System Engineering: Pedometer on pebble watch

Antoine Albertelli Eloi Benvenuti Florian Kaufmann

October 2016

Contents

1 Introduction

In the context of our System Engineering class we had to design and code a pedometer for a pebble smart watch. This document explains our design process and presents the final results

2 Requirements

The need tied to the user of pedometer are the following:

- Have a mean to count his footsteps over the day.
- Have a mean to easily read his current footsteps for the day.
- Have a mean to access an history of his footsteps over the past days.
- Must not experience discomfort due to the use of the device. ...

From those needs we derived the following functions:

1. Acquire the relevant data for footsteps computing.
2. Process those data to compute the number of footsteps.
3. Update the number of footsteps frequently (every 1-2 seconds).
4. Display the number of footsteps for the current day.
5. Offer a mean to reset the current number of footsteps.
6. Offer a mean to look at the history of footsteps for the past week.
7. Do not reduce the battery life of the pebble watch noticeably.
8. Do not result in an overheating of the watch. ...

3 Algorithms

We tried several approaches in order to find a precise enough algorithm. We implemented prototype in Matlab or Python and ran them on the sample data. We then compared the number of counted step against the actual number of steps in order to rank them.

3.1 Frequency domain based algorithm

An attempt was made on the idea of filtering the data over and below a set frequency. However the idea was rejected because the walking frequency was too much dependant on the terrain, on the type of walk and on the wearer's height.

3.2 Time based algorithms

We compared 3 time-based algorithms against each other. Here is a quick description of their different strategy:

Algorithm 1 The 1st algorithm looked at the data generated by 1 axis of the accelerometer by block of 1 second. He would then look for the maxima and the minima in the block and, if the difference between these two was bigger than a set threshold, he would count 2 steps. As shown in figure **??** he had good results. However he was discarded because we feared his performance could drop if the wearer walked really slowly.

Algorithm 2 The 2nd one was counting the steps using the data of the three axis of the accelerometer and wanted to average them to get a better estimation of the results. His strategy, applied separately to each axis, was to find the average value of the data in the block, then count one step every time the data crossed this axis with a negative slope. In order to reduce miscounts due to the noise, he had an hysteresis that the data had to cross before a new step was counted.

As it is shown in figure **??** this algorithm performed very badly, overshooting the number of steps by a factor 2 at least. The averaging between the 3 axis wasn't helping, since the 2 worst-performing axis were only worsening the count of the "best" axis.

Algorithm 3 The 3rd algorithm count the steps using only 1 axis. Every time a new data point is generated, he computes the average value on the last 50 data and use it as a threshold to decide when to count a step. Contrary to the 2nd algorithm, this result in a threshold moving more smoothly than in the 2nd algorithm, which improves the precision of the algorithm.

He also uses a hysteresis to avoid mistaking electrical noise of small hand movement as steps. As shown in figure **??** his performance are very good: him never being more that 10% off most of the time.

After doing this comparison, we decided to implement algorithm number 3, due to his good performance and relative insensibility to a variation in the users height, weight and other physical characteristics.