

## Part 1 – Data Visualization and Feature Inspection:

### Objective:

To visualize accelerometer data for different human activities and identify signal characteristics that can distinguish them.

### Command:

*`python inspect_data.py --file data/walking.csv`*

### Method

The sensor data was gathered with the Phyphox app on an iPhone 11. Each action (*sitting, standing, walking, and running*) was recorded for about 30 to 40 seconds. While collecting data, the phone was put in the left pant pocket with the screen facing out to keep the same direction in all sessions. For the walking and running actions, straight routes that were flat and clear of obstacles were chosen so that we could reduce unevenness caused by rough ground or changes in direction. All recordings were done in similar environmental conditions to keep the data consistent and comparable. Afterward, the accelerometer data was graphed as X, Y, and Z axes and as the total acceleration magnitude. The signals were visually checked to study differences in size, amount of change, and rate between the actions.

### Code:

We have chosen and implemented Python's standard libraries, such as; *csv*, *math*, *matplotlib.pyplot*, *os*, and *shutil*.

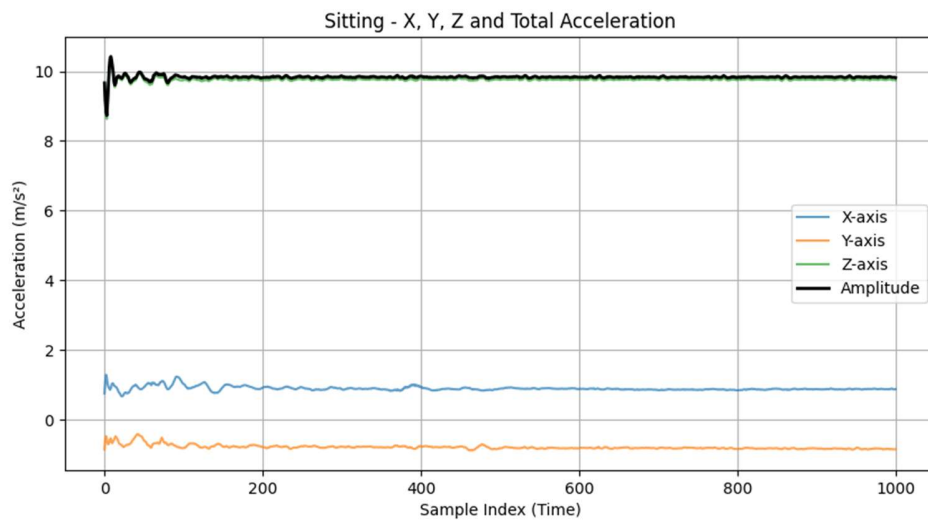
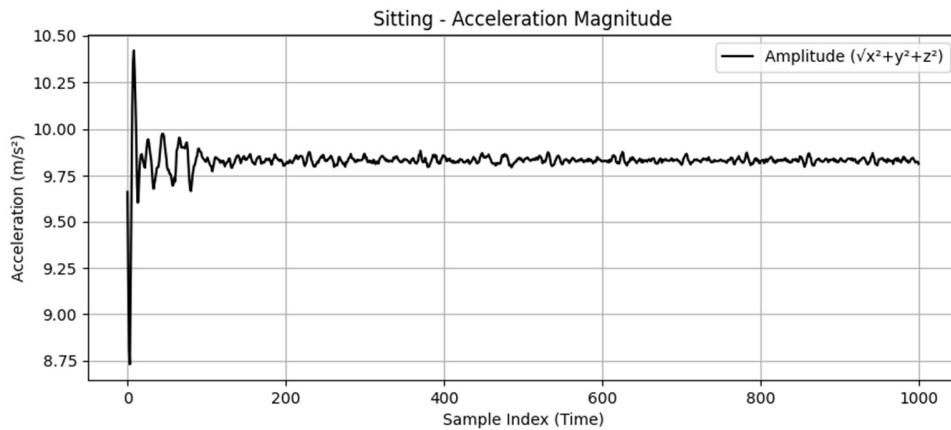
The data/ folder contains the raw accelerometer CSV files for four activities: sitting, standing, walking, and running. The script, inspect\_data.py, computes magnitudes from the files and makes individual and combined plots. All figures are saved in the outputs/ directory to keep data, code, and results separate for reproducibility.

### Results:

After plotting and visually inspecting the accelerometer data for all four activities — sitting, standing, walking, and running — clear differences can be observed in the shape, amplitude, and frequency of the signals. Each plot shows how the acceleration values change over time, both as a total magnitude and as individual X, Y, and Z axis components.

#### *Sitting:*

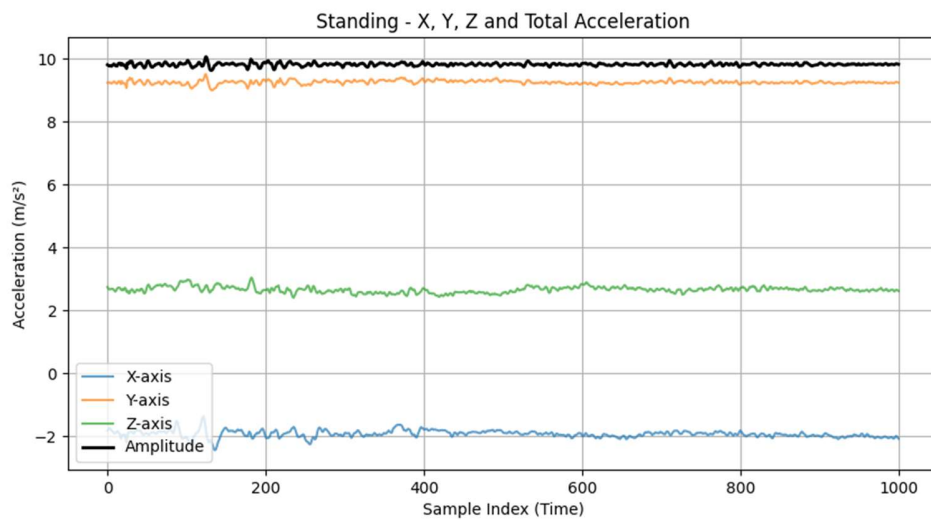
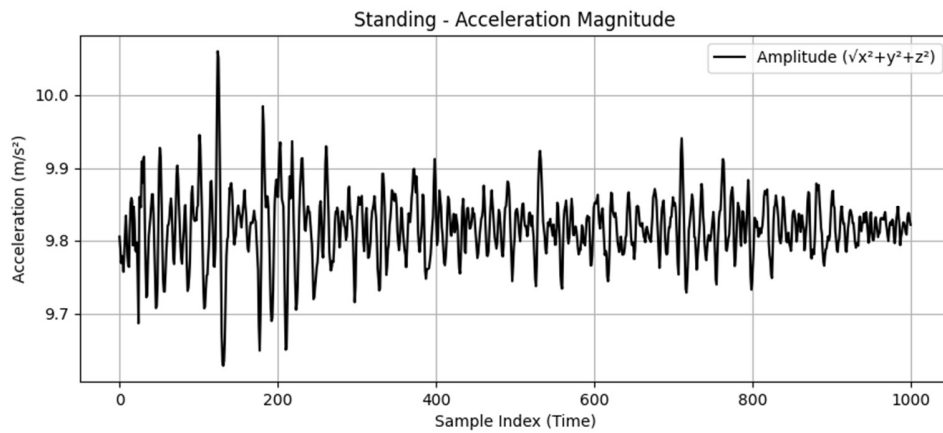
When sitting, the acceleration data shows nearly a flat line. The magnitude stays constant around  $9.8 \text{ m/s}^2$ , which mirrors the force of gravity. The data shows absence of periodic movement, with stable readings across the X, Y, and Z axes. This suggests the device is stationary and primarily detecting gravitational forces. Compared to standing, sitting has even lower variance and almost no small fluctuations. If we zoom into the combined plot, we can see that even the X, Y, and Z axes remain steady, confirming a fully static posture.



- Flat signal is around 9.8  $\text{m/s}^2$
- No oscillations or repetitive peaks
- Extremely low variance (no movement)

### ***Standing:***

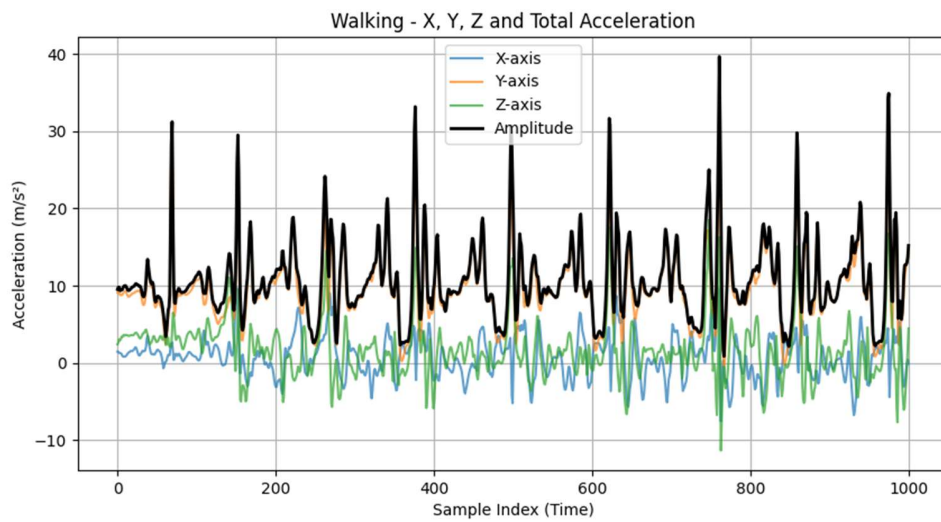
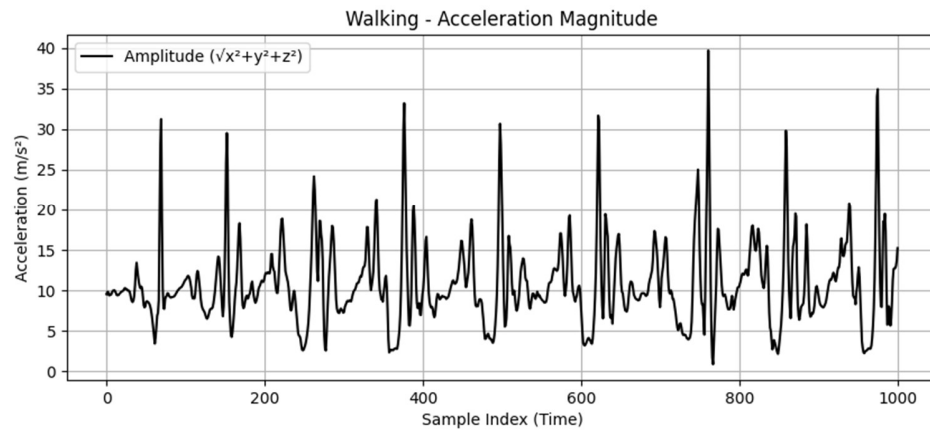
The standing data looks much like the sitting data, but there are small variations in the signal. These small changes happen because even when standing still, the human body makes small adjustments to maintain balance. That is the reason why the X and Y axes show tiny random fluctuations, while the Z-axis stays near the gravity line. When comparing standing and sitting, we can see that standing has slightly higher variance and the signal isn't totally flat.



- Mean acceleration is around 9.8 m/s<sup>2</sup>
- Small irregular fluctuations in X, Y axes
- Slightly higher variance than sitting

### ***Walking:***

The walking plots show a very clear periodic wave pattern. Unlike sitting or standing, the magnitude varies rhythmically, with each peak matching a step. This pattern repeats regularly and has moderate amplitude changes. The combined plot shows all three axes oscillating together, with the Z-axis showing clear up-down movement that reflects vertical body motion. Oscillation frequency exists at a moderate level in walking. Walking, in contrast to static activities, produces rhythmic peaks that can assist ascertain movement patterns.

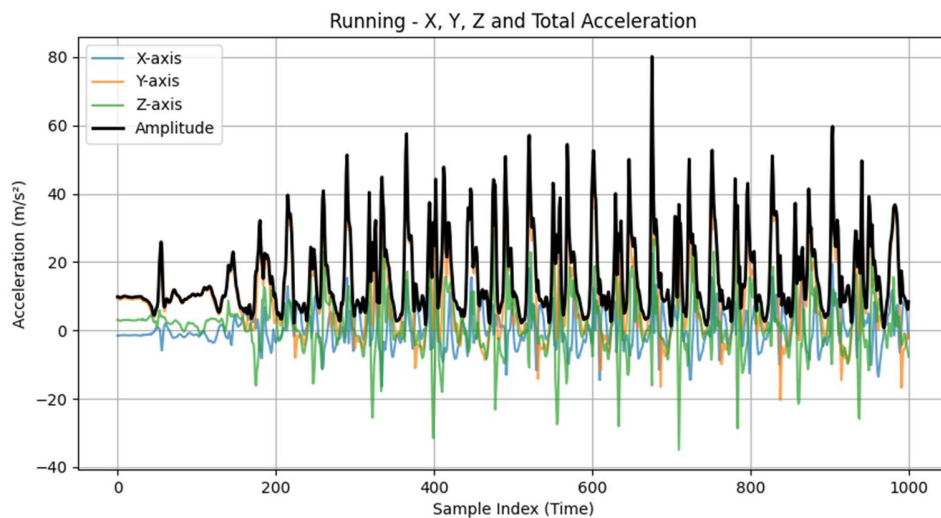
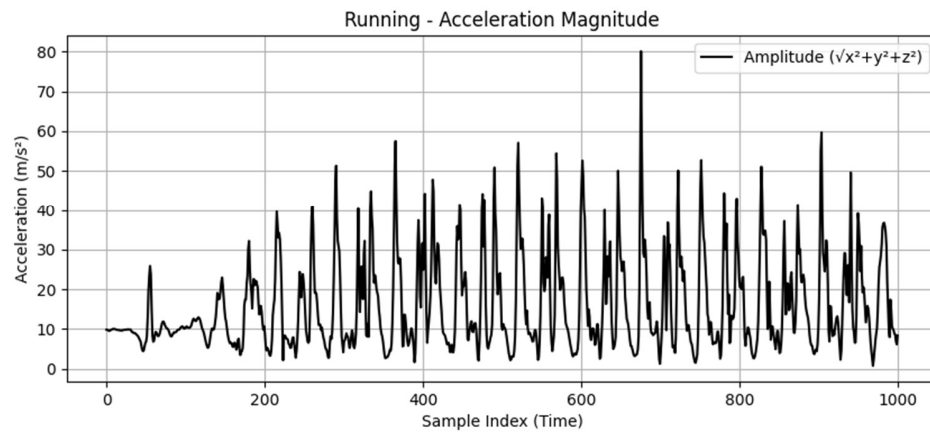


- Regular, periodic peaks in magnitude
- Medium amplitude, around 10–20 m/s<sup>2</sup>
- Moderate frequency and visible rhythm
- Z-axis oscillations correspond to body bounce per step

### ***Running:***

Running shows a very active pattern. The size shows large, frequent peaks, which points to great changes in speed. In the combined plot, all three axes change quickly with high intensity, but the Z-axis is strongest since running has strong up and down motion when the body hits the ground. The peak rate beats walking, showing faster steps and acceleration changes that happen sooner.

The size can jump to 70–80 m/s<sup>2</sup> when impacting, which clearly sets it apart from walking.



- High amplitude and high frequency oscillations
- Very high variance across all axes
- Dense periodic structure (each peak is like a quick stride)
- Dominant Z-axis movement due to vertical bouncing

#### ***Comparisons Across Activities:***

By analyzing all the graphs simultaneously, the very same progression can be observed clearly:

- **Sitting & Standing:** They both are static, however, the standing has slight random variations because of posture correction, whereas sitting is totally stable.
- **Standing & Walking:** Standing does not show any rhythm while walking adds periodic peaks of medium amplitude. Each step generates a recurring wave pattern.

- **Walking & Running:** The pattern of walking becomes denser and more intense in running. The peaks get more taller, meaning higher acceleration, and frequent, showing a faster and stronger motion.
- **Sitting & Running:** These two represent contrasting states: sitting is completely flat where there is no motion, whereas running is chaotic and full of large causing fast oscillations.

### Conclusion:

From the visual inspection of the accelerometer plots, the main distinguishing features between activities are amplitude, frequency, and variance. Static activities, such as sitting and standing, have stable, low-variance signals around  $9.8 \text{ m/s}^2$ , while dynamic ones, such as walking and running, show strong oscillations. Therefore, even by simply looking at the graphs, it's possible to visually recognize and classify each activity based on its acceleration patterns.

## Part 2 – Data Visualization and Feature Inspection:

### Objective:

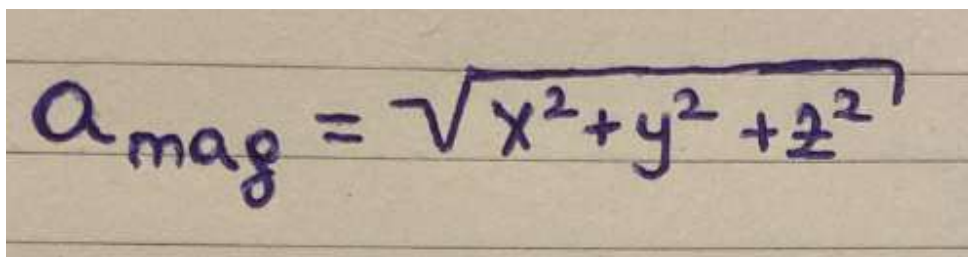
This research aims to count steps from accelerometer data by identifying repeating peaks related to foot motion. The central goal involves examining movement patterns during walking, reducing inaccuracies, and accurately detecting step cycles.

### Command:

`python main.py --file data/walking.csv`

### Method:

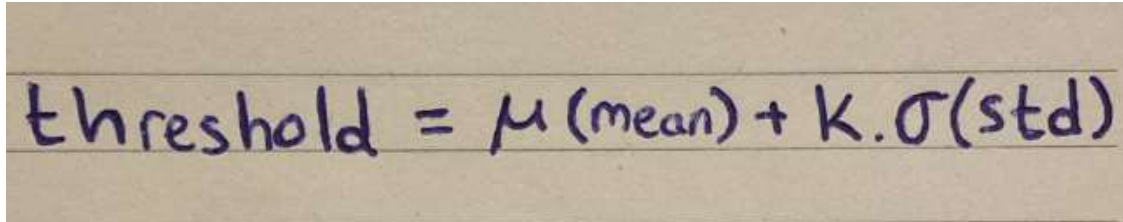
The accelerometer measurements that were taken during walking were passed through a Python pipeline for embedded IoT applications that was specifically developed for such a processing task, and the pipeline was low in terms of resource consumption. The raw data was first loaded via a simple **dataloader** (dataloader.py), which reads the X, Y, and Z acceleration columns and computes the **magnitude**:



$$a_{mag} = \sqrt{x^2 + y^2 + z^2}$$

Since our accelerometer readings contained small fluctuations caused by device vibration and minor hand or pocket motion, the signal was smoothed using a **low-pass filter** after a **high-pass gravity removal** stage (filters.py). This dual-stage filter isolates the main walking pattern while suppressing both the DC component, or gravity in other terms, and high-frequency noise.

Next, a **peak point detection algorithm** was implemented in `peaks.py` to the filtered signal. Each local maximum that exceeded a defined **threshold** was counted as one step. The threshold could be either fixed or automatically determined using:


$$\text{threshold} = \mu(\text{mean}) + k \cdot \sigma(\text{std})$$

where  $k = 0.8$  by default.

A **refractory period** (`min_gap_ms`) of about 350 ms ensured that double peaks within a single stride were not overcounted.

The complete process was integrated in `stepcounter.py` and executed through the `main.py` driver. Our script accepts arguments such as sampling rate, threshold mode, and plotting options, then gives outputs:

- *`outputs/steps.csv`*: Indices and timestamps of detected steps
- *`outputs/steps_detected.png`*: Visualization of filtered signal and detected peaks

This modular pipeline was implemented entirely via **standard Python libraries** such as; **`argparse`, `csv`, `math`, `matplotlib.pyplot`, and `os`.**

### Results:

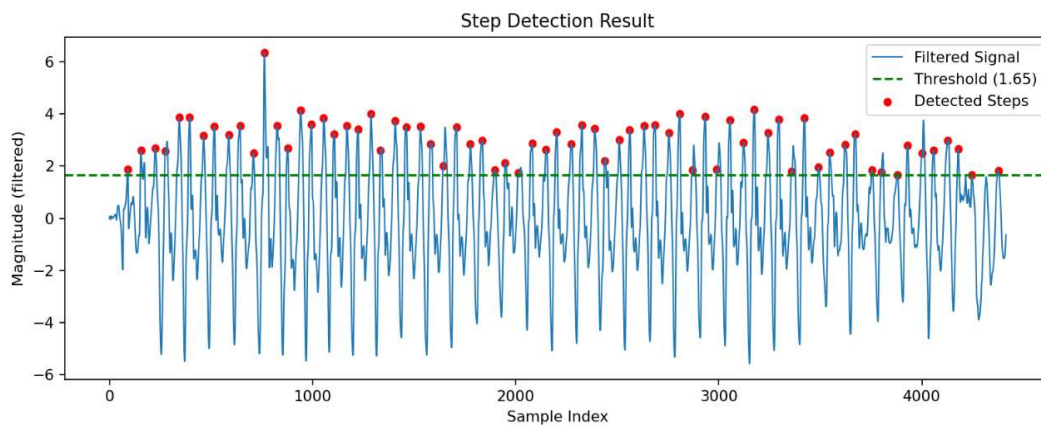
The system successfully detected steps from the walking dataset. Each rise and fall in the **filtered acceleration magnitude** corresponded to a single foot impact. The consistent spacing between the peaks showed the user's walking speed, with shorter gaps for increases in speed and longer ones for slower movement.

A sample output summary:

```
PS C:\Users\VICTUS\Desktop\48007 Ödev\Part2 StepCounter> python main.py --file data/walking.csv
Saved: outputs\steps.csv

***** Step Count Summary *****
File           : data/walking.csv
Sampling Rate (fs): 50 Hz
Threshold used  : 2.59
Steps Detected  : 73
Mean Cadence    : 49.6 steps/min
=====
```

Outputs:



```
Part2 StepCounter > outputs > steps.csv > data
1  step_index,time_s
2  77,1.540
3  153,3.060
4  228,4.560
5  271,5.420
6  290,5.800
7  332,6.640
8  381,7.620
9  452,9.040
10 502,10.040
11 576,11.520
12 631,12.620
13 652,13.040
14 696,13.920
15 751,15.020
16 824,16.480
17 867,17.340
18 939,18.780
19 983,19.660
20 1044,20.880
21 1093,21.860
22 1159,23.180
23 1209,24.180
```

The resulting figure (steps\_detected.png) shows red markers on detected peaks over the smoothed magnitude curve, confirming that each peak aligns with a stride.

### Conclusion:

Part 2 demonstrated an effective **peak-based step-counting algorithm** using accelerometer magnitude. The combination of simple filtering, adaptive thresholding, and peak detection allowed the system to accurately and efficiently estimate the number of steps in real-time, thus making it suitable for embedded IoT environments.



### Part 3 – Pose Estimation with Accelerometer and Gyroscope:

#### Objective:

To determine the device's 3D orientation (roll, pitch, and yaw) while in motion by means of the data from the accelerometer and gyroscope sensors together. The goal is to fuse these two data sources for getting a gentle and precise depiction of the device's position throughout the time.

#### Command:

```
python main.py --file data/mergedWalk.csv --fs 100 --gyro-unit rad --alpha 0.98
```

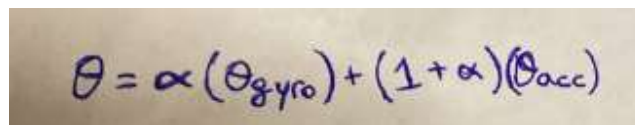
#### Method:

In this part, both **Accelerometer.csv** and **Gyroscope.csv** were used to estimate the sensor's orientation, while **mergedWalk.csv** contained the synchronized version of these readings. The data was loaded through the `dataloader.py` module, which imports time-stamped acceleration and angular velocity values for further processing.

Since raw IMU measurements are typically affected by noise and drift, the `filter.py` module performs two key operations:

- A **low-pass filter** is applied to smooth accelerometer readings and remove high-frequency noise.
- A **high-pass filter** is applied to the gyroscope data to compensate for long-term drift.

The accelerometer serves as a long-term stable reference by referring to gravity, and on the other hand, the gyroscope detects short-term angular changes with precise temporal resolution. To combine these two signals, we used a complementary filtering algorithm implemented in `poseEstimator.py`, expressed as:


$$\theta = \alpha(\theta_{gyro}) + (1 + \alpha)(\theta_{acc})$$

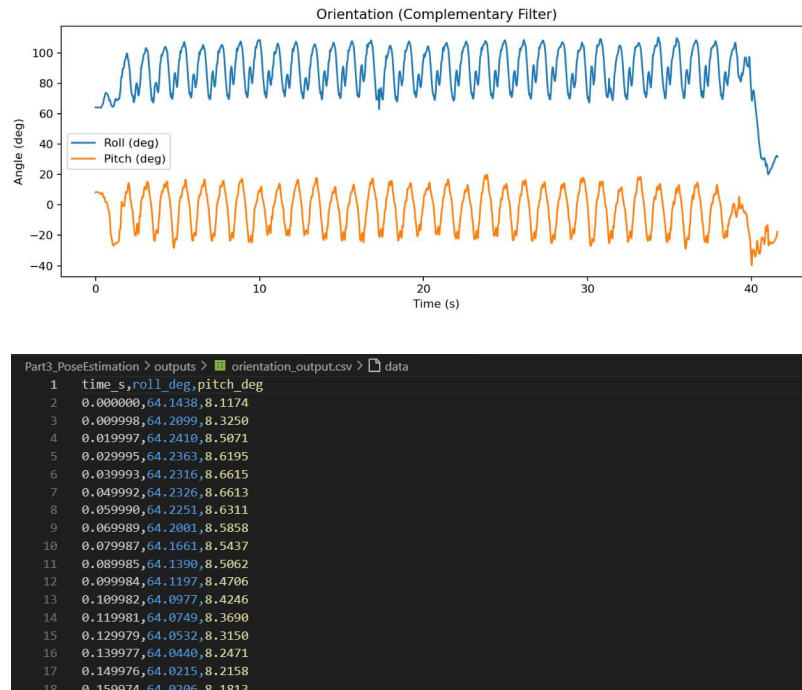
where  $\alpha$  is a weighting factor (commonly around 0.98). This blending retains the responsiveness of the gyroscope while maintaining the stability of the accelerometer.

The `mergeData.py` script on the other hand, was designed to **combine accelerometer and gyroscope readings** into one synced dataset, which is `mergedWalk.csv`. Even though both sensors were recorded simultaneously, their data were saved in separate files to reduce noise. If a new dataset is going to be used in our codes, the merging process can be done with the command: **python modules/merge\_imu.py**.

#### Results:

The filter successfully tracked orientation changes with smooth transitions. When the device tilted forward, the pitch angle increased proportionally; when rotated sideways, the roll angle followed accordingly. The yaw angle remained stable and represented rotational motion around the vertical axis.

The **pose plot** displayed continuous, noise-free curves, while the **3D animation** showed realistic orientation behavior synchronized with recorded movements:



The figure depicts the estimated roll and pitch angles obtained through a complementary filter that combines accelerometer and gyroscope data. The roll represents the sideways tilt, while the pitch shows the forward–backward motion of the device.

Both curves show smooth and periodic oscillations for about 40 seconds which reflect the rhythmic body movement during walking. Every oscillation corresponds to a single step as the phone moves in the pocket. The roll angle stays around 65 degrees, indicating a stable sideways tilt, while the pitch varies between  $-40$  and  $+10$  degrees, matching the natural forward swing of each stride.

The sharp drop near the end marks the stop of movement and recording, confirming realistic motion tracking. Overall, the complementary filter provided stable and drift-free orientation estimation, accurately capturing the device's pose without the noise seen in raw IMU signals.

**Extra:** To complement the numerical pose estimation outputs, two experimental 3D visualization videos were generated:

- **pose\_animation.mp4:** shows a 3D coordinate frame representing the device's orientation changes over time.

**Link:** [https://drive.google.com/file/d/1o5XfhFY7EC1f9sqGGolFjgBQki3KLVH1/view?usp=drive\\_link](https://drive.google.com/file/d/1o5XfhFY7EC1f9sqGGolFjgBQki3KLVH1/view?usp=drive_link)

- **pose\_human\_animation.mp4**: displays a simplified human-body model synchronized with the estimated roll, pitch, and yaw angles.

**Link:** [https://drive.google.com/file/d/1hkkAtpc-g8JGnY1tZfutBhJKKY2XeFPf/view?usp=drive\\_link](https://drive.google.com/file/d/1hkkAtpc-g8JGnY1tZfutBhJKKY2XeFPf/view?usp=drive_link)

These visualizations were developed *exclusively for demonstration purposes*. They illustrate how the estimated orientation evolves dynamically during walking and small tilting motions. Both animations visually confirm the accuracy of the complementary filter, showing smooth and realistic motion consistent with the recorded IMU data.

### **Conclusion:**

The system figured out the sensor's roll, pitch, and yaw by mixing accelerometer and gyroscope data using some clever filtering. This way worked well for tracking pose in the short term and kept noise down. So, it's good for motion-sensing stuff in IoT devices and could be a base for better orientation-tracking methods.