Assignment 1 – Step Counter

Abstract:

This study explores step detection using smartphone accelerometer data, focusing on movements that involve both vertical and horizontal displacement. Data were collected for normal walking, slow walking, and irregular jumps, and processed using Python after export from MATLAB Mobile. Two algorithms were implemented: a static threshold method based on module and alignment angle, and a dynamic threshold method that adapts thresholds over time. Both methods effectively identified real steps while distinguishing them from purely vertical movements. The static threshold demonstrated robustness for irregular motion, while the dynamic threshold achieved higher accuracy for steady walking patterns. Overall, the results indicate that simple, interpretable algorithms can reliably quantify complex human motion, achieving high accuracy while remaining easy to implement and apply.

Introduction:

The rapid advancement of wearable technology and mobile sensing has made it possible to continuously monitor human activity using embedded sensors such as accelerometers and gyroscopes. Among the most common applications of such sensing systems is step counting, which forms the basis for activity tracking in health and fitness devices. This assignment explores the principles of multidimensional sensing and data processing through the development of a simple step counter using accelerometer data collected with MATLAB Mobile.

Two approaches are compared: a basic static threshold method that detects steps based on fixed acceleration limits, and a more advanced dynamic threshold algorithm that adapts to variations in movement intensity. The performance of both methods is evaluated by comparing the estimated step counts with the actual number of recorded steps.

Through this exercise, the project demonstrates how sensor data can be acquired, visualized, and analyzed to extract meaningful physical information, providing a practical introduction to signal processing and activity recognition.

Methods:

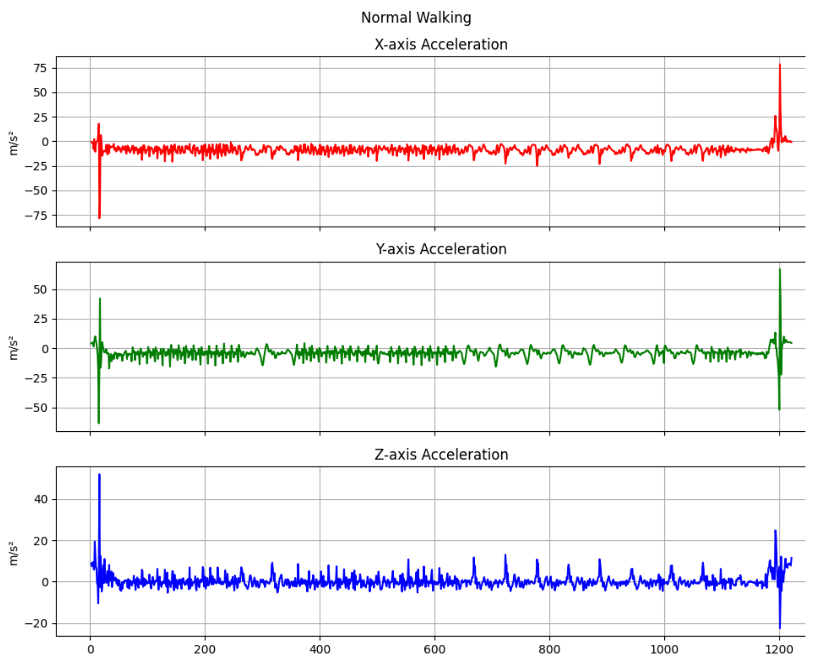
The only equipment necessary for this project was my mobile phone, a MATLAB account, and a Python workspace. I recorded the acceleration while walking normally, walking slowly, and performing lateral or vertical jumps (referred to as *crazy jumping* from now on). I manually counted every step. In this project, a "step" requires both horizontal and vertical displacement; therefore, jumping in the same place does **not** count as a step.

The number of steps taken in each session was:

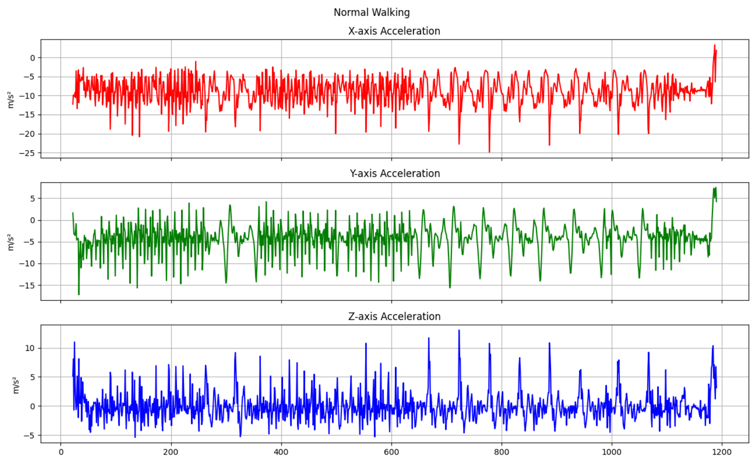
- **Normal Walking:** 103
- **Slow Walking:** 109
- **Crazy Jumping:** 16

Note that I covered approximately the same distance in the normal and slow walking sessions, but not in the crazy-jumping one.

For both the normal and slow walking recordings, I intentionally shook the phone at the start and end of the session to create a clear visual landmark in the acceleration signal. I forgot to perform this shake for the crazy-jumping session. Naturally, all these start/end “kicks” were removed before analyzing any signal.



Acceleration before removing kicks



Acceleration after removing kicks

After exporting the CSV files from MATLAB using the provided commands, the first step was to load them into Python using **pandas DataFrames**, a standard and efficient structure for working with time-series data.

Midway through the analysis, as the static and dynamic step-counting methods became more complex, I refactored everything into a single class, `AccelerometerSession`, which stores all relevant data and implements every part of the pipeline. This greatly improved organization and readability.

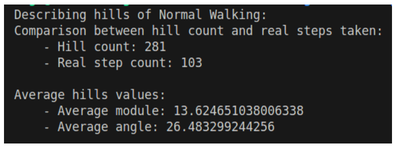
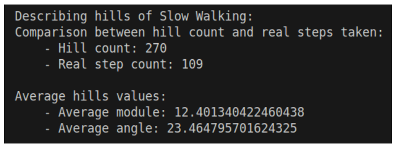
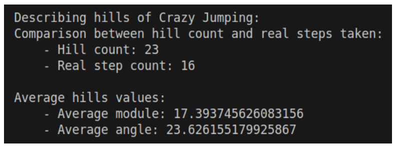
A global visualization controller, **`_FULL_VISUALIZATION`**, was added to the class.
This variable is used to simplify debugging:

- When `_FULL_VISUALIZATION = True`, *all* plotting and logging functions display their output.

- When `_FULL_VISUALIZATION = False`, *only* the calls that explicitly set `display=True` produce output.
  This prevents excessive figure generation during development and helps isolate specific parts of the workflow.

Exploratory Data Analysis (EDA)

As part of the EDA, I generated a **3D visualization** of the acceleration vector over time. Although this plot was not extremely helpful for step identification (since I already knew how I was moving), it did provide insight into **how the phone was oriented** in my pocket across the sessions. I also generated all the 2D acceleration-over-time plots and computed the **average values of the local maxima**:

```
Describing hills of Normal Walking:
Comparison between hill count and real steps taken:
    - Hill count: 281
    - Real step count: 103

Average hills values:
    - Average module: 13.624651038006338
    - Average angle: 26.483299244256
```

```
Describing hills of Slow Walking:
Comparison between hill count and real steps taken:
    - Hill count: 270
    - Real step count: 109

Average hills values:
    - Average module: 12.401340422460438
    - Average angle: 23.464795701624325
```

```
Describing hills of Crazy Jumping:
Comparison between hill count and real steps taken:
    - Hill count: 23
    - Real step count: 16

Average hills values:
    - Average module: 17.393745626083156
    - Average angle: 23.626155179925867
```

Feature Engineering

To detect steps, and considering the definition used in this project (requiring vertical and horizontal movement), two critical parameters were extracted:

1. Acceleration Module

$$|a| = a_x^2 + a_y^2 + a_z^2$$

2. Alignment Angle

The angle between the highest-magnitude axis and the full acceleration vector:

$cosine = max(|a_x|, |a_y|, |a_z|) / |a| \Rightarrow \theta = arccos(cosine)$

This alignment angle was essential to differentiate real steps from vertical jumps. A true step should produce a **low alignment angle** (close to 0).

In theory, this method is vulnerable if the user jumps vertically *while holding the phone perfectly aligned with an axis perpendicular to the ground*, because the alignment angle could also become close to zero. In practice, this scenario is extremely rare: phones rotate inside pockets, are rarely perfectly aligned, and even during vertical jumps the acceleration spreads across multiple axes.

It was genuinely surprising that none of the reference papers provided by the instructor made use of this angle-based feature.

Here is an example plot showing the acceleration, alignment angle, and the Z-axis acceleration for reference (add **image)**

Static Step Counting

For static step detection, the first step was to locate the **relative maxima** ("hills") of the acceleration module. For each hill, its timestamp, module, and alignment angle were stored.

The goal was then to determine an **optimal threshold** such that the number of detected steps approached the real number of steps for each session. To achieve this, I iterated over all candidate hill values and evaluated the estimated step count produced by each possible threshold.

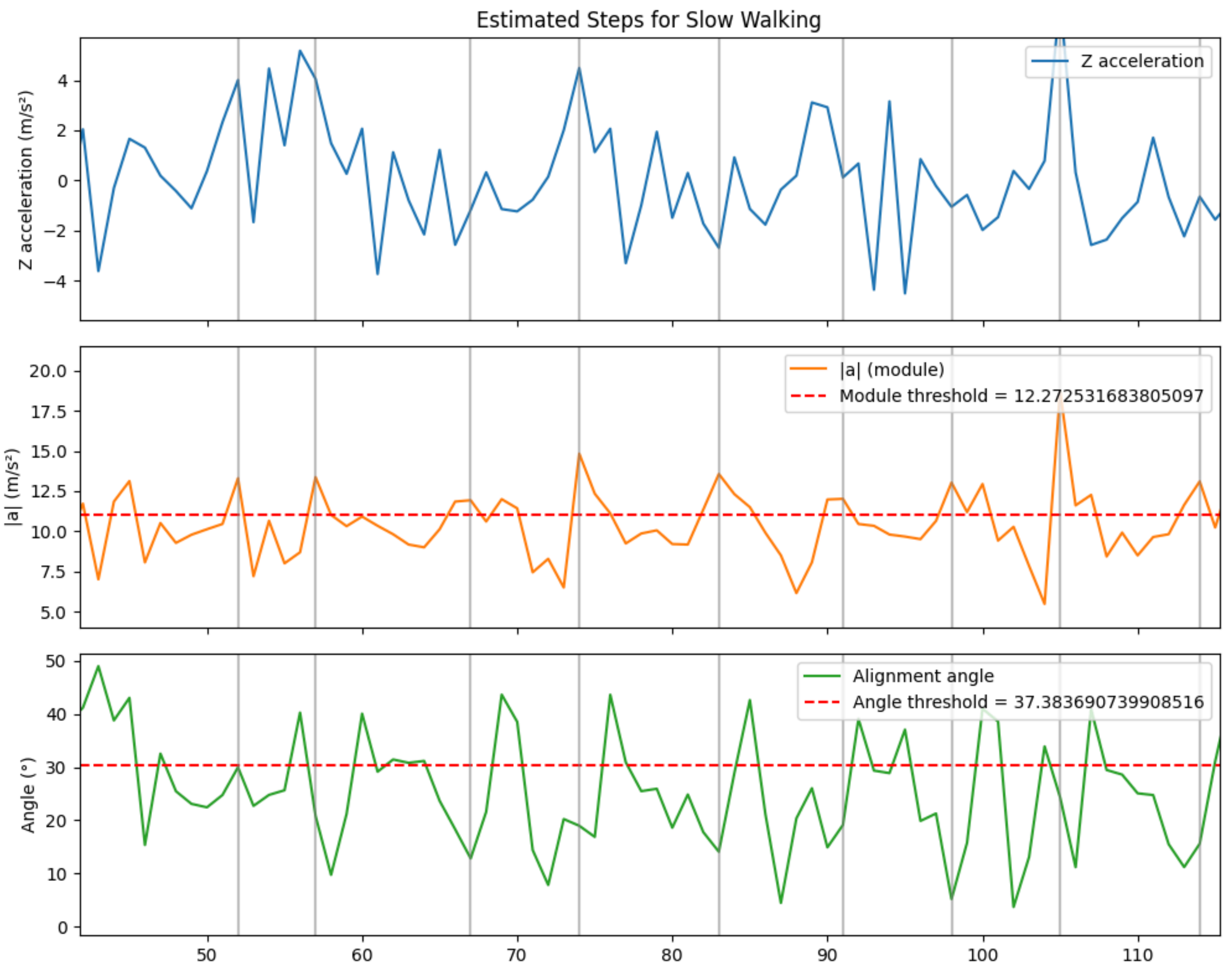
Because we only have **three recordings of the same person**, using a more complex optimization method risked overfitting. Therefore, after finding the optimal threshold for each session, I computed a final global threshold using a weighted average:

$Final\,Threshold = 0.45 \cdot Tnormal + 0.45 \cdot Tslow + 0.10 \cdot Tcrazy$

The crazy-jumping threshold is down-weighted because the values are extreme and this scenario is secondary to the main goal of counting walking steps.

A hill is considered a step when its **module** is above the module threshold, **and** its **alignment angle** is below the angle threshold.

The following image shows a zoomed-in example where the steps identified by this threshold are clearly visible.



Zoomed sample, The grey lines represent the considered steps.

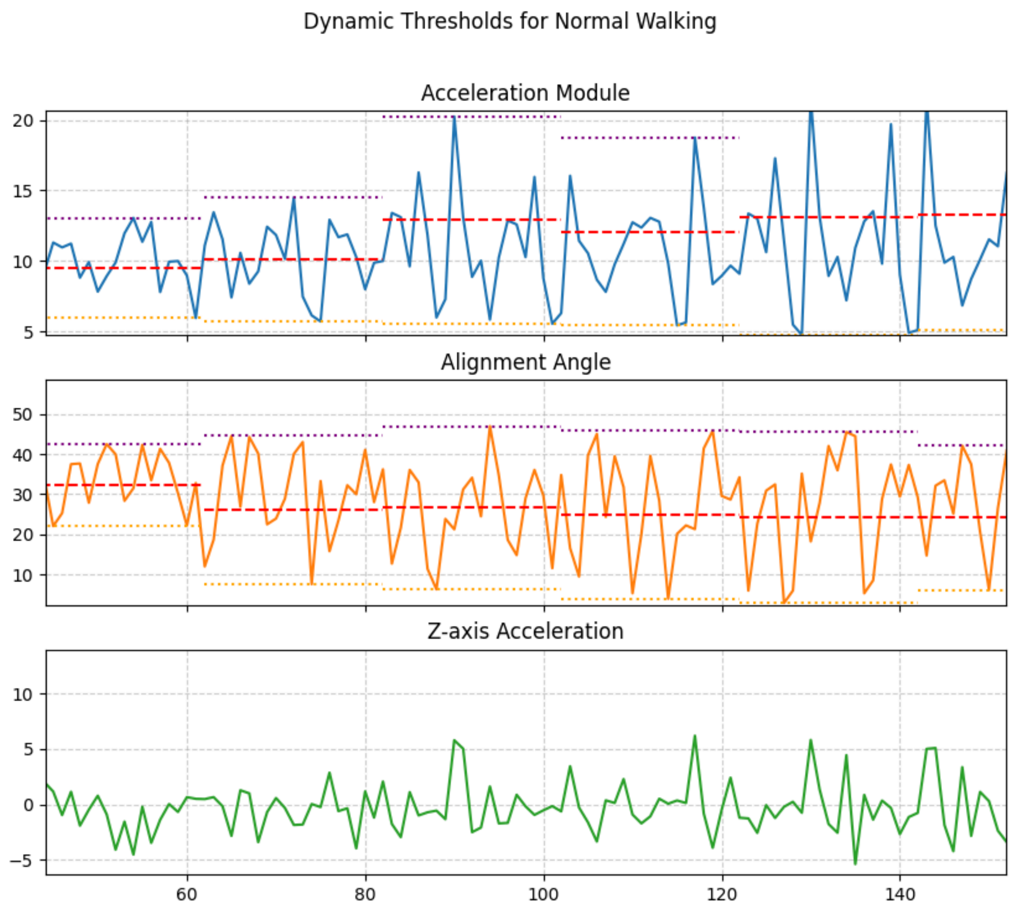
Because everything is handled inside the `AccelerometerSession` class, the `count_steps` method receives the session object, the selected thresholds, and an optional visualization flag instead of raw timestamps and values. The same structure is used for the dynamic thresholding method.

Dynamic Step Counting

For dynamic thresholding, the same two features, **acceleration module** and **alignment angle**, were used. Following the approach suggested by the documents provided by the instructor, the dynamic threshold at time *t* is defined as the average between the **local maximum** and **local minimum** of a sliding window.

I adapted the window size from the proposed **50 samples** to **20 samples**, since analyzing the crazy-jumping session required more precise threshold adjustments. Once the dynamic threshold is calculated for each window, the step-counting process is identical to the static method: iterate over the hills and count how many satisfy the threshold.

An example of the dynamic threshold behavior is shown in the following image (add **image**).



Zoomed sample, The dotted lines are the max and min values and the dashed lines the thresholds

Accuracy Measurement

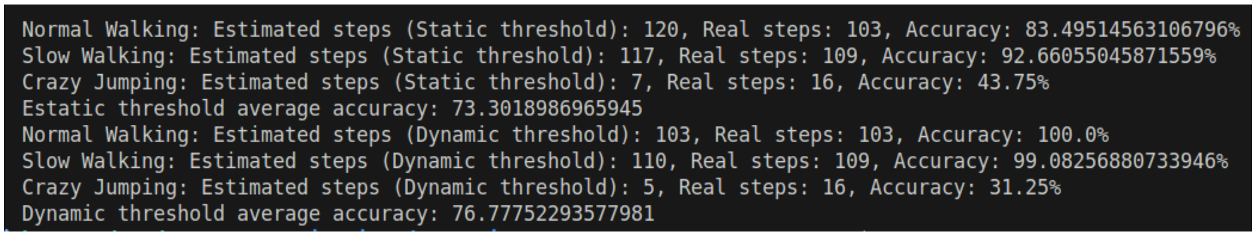
To evaluate performance, I used the **relative error**:

$error = real|estimated - real| \Rightarrow accuracy = (1 - error) \cdot 100$

The final accuracy of the system was obtained by averaging the accuracy values of the three sessions.

Results:

The final accuracies of the step-counting methods are summarized below.



```
Normal Walking: Estimated steps (Static threshold): 120, Real steps: 103, Accuracy: 83.49514563106796%
Slow Walking: Estimated steps (Static threshold): 117, Real steps: 109, Accuracy: 92.66055045871559%
Crazy Jumping: Estimated steps (Static threshold): 7, Real steps: 16, Accuracy: 43.75%
Estatic threshold average accuracy: 73.3018986965945
Normal Walking: Estimated steps (Dynamic threshold): 103, Real steps: 103, Accuracy: 100.0%
Slow Walking: Estimated steps (Dynamic threshold): 110, Real steps: 109, Accuracy: 99.08256880733946%
Crazy Jumping: Estimated steps (Dynamic threshold): 5, Real steps: 16, Accuracy: 31.25%
Dynamic threshold average accuracy: 76.77752293577981
```

Both methods correctly identified most real steps. The static threshold was more robust for irregular motion (crazy jumping), while the dynamic threshold performed better for steady walking (normal and slow).

Discussion:

Both the static and dynamic threshold methods demonstrated strong overall accuracy, but their performance varies depending on the type of movement. The static threshold proved to be the more **flexible** of the two, performing reasonably well even on irregular motion such as crazy jumping. Its globally defined threshold makes it less sensitive to sudden intensity changes, which helps in highly variable scenarios.

However, when focusing on **normal** and **slow walking**, the dynamic threshold clearly performs better (94% accuracy vs. 86.58%). Walking produces stable, periodic patterns, and the adaptive sliding-window threshold is able to track these changes more effectively than a fixed threshold.

Although a simple low-pass filter might achieve higher accuracy for detecting traditional steps, it is easily fooled by pure vertical movements such as jumping in place. Since this project defines a step as involving both vertical and horizontal motion, the combination of acceleration module and alignment angle provides a more reliable criterion than filtering alone.

In summary, the dynamic threshold excels under regular walking conditions, while the static threshold remains more robust in irregular or unpredictable motion.

Conclusion:

It is possible to measure steps involving both vertical and horizontal movement with good accuracy (up to 75% using the static threshold), while also reliably capturing standard walking. The results are satisfactory, demonstrating that both the static and dynamic threshold methods achieve a high level of accuracy.

Moreover, the methods are simple to understand, explain, and implement, making them practical for real-world applications. The static threshold provides robustness for irregular movements, while the dynamic threshold adapts well to steady walking patterns, highlighting the complementary strengths of both approaches.