1. Introduction

Step detection has many research and practical applications, including but not limited to: Computer Science, Engineering, Biology and Medical Sciences. As such, accurate and reliable step detection algorithms are in great demand. Whereas step detection required some wearable hardware in the past, the development of inertial sensor-enabled smart phones has facilitated a chance for researchers to create more cost-effective solutions. Among the myriad of sensors found in smart phones, accelerometers and gyroscopes are the most commonly used sensors in most of these applications. Considering the fact that a mobile phone can be at any position on the user body, with time-varying orientation change, we make use of the magnitude of 3-axis accelerometer reading instead of its vertical and horizontal components[1, 2]. The system we implemented combine a sort of smoothing filters and peak detection algorithm. We also introduce a calibration module for user to calibrate the step counter which can improve the accuracy.

2. Implementation

Generally speaking, our algorithms can be divided into following steps:

Step 1: Merge the (x,y,z) magnitudes of accelerometer

While the mobile phone is on the user's person, it may be in any orientation. So we cannot use the magnitudes of accelerometers in (x, y, z) axis individually. Consequently, we use a simple way to merge the magnitudes of the accelerometer in different dimensions, obtaining a length vector. Hence, we define the new merged value:

$$Value_{merged} = \sqrt{X^2 + Y^2 + Z^2}$$

Step 2: Use a low pass FIR digital filter to remove the high frequency noise and spikes

Random jitters on the phone may result in some high frequency noises, we must to use filters to rule them out. The low pass FIR is extensively used[2]. Given a cut-off frequency $F_{cut-off}$, we have $RC = 1/F_{cut-off}$, while $\alpha \approx \Delta T/(RC + \Delta T)$, where ΔT is the sampling period. We call α the smoothing factor, and by definition, $0 \le \alpha \le 1$. We use the following formula to implement implement the filter:

$$y_i = y_{i-1} + \alpha(x_i - y_{i-1})$$

In our algorithm, the cut-off frequency is empirically set to 7 Hz. Therefore, the smoothing factor is 0.13.

Step 3: Squaring operation

The squaring operation only works on positive data[2]. It can enhance large values more than small values. Specifically, it was used in order to amplify the effects of high-frequency components, hence making it easier to detect peaks.

Step 4: Integration operation

We smooth our results further from step 3 using this smoothing method, which can be called moving-window integration filter[2]:

$$y_i = (x_{i-(N-1)} + \ldots + x_{i-(N-2)} + \ldots + x_i)/N$$

In our algorithm, N=6.

Step 5: Detect the peaks under the restriction of two thresholds

After preprocessing the raw data by the previous steps, our algorithm searches for the peaks and valleys of the waveform in order to identify a distinct step. While searching for the peaks, the qualified peaks must satisfy two requirements:

1. The value of the peak must be greater than a given peak value threshold TH_{peak}

2. The interval between the peak value and its previous adjacent valley value must be greater than a given interval threshold $TH_{interval}$.

These two requirements are additional measures we take to rule out the noise. Initially, we provide default values of these two thresholds. To improve the accuracy, we introduced a calibration module into our system. This module can be used as follows:

1. User clicks the calibration button, and is instructed to take a few steps.
2. When finished, the user clicks the “Stop Calibration” button.
3. The system asks the user to enter the actual amount of steps taken.

According to the actual step counts and the collected data during the calibration period, our algorithm can estimate the peak value threshold and interval threshold.

The flowchart of our system can be described as:

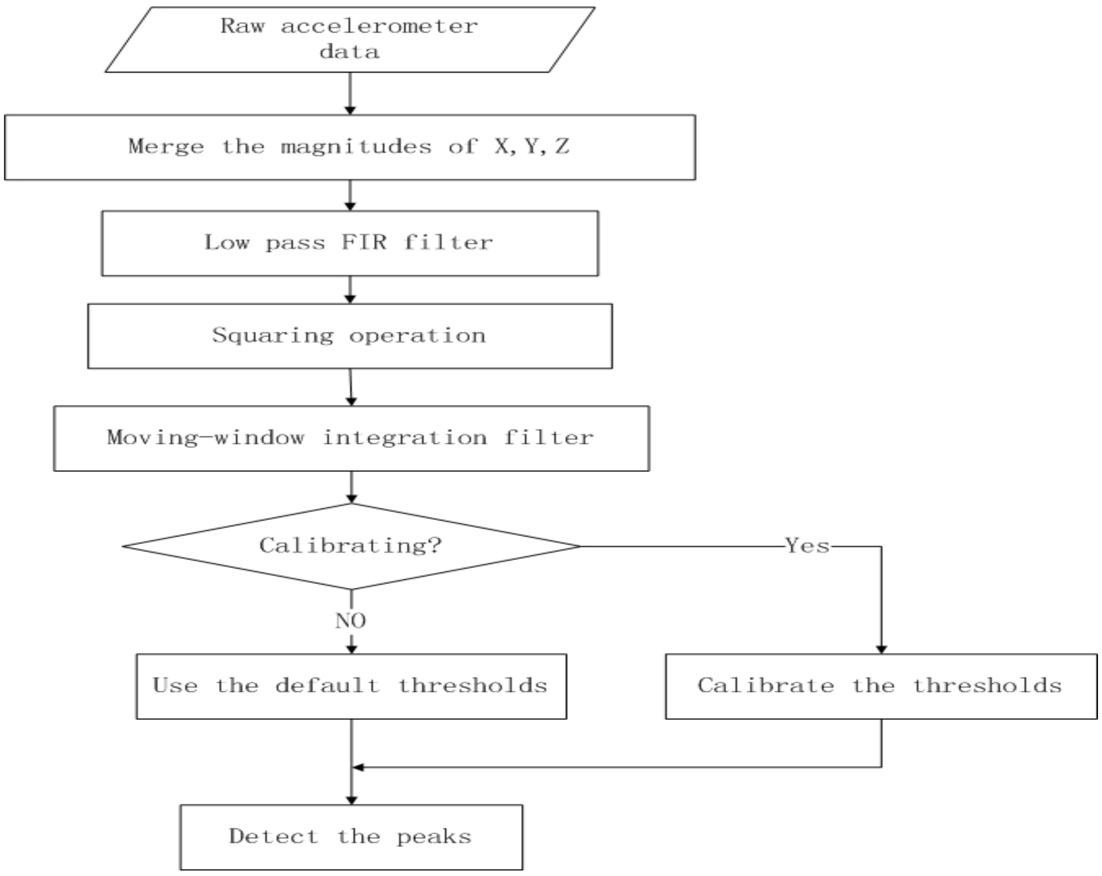

Figure 1. The flowchart of our algorithm

The smoothing filters, including the merge step, the low pass FIR, the squaring operation and moving-window filter, are relatively straightforward. Hence, will only specify the details about the calibrating and peak detection.

First of all, we describe the calibrating process. To use calibration module, the user is required to click the calibrate button and then walk several normal steps, and enter the actual step counts during the calibration period at the end. Our system can collect the value of peaks and value of intervals between peak and valley. Then, on the basis of these data, along with the actual step counts the user entered, we can estimate the peak value threshold and interval threshold:

The pseudo code of calibrating is as follows:

Input: (1) the list of peak value during the calibration period
(2) the list of bottom-up interval between peak and its previous adjacent valley
(3) the actual step counts entered by user

Output: peak value threshold and interval threshold.

```
Function calibrating(ArrayList peak_list, ArrayList interval_list, int count_step )
  sort(peak_list);                        //sort the peak_list in ascending order
  sort(interval_list);                    //sort the interval_list in ascending order
  IF step_count >= number_of_peaks        //we cannot detect enough peaks
        peak_threshold = default_peak_threshold_value
        interval_threshold = default_interval_threshold_value
  ELSE
        peak_threshold = corresponding_threshold_in_peak_list();
        interval_threshold = corresponding_threshold_in_interval_list();
```

The pseudo code of peak detection is as follows:

Peak-Detection Pseudo-code:

```
last_reading = 0
current_reading = 0
climbing = false
descending = false
num_peaks = 0
WHILE sensor_on
        IF climbing && last_reading < current_reading
                IF calibration_mode == true
                        add_to_peak_list(last_reading)
                        add_to_interval_list(last_reading - last_valley_value)
        peak_count = peak_count+1

        IF last_reading > peak_threshold &&
                    current_time - last_peak_time > 0 &&
                    last_reading - last_valley_value > interval_threshold)
            step_count = step_count + 1

        climbing = false;
        descending = true;
        last_peak_time = current_time;
        last_peak_value = last_reading;

        IF descending && lastReading < currentReading
            climbing = true;
            descending = false;
            valley_count = valley_count + 1;
            last_valley_value = last_reading;
        }

    last_reading = current_reading
```

3. Experiments

To evaluate the accuracy and performance of our system, we conduct a sort of experiments. We also compare the performances of the system with calibrations against that without calibrations. The smart phone we use in the experiments is Nexus 4.

First of all, the smoothing filters is analyzed. We pick a small piece of raw data, which is collected during five steps, to conduct this evaluation. This piece of raw data can be depicted as Figure 2. The three curves indicate sensor data in different dimensions. Then we merge the sensor data in the three dimensions (described in step 1), the output result is as Figure 3. Then we implement the derivative operator[1], the result is shown in Figure 4. Figure 5 is the output result of low pass FIR[2]. We could see that the performance of low pass FIR is much better when removing the noises. So in our final system, we make use of low pass FIR rather than the derivative operation.

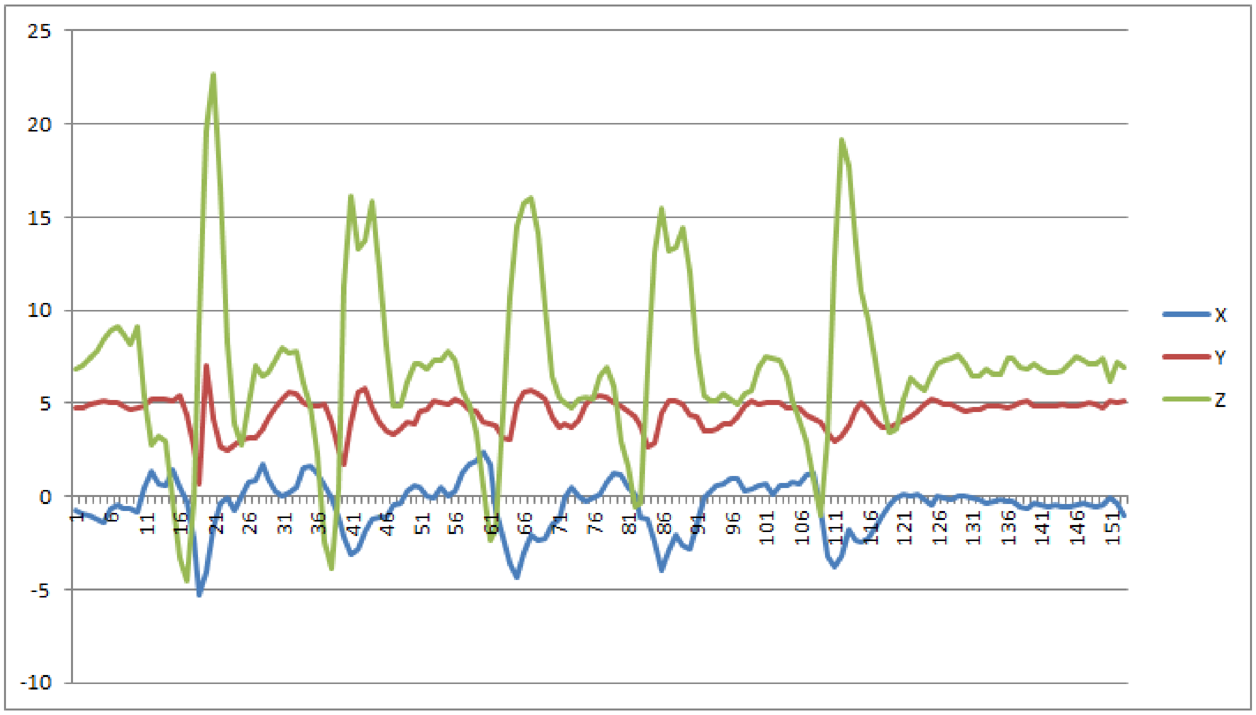

Figure2. This is the raw sensor data reported by the accelerometer over 5 steps.

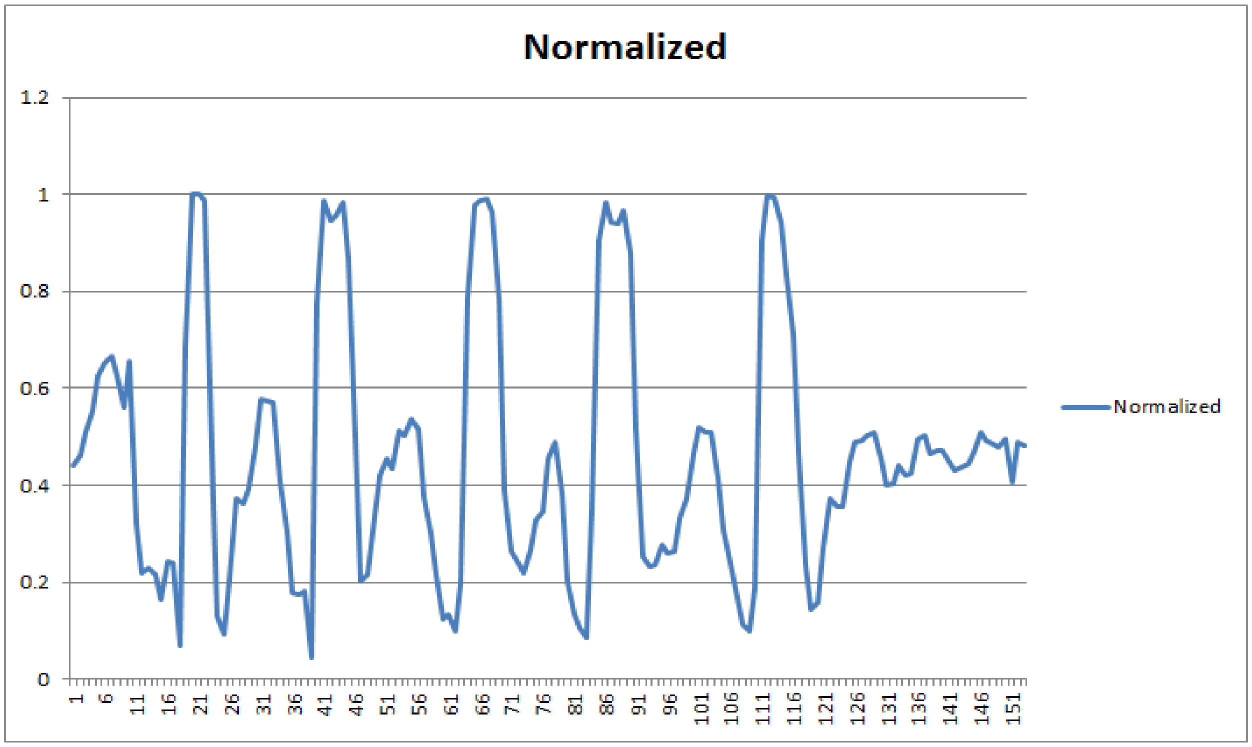

Figure 3. This is the output after normalizing (using the squaring function in step 1) the sensor values.

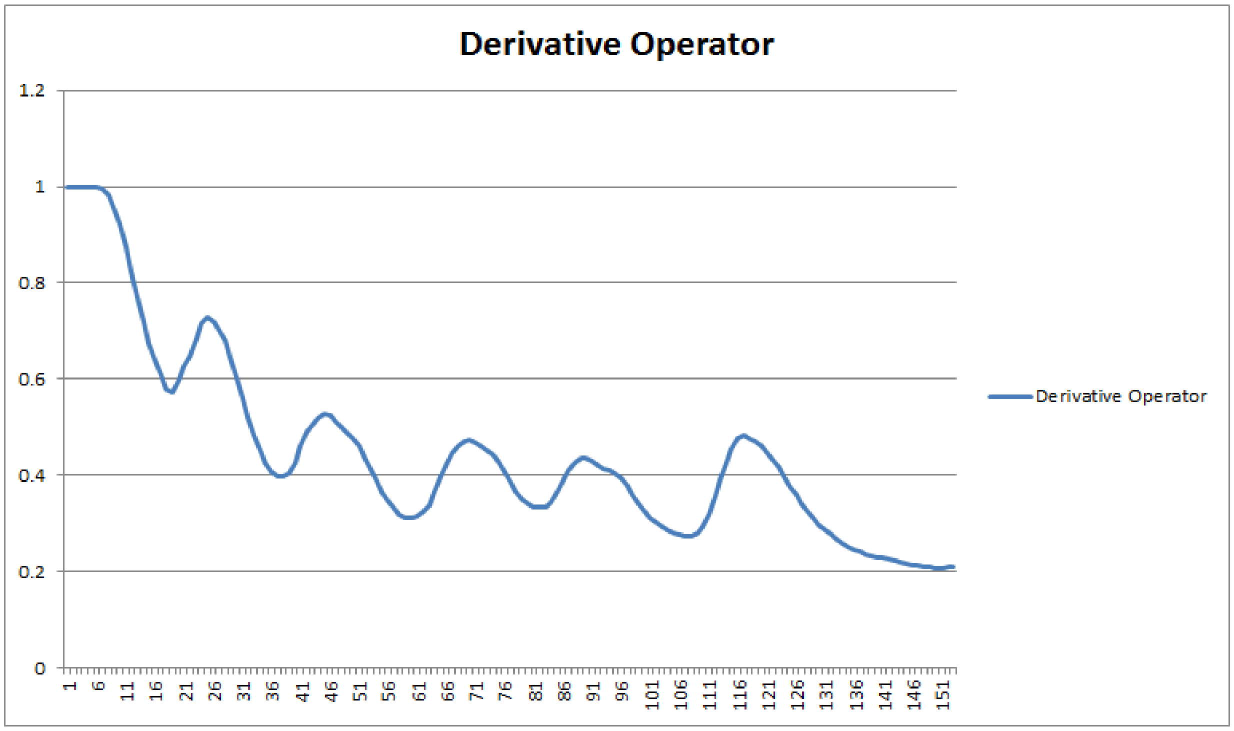

Figure 4. This is the output after only applying the derivative operator, which is not very accurate.

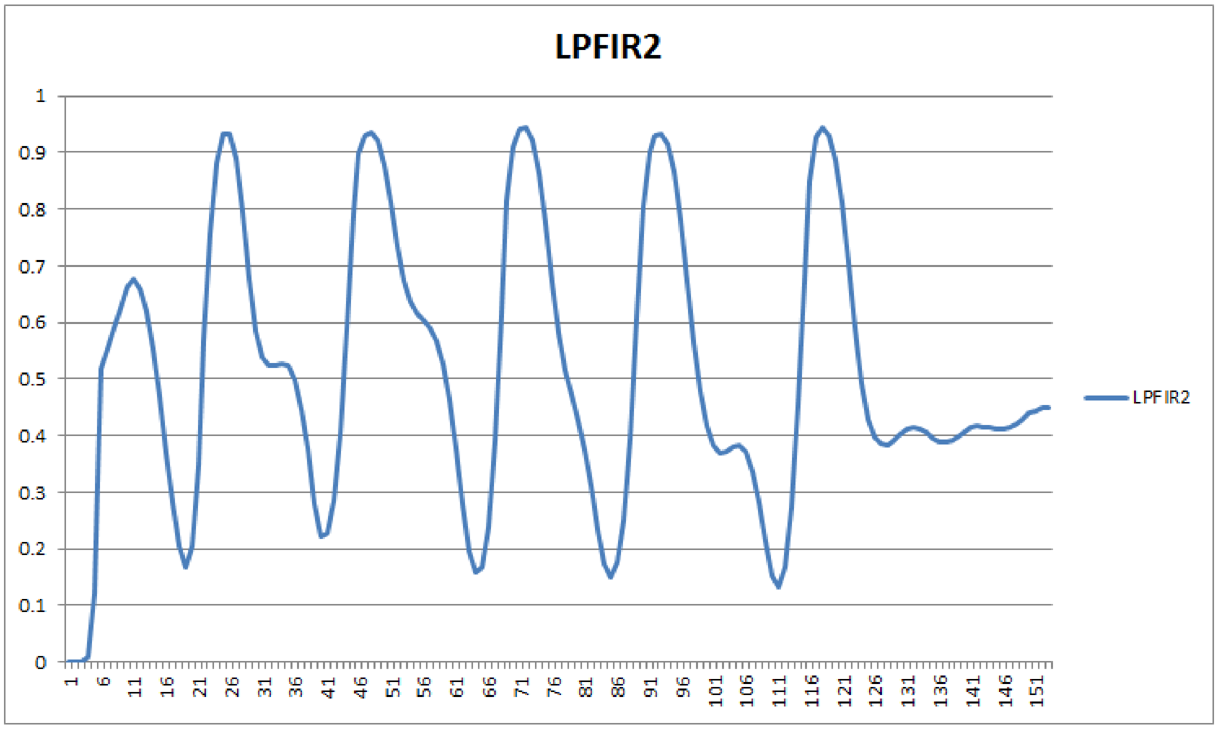

Figure 5. After applying the low pass FIR, it becomes much simpler to detect peaks resembling steps.

Then we conduct a series of experiments to evaluate the accuracy of our system. The experiments were carried out with the phone in hand. We have used the following two use cases:

1. User is walking normally.
2. User is jogging.

False positive and false negative rates were used in determining the performance of the algorithms. Tables 1 and 2 show the performance of the application in test case 1, with table 1 showing the results without calibration, and table 2 with calibration.

Table 1 (Before Calibration)

Actual # of steps	Total steps detected	False Positives	False Negatives
10	10	0	0
20	20	0	0
30	29	1	0
40	39	0	1
50	62	12	0

Table 2 (After Calibration)

Actual # of steps	Total steps detected	False Positives	False Negatives
10	10	0	0
20	20	0	0
30	30	0	0
40	40	0	0
50	53	3	0

The following tables represent use case 2, where the users were jogging with the phone in hand.

Table 1 (Before Calibration)

Actual # of steps	Total steps detected	False Positives	False Negatives
10	12	2	0
20	23	3	0
30	30	0	0
40	43	3	0
50	50	0	0

Table 2 (After Calibration)

Actual # of steps	Total steps detected	False Positives	False Negatives
10	10	0	0
20	20	0	0
30	30	0	0
40	40	0	0
50	51	1	0

From these tables, we can extrapolate that when walking and running before calibration, the percentage of error is approximately 9% and 5%, respectively. Meanwhile, the percentage of error for walking and running after calibration is approximately 2% and 1%, respectively. Therefore, the calibration is very helpful. Through further analysis, we can learn that the walking style varies from people to people. It is impossible to find out a common threshold for all cases, so the performance could be improved significantly if the thresholds are set flexibly.

One may speculate that running will cause a larger number of erroneous steps. However, the accelerometer is sensitive to any small fluctuations and jitters (even though we have applied multiple filters, the phone must remain in relatively the same position relative to the user). Jogging does not create small jitters when it is in the user's hands, in fact it is more likely to have a more fluid motion than walking as the user's movements are more in sync and coordinated with each other as opposed to walking.

Despite these results, the phone must be in the user's hand for this application to function reliably and accurately. In order to allow the user more freedom, we plan to incorporate gyroscope measurements in order to further fine tune the application to ignore movements when the user's position is still the same. In addition, in the calibration module, the participation of the users is required, we may need to develop some advanced approaches to remove the human intervention.

References

[1] Ying, H., Silex, C., Schnitzer, A., Leonhardt, S., & Schiek, M. (n.d.). Automatic Step Detection in the Accelerometer Signal.

[2] Li, F., Zhao, C., Ding, G., Gong, J., Liu, C., & Zhao, F. (2012). A reliable and accurate indoor localization method using phone inertial sensors. In Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12 (p. 421). New York, New York, USA: ACM Press. doi:10.1145/2370216.2370280.