraries:**
   Necessary libraries such as `cv2`, `mediapipe`, `numpy`, `sys`, `time`, and `matplotlib.pyplot` are imported.
5) **Main Code:**
   Variables are initialized to store angles and timestamps. `mediapipe` objects for pose estimation are initialized. Paths to two video files are defined, and the videos are opened using OpenCV's `cv2.VideoCapture()`. Video properties like frame rate, total frames, and duration are retrieved. Output video writers for annotated videos are defined. Inside the main loop, frames are read from both videos, detections are made using MediaPipe, and frames are processed. Angles are calculated, rotation matrices are generated, quaternion values are computed, and they are visualized on the frames. Processed frames are written to output videos, resources are released, and the `mediapipe` object is closed.
6) **Plotting:**
   Angles and `r0` values are plotted against timestamps using `matplotlib.pyplot`.
7) **Transfer Data to MATLAB:**
   Angle data, `r0` values, and timestamps are saved to a MATLAB `.mat` file using `scipy.io.savemat()`.

Results

Process 2 Videos

The results of this code can be seen here. It should be noted that the coordinate system is shown in both videos and the reported angle is the angle of rotation around the axis of rotation, which is reported with respect to the other axis.

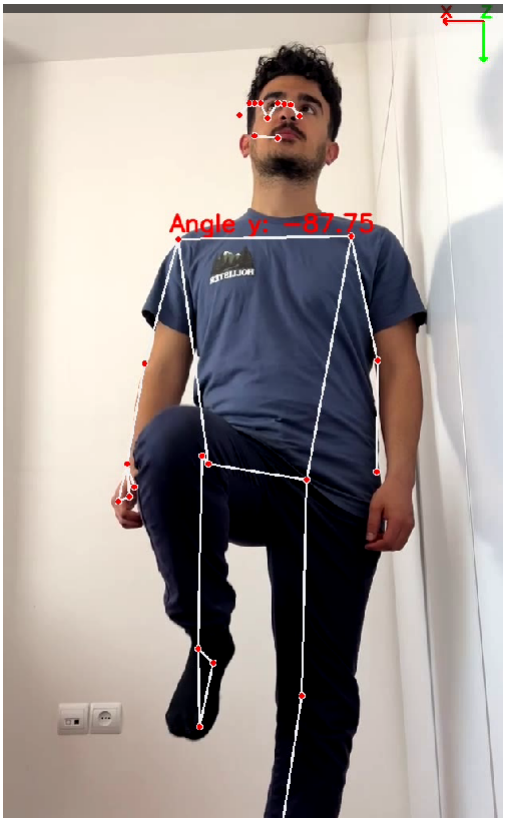

Fig. 8: Proccessed Video from Front View

Plotting

MATLAB code performs the following tasks:

1) **Loading Data:** It loads data from a saved `.mat` file named 'Problem4_PythonData.mat'.
2) **Data Extraction:** It extracts the loaded data into separate variables:
   - `angles_x`: Represents data for angles in the x direction, extracted from the field `phi` in the loaded data.
   - `angles_y`: Represents data for angles in the y direction, extracted from the field `theta` in the loaded data.
   - `r0_values`: Represents quaternion values, extracted from the field `Quaternion` in the loaded data.
   - `timestamps1`: Represents timestamps associated with `angles_x` and `r0_values`, extracted from the field `Time1` in the loaded data.
   - `timestamps2`: Represents timestamps associated with `angles_y`, extracted from the field `Time2` in the loaded data.
3) **Smoothing:** It applies smoothing to the extracted angle and quaternion data using different smoothing algorithms:

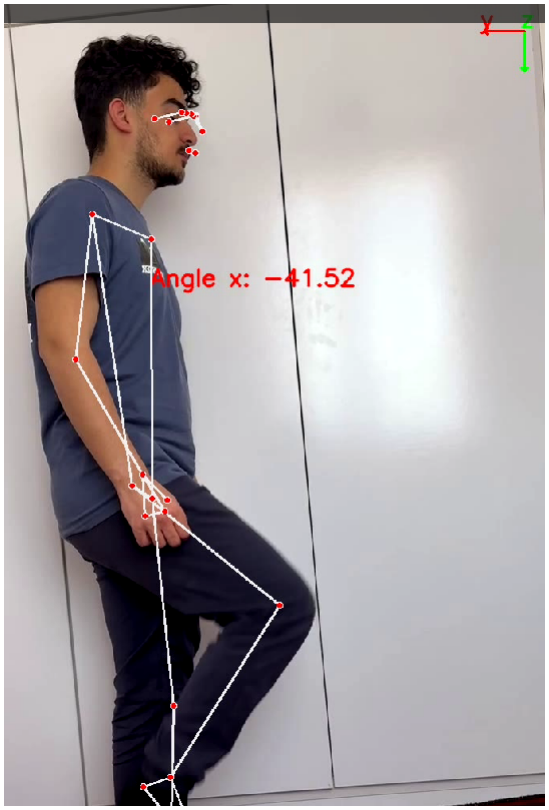

Fig. 9: Proccessed Video from Lateral View

   - `smooth_angles_x`: Smooths the `angles_x` data using a Gaussian smoothing filter with a window size of 10.
   - `smooth_angles_y`: Smooths the `angles_y` data using a lowess (locally weighted scatterplot smoothing) algorithm.
   - `smooth_r0_values`: Smooths the `r0_values` data using a moving average filter with a window size of 5.
4) **Plotting:** It plots the smoothed data:
   - The first subplot (`subplot(3, 1, 1)`) shows the smoothed angle in the x direction (`smooth_angles_x`) against `timestamps1`.
   - The second subplot (`subplot(3, 1, 2)`) shows the smoothed angle in the y direction (`smooth_angles_y`) against `timestamps2`.
   - The third subplot (`subplot(3, 1, 3)`) shows the smoothed quaternion values (`smooth_r0_values`) against `timestamps1`.

These plots provide visual representations of the smoothed angle data and smoothed quaternion values over time, allowing for easier analysis and interpretation of the data trends.

VII. Conclusion

In this project, we addressed the problem of applying two filters, the first-order complementary filter and the Kalman filter, to the output data of the gyroscope and accelerometer of the MPU-6050 sensor in order to derive angle data. Our initial challenge involved integrating the implementation of

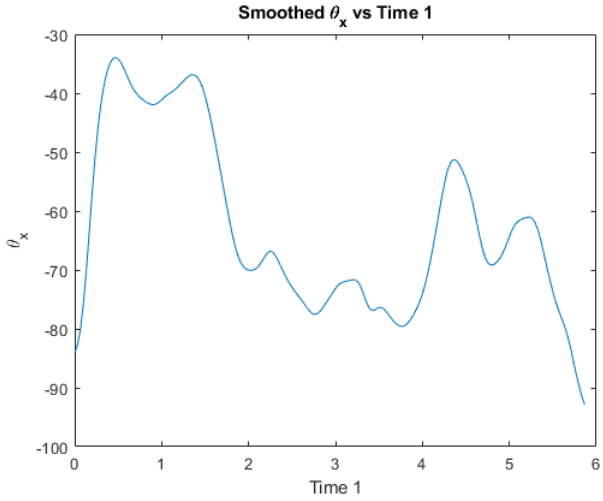

Fig. 10: Smoothing Process on θ_x from Python in MATLAB Software

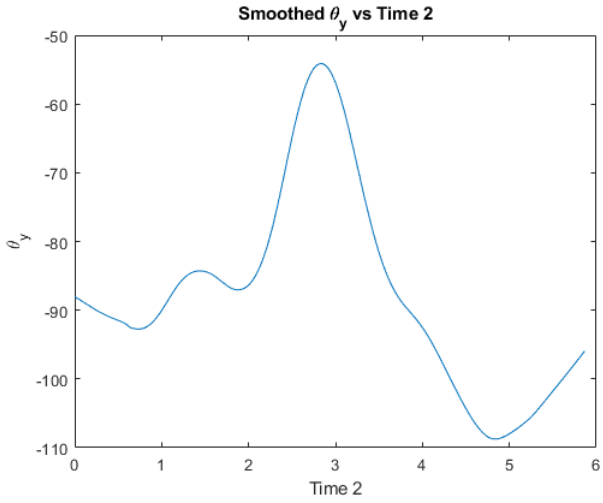

Fig. 11: Smoothing Process on θ_y from Python in MATLAB Software

these filters into the existing codebase, necessitating the use of the FIFO method for data extraction.

However, we encountered a significant obstacle in using FIFO due to the requirement for both angular velocity data from the gyroscope and acceleration data from the accelerometer. As a result, we found it impractical to rely solely on FIFO for our data extraction needs.

To overcome this challenge, we applied both filters to the data extracted through the integration method. By running the code and evaluating the performance of the filters, we were able to compare their effectiveness. Our results indicated that the first-order complementary filter exhibited the best overall performance in this context.

Nevertheless, it is crucial to note that the choice of filter should be made based on the specific requirements of each application. While the first-order complementary filter may have performed well in our scenario, other situations may call for the use of different filters tailored to the unique features of the data.

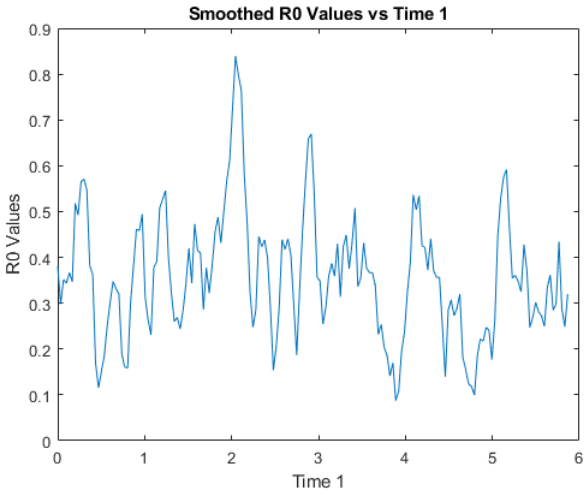

Fig. 12: Smoothing Process on r_0 values from Python in MATLAB Software

In conclusion, this project has provided valuable insights into the application of filters in sensor data processing. Moving forward, further research and experimentation could explore additional filtering techniques and their implications for real-world applications.

VIII. Conclusion

In this project, we addressed the problem of applying two filters, the first-order complementary filter and the Kalman filter, to the output data of the gyroscope and accelerometer of the MPU-6050 sensor in order to derive angle data. Our initial challenge involved integrating the implementation of these filters into the existing codebase, necessitating the use of the FIFO method for data extraction.

However, we encountered a significant obstacle in using FIFO due to the requirement for both angular velocity data from the gyroscope and acceleration data from the accelerometer. As a result, we found it impractical to rely solely on FIFO for our data extraction needs.

To overcome this challenge, we applied both filters to the data extracted through the integration method. By running the code and evaluating the performance of the filters, we were able to compare their effectiveness. Our results indicated that the first-order complementary filter exhibited the best overall performance in this context.

Nevertheless, it is crucial to note that the choice of filter should be made based on the specific requirements of each application. While the first-order complementary filter may have performed well in our scenario, other situations may call for the use of different filters tailored to the unique features of the data.

In conclusion, this project has provided valuable insights into the application of filters in sensor data processing. Moving forward, further research and experimentation could explore

additional filtering techniques and their implications for real-world applications.

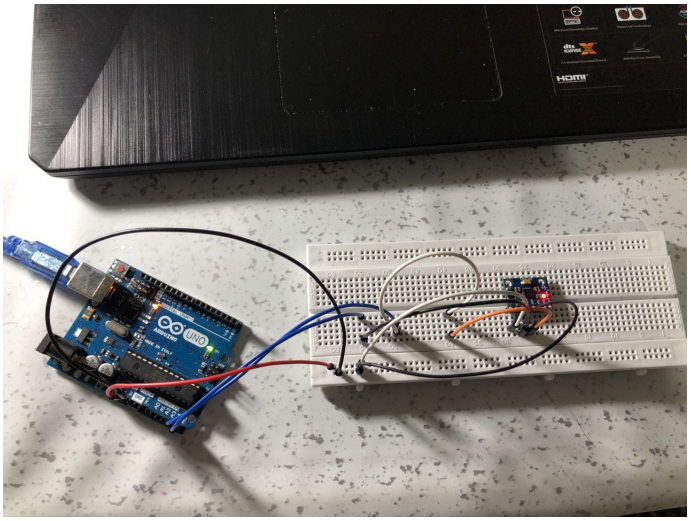

Fig. 13: Our MPU6050 Setup

REFERENCES

[1] https://github.com/google/mediapipe/issues/1589

[2] AI Body Language Decoder with MediaPipe and Python in 90 Minutes, Nicholas Renotte, 2021, Youtube, https://www.youtube.com/watch?v=We1uB79Ci-w

[3] https://developers.google.com/mediapipe/solutions/vision