Student Name: Carlos Roberto Cueto Zumaya

Student Number: 2203394

Repository: https://github.com/it-teaching-abo-akademi/assignment-1-crcz25

Assignment 1 – Step Counter

Brief Code Explanation

The script has multiple parameters needed to run. These parameters were added to the script to make it more flexible and to allow the user to change them without modifying code. The parameters are:

- *Path_to_file*: is the path to the file containing the data. The file should be inside a directory called data in the same directory as the script.
- *Algorithm*: is the algorithm used to calculate the step count. It can be either "simple" or "advanced".
- *Window_size*: is the size of the window used to calculate the step count.
- *Precision_threshold*: is the threshold used to determine if a change in acceleration is valid or not.

The simple algorithm is based on the article written by Zhao (2010); likewise, the advanced algorithm is based on the article written by Daskalov (n.d.). In the sections below, I explain each more in depth and how I use the corresponding functions.

Simple Algorithm

The function *count_steps* is called when the simple algorithm is selected. It takes in four arrays of data (timestapms, x_arr, y_arr, and z_arr). The function calculates the magnitude of the total acceleration from the 3 axes.

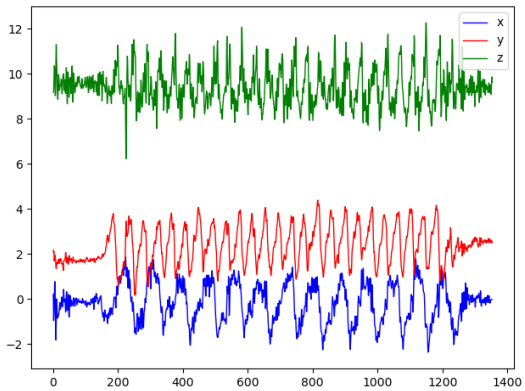

Figure 1 Raw data from accelerometer.

This function then iterates over the magnitude array based on a predefined window size (received as a parameter) and calculates the maximum and minimum acceleration in the window. It then calculates the corresponding threshold for the window as the average between the maximum and minimum.

As explained by Zhao (2010) a linear-shift-register is used in conjunction with the dynamic threshold to decide whether a step has been taken or not, this removes the high-frequency noise to make the decision more precise. Next, the function checks if the change in acceleration is greater than a predefined precision threshold, if that is the case, the sample is shifted. A step then is defined as happening if there is a negative

slope of the acceleration and when it crosses below the dynamic threshold. If a step is detected, then the timestamp is saved into the array *step_times*. Finally, the threshold, maximum, and minimum values for visualization purposes, see Figure 2.

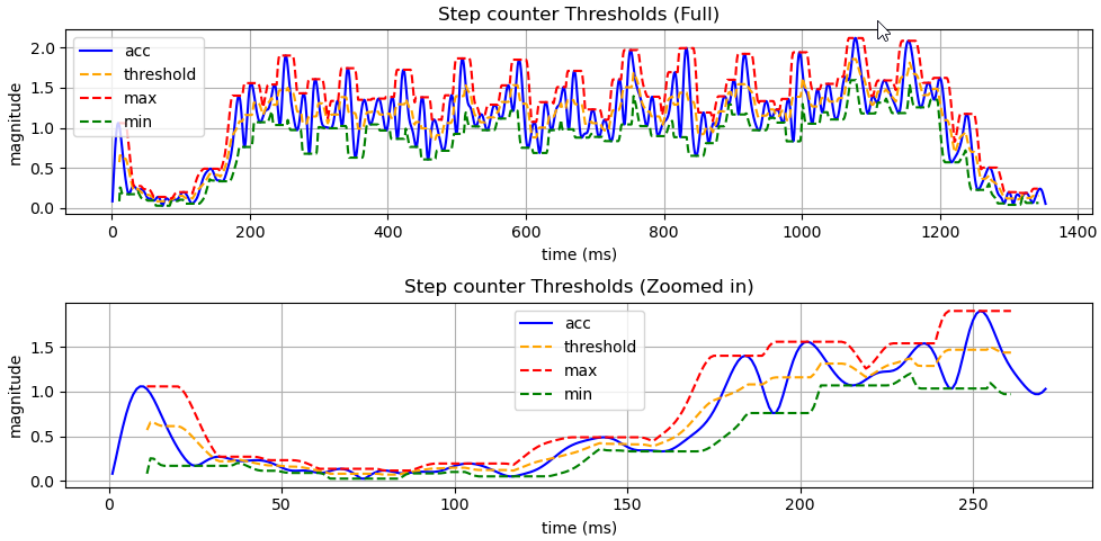

Figure 2. Dynamic Threshold.

Advanced Algorithm

The function *count_steps_advanced* is the advanced version of the simple algorithm and *count_steps*. This function applies several signal processing techniques to improve the accuracy of the step detection algorithm by smoothing the signal. The first step is to apply a 1-dimensional Gaussian Filter to each dimension (x, y, z) of the data to "pre-smooth" the signal. This is done using numpy's convolve function and a pre-defined Gaussian kernel, see Figure 3 and Figure 4 below.

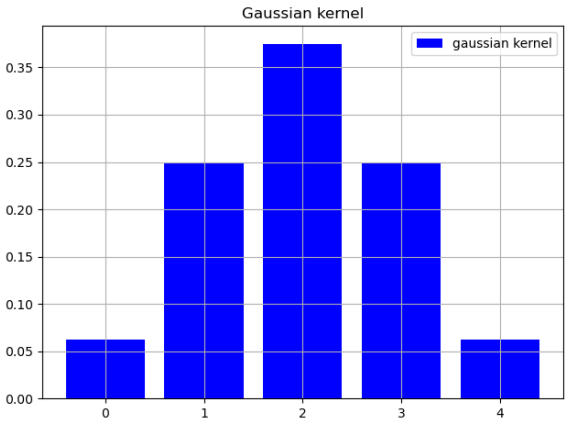

Figure 3 1-D Gaussian Kernel

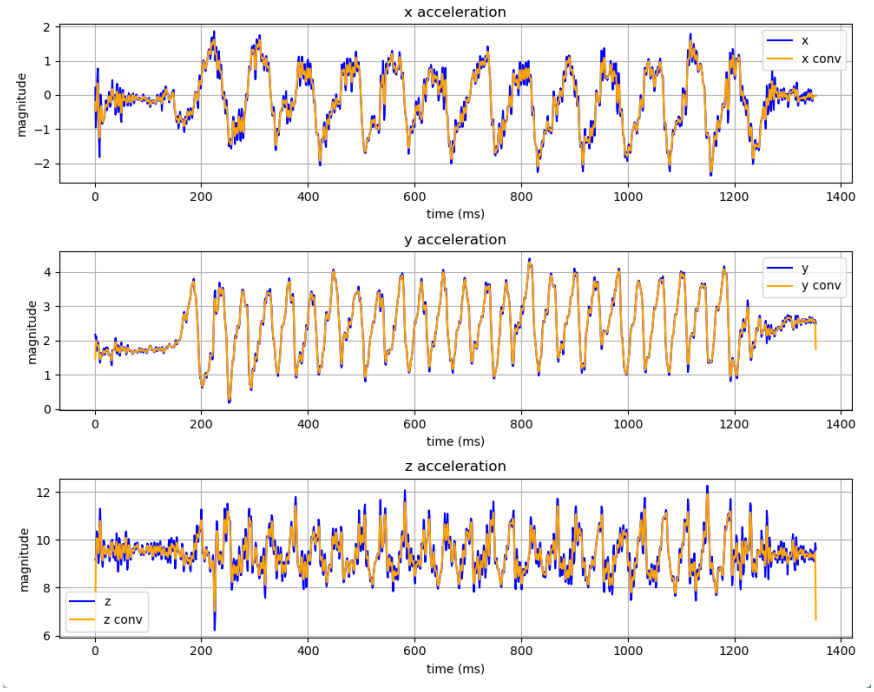

Figure 4. Original signal vs Filtered Signal (Gaussian Kernel).

Next, the total magnitude of the acceleration is calculated, and a low-pass filter is applied to calculate the gravity; this is done using the *butter* and *filtfilt* functions from SciPy. Finally, the user acceleration is then calculated by subtracting the gravity component from the total acceleration. This is possible because gravity is a constant acceleration and has a frequency closer to 0 Hz as seen on Figure 5. Particularly, this signal analysis was difficult to understand for me since it was my first approach to frequencies processing.

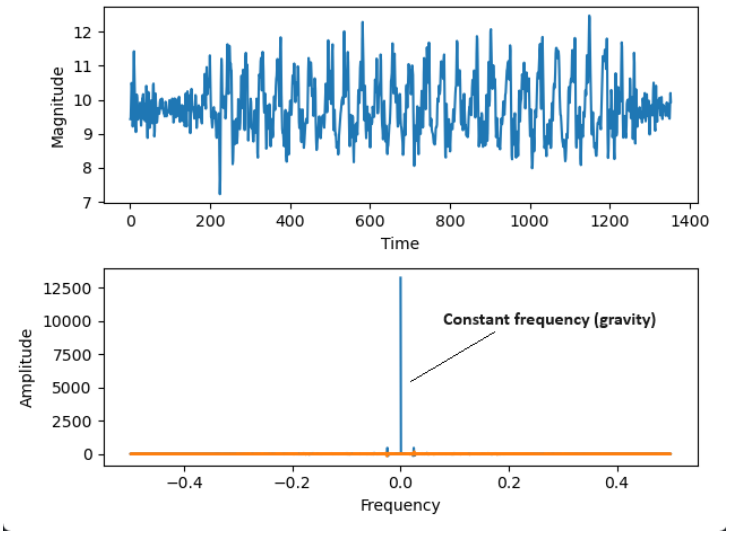

Figure 5. Frequencies contained in the signal.

Likewise, to further smooth the signal, the user acceleration is filtered using a low-pass filter with a cutoff frequency of 5HZ and a high-pass filter with a cutoff of 1 HZ to remove jumpiness and steps respectively as shown in Figure 6.

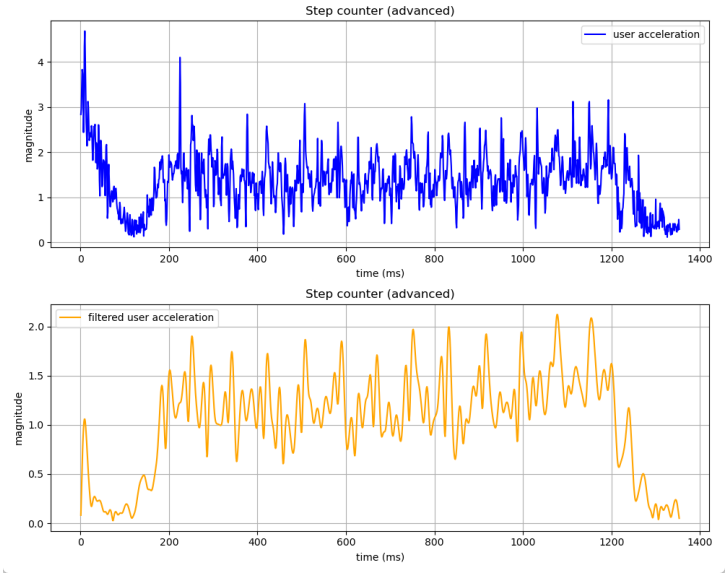

Figure 6. Original signal vs Filtered Signal.

Finally, the steps are calculated and detected using the same function *count_steps* used in the simple algorithm.

Results

15 Irregular Steps

This dataset consists of 15 steps, some of which are taken normal, jumpy, or sloppy.

Number of Steps	Algorithm	Window Size	Precision Threshold	Steps Detected	% Error
15	Simple	20	0.2	53	253.33
15	Advanced	20	0.2	6	60

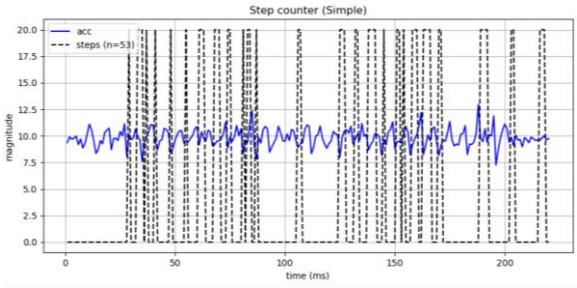

Figure 7 Signal with steps (Simple Algorithm)

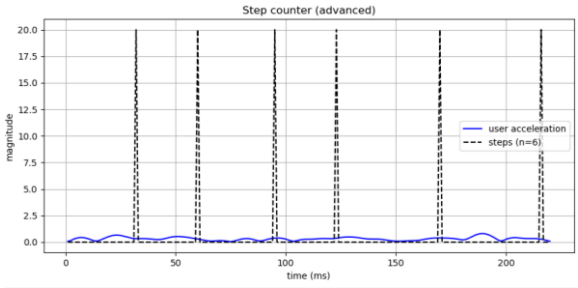

Figure 8 Signal with steps (Adv. Algorithm)

25 Normal Steps

This dataset consists of 25 steps which are normal walking steps.

Number of Steps	Algorithm	Window Size	Precision Threshold	Steps Detected	% Error
25	Simple	20	0.2	131	112
25	Advanced	20	0.2	27	76

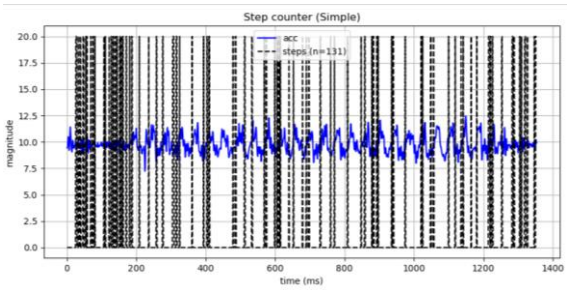

Figure 9 Signal with steps (Simple Algo.)

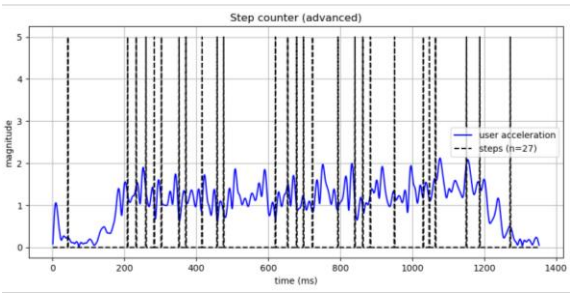

Figure 10 Signal with steps (Adv. Algo.)

100 Fast Steps

This dataset consists of 100 steps with a mix of fast and normal steps, as well as an inclined plane.

Number of Steps	Algorithm	Window Size	Precision Threshold	Steps Detected	% Error
100	Simple	20	0.2	159	59
100	Advanced	20	0.2	83	17

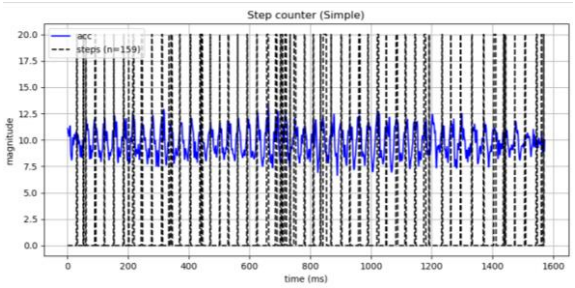

Figure 11 Signal with steps (Simple Algo.)

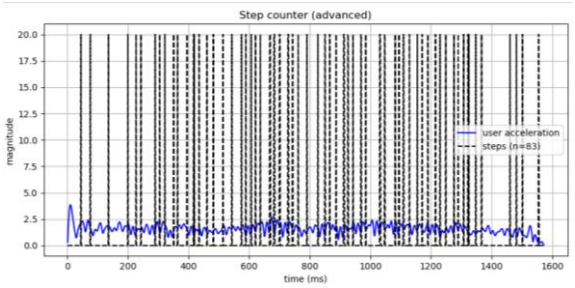

Figure 12 Signal with steps (Adv. Algo.)

Conclusions

Accelerometers are commonly used as pedometers due to their ability to measure acceleration on multiple axis and their availability in our daily lives thanks to smartphones. This lab evaluated the pedometer capability of a smartphone accelerometer. During the realization of it, I learned that this is a powerful sensor that can be used to count the steps of a person, but it is not as simple as it seems. The fact that a step is not a simple movement and is unique to each person makes it difficult to analyze and to universally develop an algorithm.

In practice, the signals obtained from this sensor are subject to noise and other factors that affect the performance and quality of it. Thus, before proceeding with counting, it is important to preprocess the data to remove unwanted elements that may impact the performance of the counting algorithm.

As explored by Ryan et al. (2021) there are multiple approaches to step detection, such as peak detection, threshold crossing, and autocorrelation, each with their pros and cons. In this lab, I used a combination between peak detection and thresholds crossing with windowing, as well as frequency filtering to smooth the signal. As shown in the results, the preprocessing step is crucial to obtaining good performance. The results indicated that the error between the simple algorithm and the advanced algorithm was reduced on average 90%.

Future work could include creating datasets with different types of steps and walking patterns to train a machine learning model that can be used to count steps. This approach would be more robust as it would be able to adapt to different types of steps and walking patterns as long as there is enough data to train with. In addition, it would be interesting to evaluate the performance of the algorithm in different scenarios, such as walking on a treadmill, walking on a flat surface, walking on a slope, on stairs, etc.

If the goal is to make a one-purpose device for step counting, it would be interesting to see how the algorithm works with a dedicated accelerometer and compare it to the performance of the smartphone accelerometer. Lastly, I would suggest creating a better test methodology to find the “best parameters” for the threshold and windows size iterating with different values involving multiple layers of cross-validation to provide a more robust evaluation.

References

Daskalov, D. (n.d.). *5000 Lines or Less Apedometer in the Real World*. Retrieved from https://aosabook.org/en/500L/a-pedometer-in-the-real-world.html

Mattfeld R, J. E. (2021). Pedometer Algorithms on Semi-Regular and Unstructured Gaits. In *Sensors.* Basel, Switzerland. doi:https://doi.org/10.3390/s21134260

Zhao, N. (2010). Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer. *Analog Dialogue*.