Åbo Akademi

Multidimensional Sensing Techniques

Assignment N°1: Step Counter

Cédric Léonard, 29/09/2021, student ID: 2100775

Introduction

This report contains information about the Assignment N°1 in Multidimensional Sensing Techniques. You can find additional informational on the following GitHub repository.

The goal of this assignment was to familiarize with reading data from sensors as well as visualizing and parsing data on a concrete example. MatLab Mobile, MatLab and Python languages were used to record, read and parse data and implement two different algorithms to count the number of physical steps recorded in the data.

Among the additional information, the scale of each figure is in m/s^2. Through this experiment we assume that each dataset corresponds to a person walking, running or climbing stairs, but keeping its phone in the same direction. Moreover, choice was made to count every step (left and right legs) and not only right leg steps for example.

What you did?

In order to set up the lab material I first installed MatLab Mobile on my phone and MatLab on my computer. Then I discovered and played with MatLab mobile in order to record different data sets (slow walking, fast walking, climbing stairs, etc.) with my phone accelerometers. Then, I picked up these datasets from the MatLab cloud and I parsed these **.mat** files into **.csv** files with a short MatLab script (furnished in the Assignment1_Instructions.pdf: **importData.m**).

Then, I spent time reading and understanding the python script furnished on Moodle. Next step was to write the code to import the data. Subsequently I thought of an "easy" solution to detect steps, with a fixed threshold and implemented that solution. And finally, I upgraded this solution with a dynamic threshold and therefore created my second algorithm to detect physical steps in my data.

How and why you did it?

I won't go through the "How I generated my datasets with MatLab Mobile and **importData.m**" because all information was provided in the Assignment1_instructions.pdf.

Reading and parsing data

The first interesting thing I did was to write the code responsible for data importation into the **stepcounter.py** script. This is really simple code which simply goes through the `.csv` file, read each line and parse them. As each line corresponds to a single timestamp, and each line contains its "number" (the timestamp) and 3 values for the X, Y and Z axes accelerations each value separated by a ',' it was quite simple. A simple for loop until the end of the line and the use of Numpy `split()` function did the job. This `read_data()` function then returns the timestamps and 3 arrays for each axis.

First algorithm: Fixed threshold

The next step was the implementation of the first algorithm. Now that I have my data, in 3 arrays, I want to determine which array I will use. A simple idea is to identify the axis subject to gravity. As the user can place his phone whatever he wants, this axis may change over the datasets. I verified my code was functional during my tests, but for a jean pocket reason the real sample were all realized with my phone head down in my pocket and therefore I have always worked with data on the Y axis.

Identifying the axis subject to gravity is quite easy, I compute the average of each axis acceleration over the whole sample, 2 averages should be around 0m/s² and 1 should be around 10m/s² due to the 10m/s² earth gravity.

After selecting the corresponding array, we want to identify significant changes over time in the data. This can be done with a threshold; we simply look whenever the data cross this threshold. We could stop there and just note the corresponding timestamp, but with this method I realized I often denote 2 steps instead of 1 because of the positive slope after step. So, I introduced a timing criterion to fix this problem.

The idea is the following: if we detect a significant change in the value, we check if the last step was really recent. If that's the case we assume that we are considering the same step than before and don't record it. This is done with the `minimum_time_between_steps` variable which, as for the threshold, is fixed and imposed.

The final fixed threshold algorithm uses a `threshold` of 13 and a `minimum_time_between_steps` of 10 (for a 10Hz sensor frequency this almost means 1 second between each step).

Second algorithm: dynamic threshold

For this second algorithm we start in the same way than the first one: we want to identify which axis is sensible to the gravity in order to analyze its data. Then, we start over from the first algorithm code, but add a periodic threshold update, which makes this method a dynamic threshold step detection algorithm. Therefore, a new threshold is computed for a specific time window, in my code I chose to create a 50 samples time window but it could have been wider. This new computation implies to check if the time window does not go out of bounds of our array.

The dynamic threshold is computed as a percentage between the min and max value inside the time window. I tried different percentages, but the best I found was 65% min and 35% min (for a positive threshold which means a phone head up).

In order to improve the results of this algorithm I also added a check on the threshold viability. This allows to not detect steps when the threshold is absurd, for example, at the beginning of the dataset when there are no steps and the noise is enough to detect false steps with a threshold too close to 10m/s². This viability check sets the threshold to an inaccessible value (default: 100m/s² or -100m/s²) if the computed threshold is below 11m/s² or -11m/s².

The results of your work

First solution (fixed threshold)

Results are rather satisfying. The simplest dataset **Slow_walking.csv** contains 22 steps and this method detects 23 steps. After analyzing the output, a last step is recorded when my phone goes out of my pocket. The other dataset: **Fast_walking.csv** contains 28 steps and 30 steps are detected. Once again there is 2 "fake steps" when the phone goes in and out of my pocket.

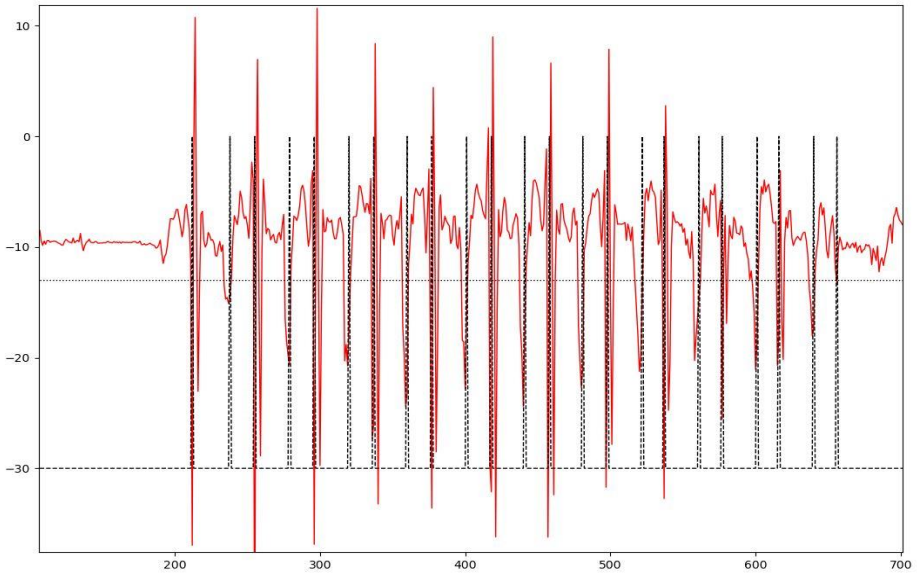
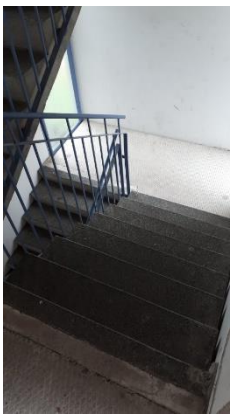


Figure N°1: Y axis acceleration (red), steps detected (dashed black) and fixed threshold (-13m/s²) output on the **Slow_walking.csv** dataset.



Picture N°1: Stairs.

Finally, I tried another experiment: climbing stairs. I wasn't expecting anything from this method but I have been pleasantly surprised. The **Stairs.csv** dataset contains 32 stairs step and the algorithm counts 37 steps. When I tried to understand why and to analyze the output curves, I realized I have done a step between each of the 4 small stairs. Indeed, the record was not realized in a 32 steps straightforward staircase but on an almost "spiral" staircase made of 4

small stairs each containing 8 steps. And with 4 stairs, each 8 steps long plus my 3 additional steps in order to switch staircase we end up with 35 steps which is quite close of the 37 steps counted by the program.

Second solution (dynamic threshold)

If the results are still correct, they are still below first method result. Considering the time spend developing each method it is normal but still surprising as we talk about a method improvement.

The simplest dataset: **Slow_walking.csv** contains 22 steps and the dynamic threshold method detects 20 steps. The other dataset: **Fast_walking.csv** contains 28 steps and 24 steps are detected. Only the last dataset is better: **Stairs.csv** contains 35 steps and 35 are detected.

We observe an under evaluation of steps number, this is due to the dynamic threshold computation and the fact that we are trying to detect both legs steps. During my datasets recording my phone was located in my right leg pocket, therefore left leg steps result with lower signal picks.

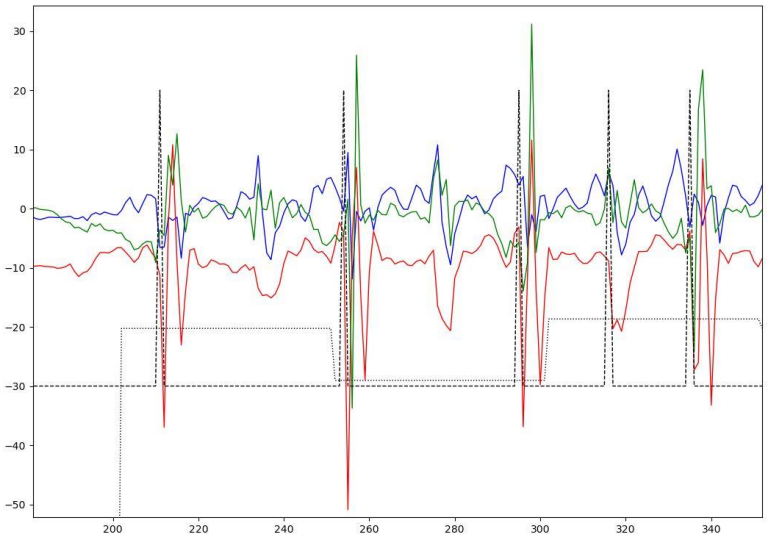


Figure N°2: All axes acceleration, steps detected (dashed black) and dynamic threshold output on the **Slow_walking.csv** dataset.

On the Figure N°2 around 250 timesteps we can observe that the computed threshold is too low compared to left leg step signal (the phone is head down so we talk about negative values) and that implies 2 real steps missed.

How your solution could be improved/generalized

We set up a dynamic threshold to improve fixed threshold behavior, there must also be a way to set up a dynamic least time between steps detection. Such an implementation will allow a better flexibility of my algorithms. Indeed, I didn't try it but if you try a running dataset or at least with a step frequency higher than 1Hz this minimal time security will result in missing real steps.

Another obvious point is about data filtering. I tried to implement this feature but struggled too much and had to give up. Whatever, a simple Low-Pass filter will attenuate noise and also reduce step resulting signal picks which sometimes imply a max or min value really strong and therefore create a not suitable threshold.

On the other hand, my code efficiency is not optimal, I think there is easy ways to optimize my for loops or thresholds computation. As we are almost talking about real time application that may be really important.

What did you learn?

This first assignment allowed me to re-use python and MatLab, even if there was not that much code it's always good to rediscover yours habits like reading function documentation, Git, etc. I also learn how to compute a dynamic threshold and to care about every unwanted behavior.

Did anything surprise you?

In a first place the easiness of MatLab Mobile usage, I had no idea you could access and use all your phone sensors like this. This is incredible and I will definitely use MatLab Mobile more often. And then the efficiency of a static threshold for that kind of exercise. Even if the datasets were pretty constant and I had to try several thresholds, the final fixed threshold was almost perfect for all of my 3 datasets even for stairs climb which is much more complex.

Did you find anything challenging? Why?

Nothing really challenging but thinking of ways to improve fixed threshold method result was not straightforward. Moreover, I spend quite a lot of time searching for the right threshold.

Did you find anything satisfying? Why?

Trying new thresholds and methods on different datasets and having no idea if it will fit or not is pretty exciting. Besides, results were satisfying so that was a nice experience. I would have that in general python is satisfying, the possibility to draw nice customized curves in a single line is impressive (especially when you come from the C world).