A Step Counting Algorithm for Smartphone Users: Design and Implementation

Meng-Shiuan Pan, *Member, IEEE*, and Hsueh-Wei Lin

***Abstract*— The step count is an important information for developing services for smartphone users. Most existing step counting solutions restrict that: 1) the phone has to be fixed to the user and 2) the user cannot use the phone naturally while walking. We can see that these restrictions are inconvenient for users. In this paper, we propose a step calculation algorithm, which can relieve the above restrictions and can count users' steps precisely. The proposed algorithm is composed of two phases. The first phase collects linear acceleration and gravity values from the smartphone's accelerometer. Then, this phase derives the horizontal components of the perceived linear acceleration values and identifies possible start points of periodical regular fluctuations (of linear acceleration measurements). The second phase adopts the concept of correlation coefficients to identify whether the collected sensing measurements exhibit similar tendencies and calculates step counts. In this paper, we implement the proposed method on the android platform. The experiment results indicate that the proposed scheme can analyze gaits accurately and count steps effectively.**

***Index Terms*— Accelerometer, gait analysis, smartphone, step counting.**

I. Introduction

SMARTPHONES have become the most popular devices in our daily lives. In a smartphone, there are many in-built communication modules (e.g., WIFI, Bluetooth, and GPS) and sensor modules (e.g., accelerometer, gyroscope, and magnetic sensor). By these communication and sensing modules, smartphones can be used to develop various kinds of services and applications.

Recently, some researchers or companies use the *step count* information to develop various smartphone services. For example, in the indoor localization service, some researches (see [9], [21], [25]) propose to utilize users' trajectories to assist wireless fingerprint indoor positioning functions (see [8], [13], [17]). To infer a user's walking trajectory, these schemes need to know the user's walking direction and *distance*, where the distance information is usually estimated by the user's step counts. Another example is the health management service. Some APPs (see [2]–[4]) record how many steps that a user walked in a day. After collecting the user's habits, these APPs can give some health tips for the user. Some APPs (see [5], [6]) further utilize the step count information to design games, which can motivate user to do exercises.

Many existing step counting solutions have shown that when a user is walking, the perceived readings of the accelerometer (on the user's body) will change regularly. These changes can be used to estimate the number of steps that the user takes. Fig. 1 shows two experiment results on the perceived *linear acceleration*[1] values from the accelerometer when a user walked 10 steps. In the Fig. 1(a), the user holds the smartphone in the right hand and swinging her hand naturally with gaits, and in the Fig. 1(b), the user holds the smartphone in the right hand and watches the screen while walking. From these results, we can see that regardless of how the user carries the phone, the linear acceleration values exhibit periodical and regular changes over time based on the gaits of the user. However, in Fig. 1, compare to the linear acceleration values of Y and Z axes, the values of X may not change regularly. This is because that the linear acceleration values of X axis may be affected easily even when the user slightly shakes the phone. From Fig. 1, one regular fluctuation of the linear acceleration values of Z axis can be taken as one step of the user. But, in Fig. 1(a) and Fig. 1(b), one regular fluctuation of Y's linear acceleration values represents two steps and one step, respectively.

Based on the above experiments, our work is motivated by the following two observations. First, from the above post-processed graphical results, we can easily judge that one regular fluctuation of perceived linear acceleration values represents one step or two steps. However, in reality, the sensory values obtained from the accelerometer are a series of continuous raw data. So, we need an algorithm that can distinguish or divide continuous data exhibiting similar tendencies in realtime. Second, we can see that different *carrying ways* (i.e., hold the phone by different manners or place the phone in different locations) will result in different changes on linear acceleration values. However, existing solutions restrict the users to carry their smartphones (or sensors) in a particular fashion or to void shaking the carried sensing devices, and these restrictions are inconvenient for users. Thus, the designed algorithm should have the capability to handle sensory readings, which may fluctuate dynamically.

Manuscript received September 18, 2014; revised November 26, 2014; accepted November 26, 2014. Date of publication December 4, 2014; date of current version February 5, 2015. This work of M.-S. Pan was supported by the Ministry of Science and Technology under Grant 103-2221-E-032-030. The associate editor coordinating the review of this paper and approving it for publication was Prof. Octavian Postolache.

The authors are with the Department of Computer Science and Information Engineering, Tamkang University, Taipei 25157, Taiwan (e-mail: mspan@mail.tku.edu.tw; 601410052@s01.tku.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSEN.2014.2377193

[1]The linear acceleration readings indicate the forces that are not caused by gravity.

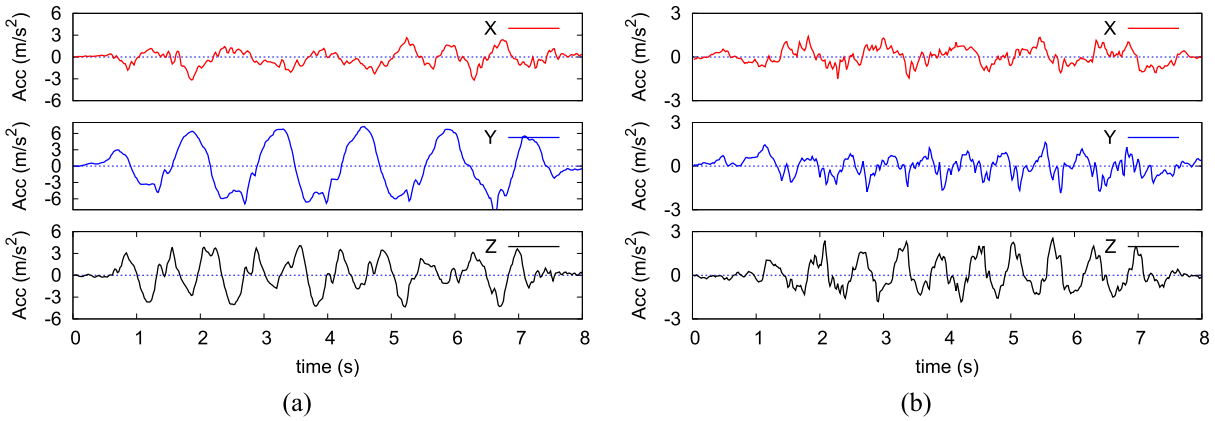

Fig. 1. The perceived linear acceleration values when the user (a) holds the smartphone in the right hand and swings her hands naturally with gaits and (b) holds the smartphone in the right hand and watches the screen while walking.

In this paper, we propose a step counting algorithm, which can identify steps in realtime and allow users to carry their smartphones arbitrarily while walking. The proposed algorithm utilizes the linear acceleration values of Y and Z axes to count steps.[2] Our algorithm is composed of a *data collection phase* and a *data analysis phase*. The data collection phase involves data collection, deriving the horizontal components of the perceived linear acceleration values, and identifying possible start points of regular fluctuations. The data analysis phase adopts the concept of correlation coefficients to identify similar tendencies, and then counts steps. In this work, we implement the proposed method on the Android platform. The experiment results indicate that regardless of how the user changes carrying ways of the phone while walking, the proposed method can accurately divide gait changes and effectively identify steps.

The contributions of this paper are threefold. First, the data collection phase can facilitate allowing users to carry phones without any constraint. Second, the data analysis phase can accumulate users' steps correctly even if users change carrying ways while walking. Third, the proposed scheme can identify similar tendencies effectively in realtime. The proposed scheme can also be applied to analyze continuous and regular sensory values generated by sport postures.

The rest of this paper is organized as follows. Section II introduces some related works. Section III and Section IV describe the designed data collection phase and data analysis phase, respectively. Section V shows our prototyping results. Finally, we conclude this paper in Section VI.

II. Related Works

In this section, we discuss some existing step counting methods, which are divided into the following four categories.

1) Threshold Setting: In references [7], [14], [16], [24], the proposed methods count steps based on judging whether the sensory readings can satisfy some thresholds. In [16], the authors propose to install an inertial sensor on users' legs to facilitate step calculation. The main idea is to compare whether the perceived accelerometers' data tendencies successively exceeding two predefined thresholds. If so, one moving step will be counted. In addition, the authors also propose to use gyroscopes and magnetic sensors to determine the number of steps. This idea is similar to the one by using accelerometer sensing measurements and can yield a similar effectiveness. In [24], the authors develop an approach, which is similar to the one in [16]. In [14], the authors tie an inertial sensor on the user's ankle. When the measured acceleration sensing data exceeds a threshold, the user's step counts will be accumulated accordingly. In [7], the authors suggest to use a finite state machine combining with threshold settings to calculate steps. Their work is so designed based on the observation that when a user walks, walks briskly, or runs, the acceleration measurements detected by the handheld device will exhibit different regular variations. The authors divide the regular variations into several states by using various threshold values. When counting steps, the proposed scheme first determines the state of a user, and then determines if the sensory values can satisfy the threshold designated for that state. Although the reference [11] supports that the threshold-based step counting methods are robust, it will be hard to designate a unified threshold value, which is suitable for various carrying ways and users.

2) Peak Detection: References [12], [15], [19], [23] show that steps can be detected by the pattern of successive peak and valley acceleration values. The authors in [15] show that when walking, the perceived horizontal and vertical acceleration values can be modeled as sinusoidal waves. Thus, a step can be determined when a sinusoidal pattern is identified. Reference [19] designs a step counter service for JAVA-enabled devices. The implemented program detects "hills" of continuously perceived acceleration values. When a hill is found, one step is accumulated. In [12], after obtaining acceleration readings, the proposed scheme determines local maximums among a series of sensing measurements. Then, a local maximum is considered to represent that the user has moved one step. The authors of [23] explore using a low-pass filter to filter noises in received acceleration values. Then, the proposed scheme tries to determine a local maximum value between two local minimum ones among

[2]Note that we only utilize the linear acceleration values of Y and Z axes to count steps because that the linear acceleration values of X axis may be easily affected when users shake phones.

a group of measurements. The authors identify whether the difference between two nearby local minimum and local maximum exceeds a predefined threshold. If so, one step is identified. We can see that the complexities of peak detection schemes are low. But, the step count results of peak detection schemes may not be accurate because that when users carries their phones arbitrary, extra steps (due to unexpected peaks) will be sensed.

3) Gait Analysis: In [10], [18], and [20], the authors employ the *zero velocity update* (*ZUPT*) concept to calculate steps. The ZUPT concept is that when walking, a user's lower extremity will be vertical to the horizontal plane at a certain time, and at the time instant, the walking velocity will zero. References [10], [18], [20] use this characteristic to count steps. To detect the zero velocity, in [20], the authors gather scalar values from the gyroscope sensor mounted on a user's shoe. If the received scalar values are smaller than a threshold, one stride will be counted. In [10], the authors install pressure sensors in shoes to detect the zero velocity while walking. The authors in [18] detect the zero velocity by installing an inertial sensor on the user's calf. From the results in [10], [18], and [20], the above three schemes can effectively detect user's steps. However, users are required to wear additional inertial sensors, which are considered to inconvenience users.

4) Correlation Calculation: In reference [22], the authors propose to use the autocorrelation concept to calculate steps. After collecting the sensory data, the proposed scheme determines a start point among the collected data, and then determine a possible end point. Then, the proposed scheme performs autocorrelation from the start point to the end point. If the result of autocorrelation exceeds a given threshold, one step will be counted. This method enables the user to carry the smartphone at will. But, the proposed method in [22] does not consider the case that the user may change carrying ways of the phone while walking.

Based on the above discussions, in this paper, we proposed a step counting method for smartphone users, which can effectively count steps and enable users to change carrying ways of phones while walking. Our scheme contains two phases, and the details will be presented in the following sections.

III. Data Collection Phase

In this phase, we gather the sensory readings of linear acceleration and gravity from accelerometer every n millisecond. At time t_i, the perceived linear acceleration and gravity values are defined as $A(t_i) = (a_{x,t_i}, a_{y,t_i}, a_{z,t_i})$ and $G(t_i) = (g_{x,t_i}, g_{y,t_i}, g_{z,t_i})$, respectively. After obtaining $A(t_i)$ and $G(t_i)$, this phase executes a *data transformation module* and then a *start point detection module*.

A. Data Transformation Module

This module is needed because that when a user walks and carries her phone arbitrarily, the sensed linear acceleration values may change irregularly. More specifically, when the phone is not parallel with the horizontal plane, the linear acceleration values will be distributed on X, Y, and Z axes, and thus the gathered linear acceleration values cannot represent the actual forces that performed by the user. The upper and bottom subfigures in Fig. 2 shows the linear acceleration values of Z axis before and after transformation, respectively. We can see that in Fig. 2, the linear acceleration values of the bottom subfigure are more regular. So, the main goal of this module is to derive the horizontal components of the received linear acceleration values. This module facilitates users to carry phones without any constraint because that after transformation, their phones can be considered to be parallel with the horizontal plane while walking. (Note that the results shown in Fig. 1 are transformed by this module.)

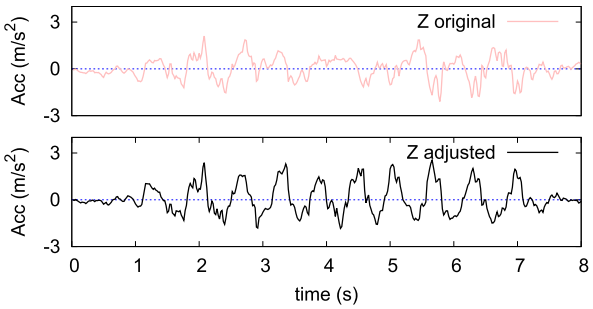

Fig. 2. The linear accelerate values of Z axis before and after transformation.

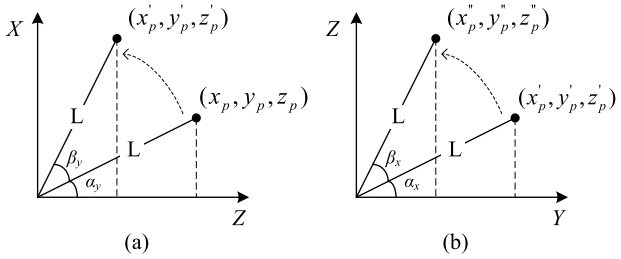

Fig. 3. The rotations of the point p around (a) the Y axis and (b) the X axis.

Before showing the details, we first present the concept of *coordinate rotation* as follows: Assume that there is a point p with axes (x_p, y_p, z_p) in a 3D plane. The distance between the origin of coordinates and the point p is L. Refer to Fig. 3(a), the included angle between the Z axis and the line segment $\vec{p}$ is α_y. (Note that in Fig. 3(a), the Y axis injects from the plane.) In Fig. 3(a), when rotating the $\vec{p}$ around the Y axis counterclockwise by β_y degrees, the coordinates of p will change to (x'_p, y'_p, z'_p). According to the definitions of the trigonometric functions, we can obtain the following four relationships:

$$\sin(\alpha_y) = \frac{x_p}{L}, \quad \cos(\alpha_y) = \frac{z_p}{L},$$
$$\sin(\alpha_y + \beta_y) = \frac{x'_p}{L}, \quad \cos(\alpha_y + \beta_y) = \frac{z'_p}{L}.$$

Then, according to the sum formulas $\sin(v + u) = \sin(v)\cos(u) + \cos(v)\sin(u)$ and $\cos(v+u) = \cos(v)\cos(u) - \sin(v)\sin(u)$, we can further obtain the following relationships.

$$\sin(\alpha_y + \beta_y) = \frac{x'_p}{L} = \frac{x_p}{L} \times \cos(\beta_y) + \frac{z_p}{L} \times \sin(\beta_y)$$
$$\cos(\alpha_y + \beta_y) = \frac{z'_p}{L} = \frac{z_p}{L} \times \cos(\beta_y) - \frac{x_p}{L} \times \sin(\beta_y)$$

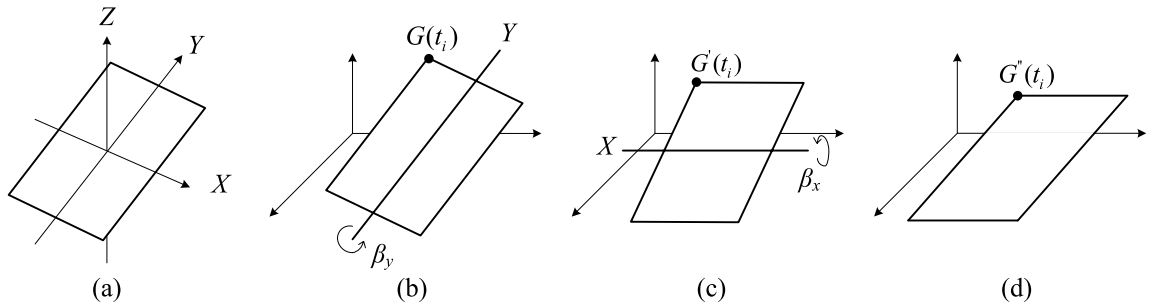

Fig. 4. (a) The coordinate of the smartphone and (b)–(d) the transform procedures to derive $G''(t_i)$.

By eliminating L, we can obtain the Eq. (1), which can be used to calculate the values of (x'_p, y'_p, z'_p).

$$\begin{aligned} x'_p &= x_p \times \cos(\beta_y) + z_p \times \sin(\beta_y) \\ y'_p &= y_p \\ z'_p &= -x_p \times \sin(\beta_y) + z_p \times \cos(\beta_y) \end{aligned} \tag{1}$$

Then, as shown in Fig. 3(b) (where the X axis injects from the plane), we can further rotate the $\vec{p}$ around the X axis counterclockwise by β_x degrees. The coordinate of p will further change to (x''_p, y''_p, z''_p). Following the above calculations, we can obtain the Eq. (2).

$$\begin{aligned} x''_p &= x'_p \\ y''_p &= y'_p \times \cos(\beta_x) - z'_p \times \sin(\beta_x) \\ z''_p &= y'_p \times \sin(\beta_x) - z'_p \times \cos(\beta_x) \end{aligned} \tag{2}$$

In this module, we utilize the perceived gravity values and the coordinate rotation concept to derive the tilt angles of the phone. For a smartphone, the coordinate system is defined relative to the device's screen as shown in Fig. 4(a). Assume that at time t_i, the perceived gravity value $G(t_i) = (g_{x,t_i}, g_{y,t_i}, g_{z,t_i})$. We can take the $G(t_i)$ as a point on the phone's coordinate system. According to the characteristic of gravity sensors, when a phone is placed horizontally on a surface, the perceived gravity values of X, Y, and Z axes will be 0, 0, and 9.8 m/s^2, respectively. So, the values of $G(t_i)$'s horizontal component $G''(t_i) = (g''_{x,t_i}, g''_{y,t_i}, g''_{z,t_i})$ will be $(0, 0, 9.8)$. Assume that we know how to rotate the phone to be paralleled with horizontal plane, i.e., the tile angles β_x and β_y are assumed to be known. Then, we derive the relationship between $G(t_i)$ and $G''(t_i)$ as follows: First, we rotate the $G(t_i)$ around the Y axis counterclockwise by β_y degrees (as shown in Fig. 4(b)) to obtain $G'(t_i)$. Next, as shown in Fig. 4(c), we further rotate the $G'(t_i)$ around the X axis counterclockwise by β_x degrees to obtain $G''(t_i)$ in Fig. 4(d). So, based on the Eq. (1) and Eq. (2) discussed above, we have the following equations.

$$g''_{x,t_i} = g_{x,t_i} \times \cos(\beta_y) - g_{z,t_i} \times \sin(\beta_y) \tag{3}$$

$$\begin{aligned} g''_{y,t_i} = {} & g_{y,t_i} \times \cos(\beta_x) + g_{x,t_i} \times \sin(\beta_y) \times \sin(\beta_x) \\ & + g_{z,t_i} \times \cos(\beta_y) \times \sin(\beta_x) \end{aligned} \tag{4}$$

$$\begin{aligned} g''_{z,t_i} = {} & g_{y,t_i} \times \sin(\beta_x) + g_{x,t_i} \times \sin(\beta_y) \times \cos(\beta_x) \\ & + g_{z,t_i} \times \cos(\beta_y) \times \cos(\beta_x) \end{aligned} \tag{5}$$

By the above relationships, we can substitute the perceived gravity value $G(t_i)$ and $(g''_{x,t_i}, g''_{y,t_i}, g''_{z,t_i}) = (0, 0, 9.8)$ into Eq. (3)–(5) to derive the tile angles β_x and β_y.

In the following, we calculate the horizontal components of linear acceleration values based on the derived β_x and β_y. Assume that, at time t_i, the linear acceleration values are $A(t_i) = (a_{x,t_i}, a_{y,t_i}, a_{z,t_i})$. We can substitute a_{x,t_i}, a_{y,t_i}, and a_{z,t_i} and the β_y into Eq. (1) to obtain $(a'_{x,t_i}, a'_{y,t_i}, a'_{z,t_i})$. Then, we further substitute the results and the β_x to the Eq. (2). Finally, we can obtain the $A''(t_i) = (a''_{x,t_i}, a''_{y,t_i}, a''_{z,t_i})$ such that the values in $A''(t_i)$ represent the horizontal components of $A(t_i)$.

After the above transformations, this module further updates $A(t_i) = A''(t_i)$ for the following start point detection module.

B. Start Point Detection Module

In this module, we determine some reference points, which can be considered as the start timings that the smartphone user starts to walk or the section points between the user's steps. Before showing the details of finding start points, we first observe that when placing a smartphone on a flat surface, the detected linear acceleration values may fluctuate due to the noise of the accelerometer. For example, in Fig. 5, during the time interval 0 to 1 second, the linear acceleration values change between $\pm 0.2\ m/s^2$.

In this work, we first use a training procedure to learn the noise boundaries of linear acceleration values of Y and Z axes. The training procedure starts by placing the phone on a flat surface for m minutes. Then, the training procedure uses variables T^y_{usp} and T^y_{lsp} (resp., T^z_{usp} and T^z_{lsp}) to record the maximum and the minimum linear acceleration values of the Y axis (resp. Z axis) during the m minutes, respectively. For example, in Fig. 5, we have $(T^z_{usp}, T^z_{lsp}) = (0.2, -0.2)$.

Given a set of Y's linear acceleration values, say $A_y = \{a_{y,t_i}, a_{y,t_{(i+1)}}, a_{y,t_{(i+2)}}, ...\}$ (processed by the module in Section III-A), and two boundary values (T^y_{usp}, T^y_{lsp}), this module finds *time points* of Y axis as follows: This module traverses those values in A_y in sequence. Assume that this module currently handles the value a_{y,tp_k} at time instant tp_k. Then, this module performs the following two checks:

- C1: $a_{y,tp_k} \geq T^y_{usp}$ and $a_{y,tp_{(k-1)}} < T^y_{usp}$
- C2: $a_{y,tp_k} \leq T^y_{lsp}$ and $a_{y,tp_{(k-1)}} > T^y_{lsp}$

If C1 or C2 is satisfied, this module records the time instant tp_k in a set TP_y, and then processes the next value

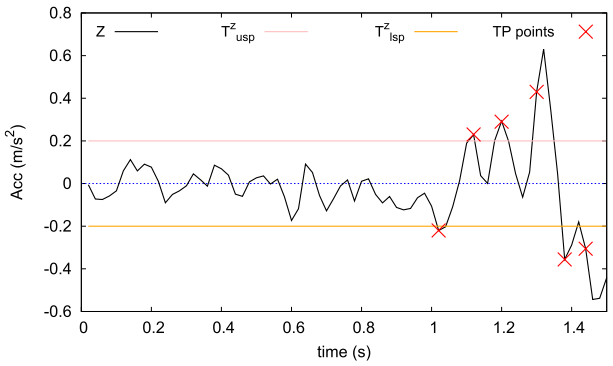

Fig. 5. The sets of TP_z points derived from T^z_{usp} and T^z_{lsp}.

in A_y. Note that the above procedure records those time instants that the corresponding sensory values cross the boundaries T^y_{usp} or T^y_{lsp}. Also note that after the above procedure, this module records time points of Y axis by the set $TP_y = \{tp_j, tp_{(j+1)}, tp_{(j+2)}, ...\}$, where the recorded tp_j can map to the corresponding a_{y,tp_j} in A_y. Moreover, given $A_z = \{a_{z,t_i}, a_{z,t_{(i+1)}}, a_{z,t_{(i+2)}}, ...\}$ and (T^z_{usp}, T^z_{lsp}), we can determine the set TP_z by the same procedure. For example, Fig. 5 shows six time points in set TP_z.

We remark that our algorithm identifies steps while the user walks. So, the sets A_y, A_z, TP_y, and TP_z will dynamically increase every n milliseconds (when receiving sensory readings from the accelerometer). Then, these sets A_y, A_z, TP_y, and TP_z will be consumed by the data analysis phase.

IV. Data Analysis Phase

In our scheme, we count steps based on the number of *segments* found in Y and Z axes. For the Z axis's linear acceleration values, we take every two sets of regular fluctuations as one segment. For the Y axis's linear acceleration values, we take every two or four sets of regular fluctuations as one segment. Based on the observation in Fig. 1, a segment of Y and Z axes will represent four and two steps, respectively. Given the TP_y and TP_z derived in the previous phase, we utilize the calculation of *correlation coefficient* to determine segments. Given two sets of values A_1 and A_2, the correlation coefficient $\rho(A_1, A_2)$ will be located between -1 and 1. When the result of $\rho(A_1, A_2)$ approaches 1, it means that the sets of values in A_1 and A_2 have a stronger correlation.

The basic idea of identifying segments is as follows: From a TP_y (or TP_z), we find two time points tp_i and tp_j, and take those linear acceleration values between tp_i and tp_j as a temporary segment S_1. Then, we divide the linear acceleration values of S_1 into two equal parts, and use a function $\rho_{ia}(tp_i, tp_j)$ to calculate the correlation coefficient of these two parts. If the result of $\rho_{ia}(tp_i, tp_j)$ exceeds a predefined threshold T_{ia}, we can say that the segment S_1 is identified to contain two similar fluctuations. The S_1 can represent four or two steps (depended on the tp_i and tp_j values come from TP_y or TP_z). However, in order to ensure that the perceived linear acceleration values are regularly fluctuated, we need to further check if two adjacent segments are correlated. So, we find another segment S_2, which is bounded by time points tp_j and tp_k. (Note that the tp_j is a boundary of S_1.) After ensuring that the $\rho_{ia}(tp_j, tp_k)$ also exceeds the threshold T_{ia}, we further compute the correlation coefficient of S_1 and S_2 by a function $\rho_{ir}(tp_i, tp_j, tp_k)$. If the result of $\rho_{ir}(tp_i, tp_j, tp_k)$ also exceeds a threshold T_{ir}, it means that the segments S_1 and S_2 are continuously and regularly fluctuated. Then, we can follow the above procedure to find the next segment S_3. However, if we cannot find the S_2 or the S_1 and S_2 are not correlated, we will re-find a new S_1. Moreover, from Fig. 1, we observe that the linear acceleration values may not be so regular when the user starts to walk. So, in the propose algorithm, when comparing the first two segments, we will adopt a smaller threshold value T_{ir_low}. When the detected fluctuations are stable, we will then adopt the threshold T_{ir}.

In the following, we show some variables used in the proposed segment finding procedure.

- A variable SC, which records the user's step counts, and will be updated by an update_SC() thread (described later). The default value of SC is 0.
- A flag *stable*, which represents that whether we have found consecutive segments in Y or Z axis, and the default value of *stable* is false.
- A window variable W, which is used to record the estimated period of segments in Y or Z axis, and the default value of W is 0.
- A temporary time point variable tp_{prev}, which is used to keep the start time point of the previous segment, and the default value of tp_{prev} is null.
- Two constants δ_y and δ_z, which represent the deviations of the variable W.
- Two constants d_{min_y} and d_{min_z}, which represent the least length of a segment of Y and Z axes, respectively. According to the our experiment results and the ones in [22], a user needs more than 0.4 s to complete a step. When finding segments in TP_y and TP_z, the time differences between the start and end time points of a segment should be larger than 1.6 s (4 steps) and 0.8 s (2 steps), respectively. So, the values of d_{min_y} and d_{min_z} will be set to 1.6 and 0.8, respectively.
- Two constants d_{max_y} and d_{max_z}, which represent the stop conditions when searching segments in Y and Z axes, respectively. In this work, we assume that a gait will not be longer than three seconds. So, the default values of d_{max_y} and d_{max_z} will be set to 12 and 6, respectively.

Given the sets A_y, A_z, TP_y, and TP_z, assume that we are now processing a time point, say tp_{cur}, in TP_y or TP_z. The designed segment_finding(tp_{cur}) procedure calculates step counts SC as follows. (Note that in the system, there exists two segment_finding(.) procedures, which handle the linear acceleration values of Y and Z axes, respectively.)

The procedure segment_finding(tp_{cur})

1) If the flag *stable* is false, we perform P1(tp_{cur}).
2) Otherwise, we perform P2(tp_{cur}). □

The sub-procedure P1(tp_{cur})

1) We put those time points, which locate between $tp_{cur} + d_{min\_y}$ (resp., d_{min_z}) and $tp_{cur} + d_{max\_y}$ (resp., d_{max_z}),

to a queue Q_{tp1} according to their timings in an increasing order. These time points are the candidate end points of the segment (starting from the tp_{cur}).

2) Letting tp_j = dequeue(Q_{tp1}), there are three cases.
   a) $tp_j = \emptyset$: This case represents that we cannot find an end point from Q_{tp1}. We set tp_{cur} = next_of(tp_{cur}), where the function next_of(tp_{cur}) is to find the next time point of tp_{cur} in TP_y (resp., TP_z). Then, we restart segment_finding(.) with the updated tp_{cur}.
   b) $\rho_{ia}(tp_{cur}, tp_j) < T_{ia}$: This case represents that the linear acceleration values in the segment bounded by (tp_{cur}, tp_j) are not correlated. So, we re-perform the step 2 again.
   c) $\rho_{ia}(tp_{cur}, tp_j) \geq T_{ia}$: This case represents that we find a segment. Then, we set the variable W to be the time differences between tp_{cur} and tp_j, and execute the step 3.
3) From tp_j, we further put those time points, which locate between $tp_j + W - \delta_y$ (resp., δ_z) and $tp_j + W + \delta_y$ (resp., δ_z), to a queue Q_{tp2} according to their timings in an increasing order.
4) Letting tp_k = dequeue(Q_{tp2}), there are three cases.
   a) $tp_k = \emptyset$: This case represents that we cannot find an end point in Q_{tp2}. In this case, we set tp_{cur} = next_of(tp_{cur}), and then re-perform segment_finding() with the updated tp_{cur}.
   b) $\rho_{ia}(tp_j, tp_k) < T_{ia}$ OR $\rho_{ir}(tp_{cur}, tp_j, tp_k) < T_{ir\_low}$: This case represents that the linear acceleration values in the segment bounded by (tp_j, tp_k) are not correlated or the segments bounded by (tp_{cur}, tp_j) and (tp_j, tp_k) are not correlated. So, we re-perform the step 4 again.
   c) $\rho_{ir}(tp_{cur}, tp_j, tp_k) \geq T_{ir\_low}$: This case represents that we find a segment, and the segment is correlated with the one found in the step 2. Then, we perform the following five operations. (i) We set *stable* to be true. (ii) We set the variable W to be the time difference between tp_j and tp_k. (iii) We update $tp_{prev} = tp_j$ and $tp_{cur} = tp_k$. (iv) We perform the record_segment() function (describe later). (v) Finally, we re-perform segment_finding(.) with the updated tp_{cur}. □

The sub-procedure P2(tp_{cur})

1) From tp_{cur}, we put those time points, which locate between $tp_j + W - \delta_y$ (resp., δ_z) and $tp_j + W + \delta_y$ (resp., δ_z), to a queue Q_{tp3} according to their timings in an increasing order.
2) Letting tp_j = dequeue(Q_{tp3}), there are four cases.
   a) $tp_j = \emptyset$: This case represents that we cannot find a segment that can be similar to previous ones. In this case, to compensate possible losses on counting steps, we first start a compensate_SC() thread (describe later). Next, we set $tp_{prev} = null$ and $W = 0$. We then re-find a new start of consecutive segments by setting $stable = false$ and tp_{cur} = next_of(tp_{cur}), and re-perform segment_finding(tp_{cur}).
   b) $\rho_{ia}(tp_{cur}, tp_j) < T_{ia}$ OR $\rho_{ir}(tp_{prev}, tp_{cur}, tp_j) < T_{ir\_low}$: This case is similar to the case 4(b) in procedure P1. So, we re-perform the step 2 again.
   c) $T_{ir\_low} \leq \rho_{ir}(tp_{prev}, tp_{cur}, tp_j) < T_{ir}$: This case represents that the user may slightly shake the phone or the user is going to stop walking. We first perform the record_segment() function, and then set $tp_{prev} = null$, $W = 0$, and $stable = false$. In this case, we take the bound tp_j as the next start point by setting $tp_{cur} = tp_j$, and then re-perform segment_finding(tp_{cur}).
   d) $\rho_{ir}(tp_{prev}, tp_{cur}, tp_j) \geq T_{ir}$: This case represents that we find a segment, and the segment is correlated with the one found in the last segment_finding(.) procedure. We perform the following four operations. (i) We set the variable W to be the time difference between tp_{cur} and tp_j. (ii) We update $tp_{prev} = tp_{cur}$ and $tp_{cur} = tp_j$. (iii) We perform the record_segment() function. (iv) Finally, we re-perform the segment_finding(.) procedure with the updated tp_{cur}. □

Note that in the segment_finding(.) procedure, the tp_{cur} is considered to be the start point of consecutive segments. When we cannot find a segment or two consecutive segments from tp_{cur}, the procedure will start searching from the next time point of the tp_{cur} in TP_y or TP_z. In our scheme, those values in A_y, A_z, TP_y, and TP_z, which are produced before the time tp_{prev}, will be eliminated from these sets automatically. Also note that in P1's step 4(c) (resp., P2's steps 2(c) and 2(d)), the condition $\rho_{ia}(tp_j, tp_k) \geq T_{ia}$ (resp., $\rho_{ia}(tp_{cur}, tp_j) \geq T_{ia}$) holds implicitly. Moreover, we remark the designs in P2's step 2(a) and 2(c). The P2's step 2(a) is to deal with the case that the regularity of linear acceleration values are greatly changed, e.g., the user may put the phone in pocket, and then pick up a phone call. So, we need to re-find segments from the next time point of the tp_{cur} in TP_y or TP_z. The P2's step 2(c) is to deal with the cases that when a user is going to stop walking or a user slightly shakes the phone. In these cases, the perceived linear acceleration values will not be so correlated. In our design, when the result of $\rho_{ir}(tp_{prev}, tp_{cur}, tp_j)$ is above a certain threshold (i.e., T_{ir_low}) but lower than a threshold (i.e., T_{ir}), we reset the procedure and re-find segments from the current boundary.

When performing the record_segment() function in P1's step 4(c) and P2's steps 2(c) and 2(d), we record the accumulated segments (identified between previous and current time instants when calling record_segment()) by variables seg_y or seg_z, respectively. After recording seg_y or seg_z, an update_SC() thread, which runs concurrently with the segment_finding(.) procedures of Y and Z axes, checks if the step count (i.e., the SC value) needs to be updated. However, we observe that when the perceived linear acceleration values are not so regular, the segment_finding(.) procedures of Y and Z axes may miss some segments. In this work, we take the result of Y or Z axes, which has more accumulated

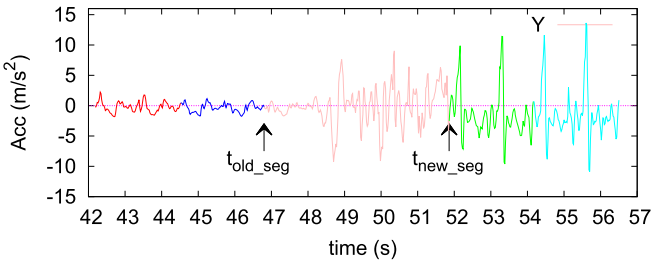

Fig. 6. An example when a user changes carrying ways.

step counts, as the final result. Given the user's current step count SC, the update_SC() thread works as follows:

1) If both segment_finding(.) procedures of Y and Z axes are in the stable state, the SC will be updated to $SC + \max\{seg_y \times 4, seg_z \times 2\}$ only when both of seg_y and seg_z are refreshed.
2) If only the Y's segment_finding(.) procedure is in the stable state, the SC will be updated to $SC + seg_y \times 4$ when seg_y is refreshed.
3) If only the Z's segment_finding(.) procedure is in the stable state, the SC will be updated to $SC + seg_z \times 2$ when seg_z is refreshed.

Note that when the stable flag is true, we say the corresponding segment_finding(.) procedure is in the *stable state*. We can see that the SC will be updated when four or two steps are identified.

In P2's step 2(a), the segment_finding(.) procedure of Y or Z axis starts a compensate_SC() thread. This thread is needed because that when a user changes the carrying way of the phone while walking, the segment_finding(.) procedure cannot identify segments during this time interval. For example, in Fig. 6, we can identify two segments during the time interval 42 s to 46.8 s. Then, the user changes the carrying way of the phone at time 46.8 s. We can see that the perceived linear acceleration values become irregular. In this example, the segment_finding(.) procedure cannot identify similar tendencies until time 51.9 s. Since the user is still walking, we need to compensate SC during the interval 46.8 s to 51.9 s. Let the t_{old_seg} and t_{new_seg} be the time instants when starting the thread and the start time point of the next consecutive segments, respectively (as shown in Fig. 6). After obtaining t_{old_seg} and t_{new_seg}, the compensate_SC() thread performs the following three steps.

1) This thread sets $TG = t_{new\_seg} - t_{old\_seg}$.
2) This thread compensates the segments on Y (resp., Z) axis by $\lfloor \frac{TG}{W_y} \rfloor$ (resp., $\lfloor \frac{TG}{W_z} \rfloor$).
3) This thread terminates itself.

Note that the W_y and W_z represent the instant W values when the segment_finding(.) procedures of Y and Z axes start the compensate_SC() thread, respectively. Also note that the compensated segments will be activated when the update_SC() thread updates SC. We further remark that when implementing, t_{old_seg} and t_{new_seg} can be obtained easily by recording the corresponding t_{cur} values. (The details are omitted here.)

V. Prototyping Results

In this work, we implement the proposed step counting algorithm on the Android platform. Recall that in the data collection phase, we design to gather the sensory readings from accelerometer every n millisecond. In this work, we set $n = 20$. However, according to the Android API, we can only configure the reporting rate of accelerometer to be "SENSOR_DELAY_FASTEST", and the actual rate depends on the used device. So, in our implementation, we use the concept of *linear interpolation* to realign the perceived linear acceleration and gravity values to follow a fixed interval (i.e., 20 ms). According to the training procedure in Section III-B, we set $(T^y_{usp}, T^y_{lsp}) = (T^z_{usp}, T^z_{lsp}) = (0.2, -0.2)$. Moreover, in our experiments, we set parameters δ_y and δ_z to be 0.6 s and 0.3 s, respectively.

We decide the threshold values T_{ia}, T_{ir_low}, and T_{ir} by numerical tests. In our tests, we first record the linear acceleration values while users walk by various carrying styles. Next, we use the collected logs as inputs to the implemented program, and observe the effects when we set different T_{ia}, T_{ir_low}, and T_{ir} values. Our observations are summarized as follows. First, when setting a large T_{ia}, the procedure P1 of segment_finding(.) may not be able to find segments that are similar enough to satisfy T_{ia}. Second, when setting a large T_{ir}, the similarity of nearby segments needs to be higher, and thus the segment_finding(.) procedures may repeatedly enter and leave the stable state. Third, when setting a large T_{ir_low}, the segment_finding(.) procedures will be harder to enter the stable state, and some segments may be missed. Fourth, when setting a small T_{ia} or T_{ir}, the segment_finding(.) procedures may find segments, which durations are smaller than the user's gaits. Finally, when setting a small T_{ir_low}, the segment_finding(.) procedures can enter the stable state easily, and then may have difficult to find similar tendencies. According to above discussions, improper threshold values may cause the following two phenomena. First, the segment_finding(.) procedures cannot stay in the stable state, and thus the system may consume computing power to traverse the sets TP_y and TP_z to re-find start points of new segments. Second, some segments may be missed or be unsynchronized with gaits, and thus the counted steps will be imprecise. Based on our tests and the collected logs, we choose $T_{ia} = 0.5$, $T_{ir\_low} = 0.3$, and $T_{ir} = 0.5$, which are considered to be good enough to relieve the above two phenomena. It is possible to reassign these three thresholds adaptively according the user's gaits when counting steps (which is considered to be one of our future works).

A. Some Segmenting Results

Fig. 7 shows six segmenting results. In Fig. 7, the pink lines represent the sets of raw data. A line segment, which is colored other than the pink, represents the identified set of data that belong to one segment. In these six experiments, the user walked 12 steps. Fig. 7(a) shows the result that the phone is held in the user's right hand and the user watches the screen while walking. The result indicates that we can find 3 and 6 segments in Y and Z axes, respectively. From Fig. 7(a), we can

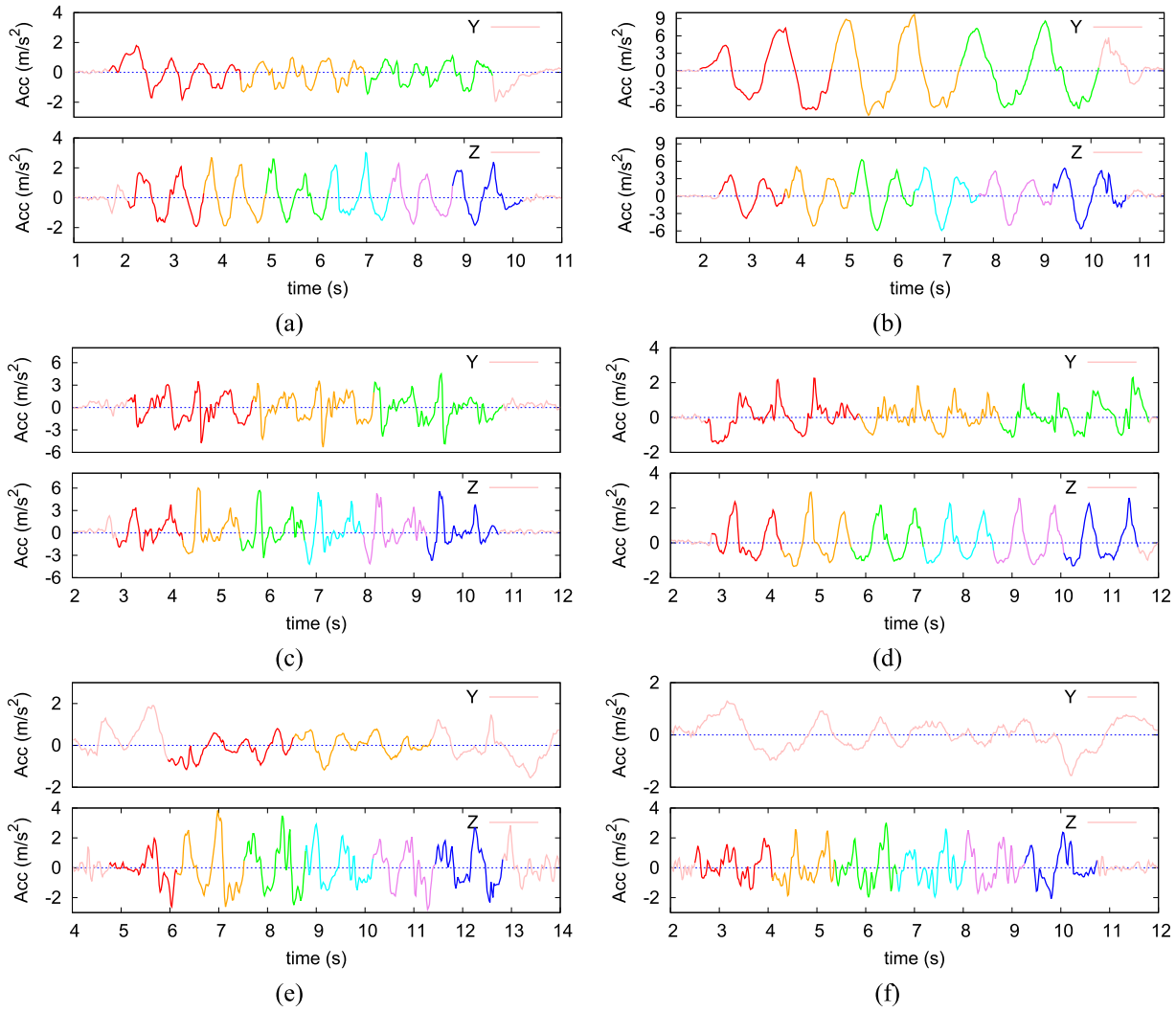

Fig. 7. Data analysis results of the user (a) holds the phone in the right hand and watches the screen, (b) holds the phone in the right hand and swings naturally with gaits, (c) puts the phone in the right pocket of her pant, (d) holds the phone in the right hand and talks, (e) puts the phone in a back bag, and (f) puts the phone in a handbag while walking.

see that the start times of the first segments of Y and Z axes and the stop times of the last segments of Y and Z axes are not synchronized. This phenomenon will not affect the step counting result since our algorithm considers the states of the segment_finding(.) procedures of Y and Z axes when counting steps. More specifically, in Fig. 7(a), the last segment of the Y axis is identified before the Z's last segment. Since both of the segment_finding(.) procedures of Y and Z axes are in the stable state, our algorithm will update the step counter at the time instant that the Z's segment_finding(.) procedure finds the last segment. Fig. 7(b)–(d) shows the results that the user holds the phone in the right hand and swing her hand naturally while walking, the user puts the phone in the pocket of her pant, and the user holds the phone and talks, respectively. From Fig. 7(b)–(d), we can see that our algorithm can successfully identify 3 and 6 segments in Y and Z axes, respectively. In the Fig. 7(c), when the phone puts in the pocket of the pant, the linear acceleration values are not so regular. This is because that these linear acceleration values are affected by the swings and vibrations of the user's leg while walking. To summarize, in the above four experiments (Fig. 7(a)–(d)), the phone can be considered to be "almost static" while the user walks, and the proposed scheme can successfully identify segments.

Next, Fig. 7(e) and Fig. 7(f) show the results that the user puts the phone in her back bag and handbag, respectively. From Fig. 7(e), we can see that the procedures of Z axis can identify six segments (i.e., 12 steps). But, the procedure of the Y axis can only identify two segments. We can see that the start time of the first segment of Y is not synchronized with the start time of the user's gait. Moreover, in Fig. 7(f), we can see that the segment_finding(.) procedure of Y axis cannot identify any segment. From the above discussions, the segmenting results of Y axis in Fig. 7(e) and Fig. 7(f) are not reliable. But, in these two cases, our algorithm can still count steps correctly based on the segmenting results of Z axis.

B. Experiment Results

In the following, we compare the calculated step counts of our designed scheme (installed on an HTC Sensation XL

TABLE I
EXPERIMENT RESULTS OF AVERAGED ERROR ON STEP COUNTS

The carrying ways	OUR on HTC Sensation XL	OUR on HTC Butterfly	Noom walk on HTC OneSV	Fitbit on HTC M8	S-health on Samsung Note3
m1: Hold the phone in the left hand and watch the screen	0.89%	0.22%	-3.89%	3.78%	-6.56%
m2: Hold the phone in the right hand and watch the screen	-0.22%	-0.22%	-18.89%	-5.78%	-0.89%
m3: Hold the phone in the right hand and watch the rotated 90° screen	0.00%	-0.67%	-0.33%	4.33%	-0.78%
m4: Hold the phone in the right hand and NOT swing the hand with gaits	0.00%	-0.22%	2.11%	3.22%	-0.22%
m5: Hold the phone in the right hand and swing naturally with gaits	0.89%	1.33%	-18.89%	5.44%	-22.19%
m6: Hold the phone in the right hand and talk	0.67%	-0.44%	0.44%	3.00%	-0.56%
m7: Put the phone in the right pocket of the pant	-0.44%	-1.33%	-31.78%	0.44%	1.67%
m8: Put the phone in the right pocket of the jacket	0.22%	0.22%	-3.89%	-5.67%	-0.22%
m9: Put the phone in the left pocket of the shirt	1.56%	-0.44%	0.78%	2.89%	-0.22%
m10: Put the phone in a handbag	-0.44%	-1.56%	-2.00%	2.89%	-1.56%
m11: Put the phone in the back bag	-0.22%	-0.22%	-0.33%	3.33%	0.11%
m12: Put the phone in a waist pouch	0.78%	1.11%	-9.89%	1.67%	-0.67%
Averaged error (over 12 carrying ways)	0.31%	-0.19%	-7.21%	1.63%	-2.67%

TABLE II
EXPERIMENT RESULTS OF AVERAGED ERROR ON WHEN USERS DYNAMICALLY CHANGE CARRYING WAYS

The designed scenarios	OUR on HTC Sensation XL	Noom walk on HTC OneSV	Fitbit on HTC M8	S-health on Samsung Note3
m5 → m2 → m3 → m6 → m7	-1.0%	-25.7%	-12.7%	-6.7%
m7 → m2 → m6 → m3 → m8	-2.0%	-29.0%	-26.7%	-2.3%
m9 → m2 → m7 → m5 → m6	-2.7%	-26.7%	-18.0%	-3.7%
m10 → m2 → m5 → m6 → m7	-3.0%	-26.7%	-17.3%	-2.3%
Averaged error (over 4 scenarios)	-2.2%	-27.0%	-18.7%	-3.8%

smartphone and an HTC Butterfly smartphone) with the Noom Walk APP [3] (installed on an HTC ONE SV smartphone), the Fitbit APP [2] (installed on an HTC M8 smartphone), and the S-health program of Sansumg Note3. We design 12 carrying ways, labeled as m1 to m12 (as shown in Table I). For each carrying way, we demand users to walk 300 steps. Table I shows the averaged errors of each carrying way by different step counting schemes. The results are summarized in Table I, where the result $x\%$ (reps. $-x\%$) represents that the final step count is $x\%$ more (resp. less) than the ground truth (i.e., 300 steps).

From Table I, we can see that the averaged error of the proposed scheme is within the range of $\pm 0.5\%$, which outperforms the other schemes. In our scheme, we may loss some segments if the perceived linear acceleration values are not so regular. In some cases, our scheme may find more segments when the derived window sizes (W) are too small. However, we can see that according to our experiment results, the possibilities of losing some segments or having improper window sizes are small. From the experiment results, the Noom Walk APP has the worst performance. During our experiments, we observe that the Noom Walk has less sensitivity on detecting steps. So, in most cases, the Noom Walk APP will report less step counts than the users walked. On the other hand, we observe that the Fitbit has a higher sensitivity. But, a small (up and down) shake may cause the Fitbit to add one more step. When the carrying ways are static, the S-health of Sansumg Note3 can have good performance on counting steps. However, we can see that when the user carries the phone and swings her hand naturally whil walking (i.e., scenario m5), the performance of the S-health of Sansumg Note3 will be greatly degraded.

In the following, we conduct experiments that the user will change carrying ways while walking. We design four scenarios. In each scenario, there will have a combination of five carrying ways as indicated in Table I. In a scenario, each user walks 100 steps in total. Each user walks 20 steps by a designated carrying way, and then changes to the next designated one. Table II shows the experiment results. From the results, we can see that the proposed scheme can still outperform the others. These experiments also demonstrate that the proposed compensate_SC() thread can effectively compensate step counts. The Noom walk and the Fitbit cannot perform well when users dynamically change carrying ways. The performance of S-health is almost similar to the one of the proposed scheme. However, when changing carrying ways, the S-health may unexpectedly stop counting steps. In Table II, we do not take those failure experiments of S-health into account.

C. Video Demonstration

In [1], we recorded four of our experiments by a video. In the video clip, the user walks by scenario s1 (for 100 steps) and carrying ways m3, m5, and m6 (for 80 steps), and we can see that the implemented program increments step counts effectively according to the user's gaits. Some discussions on the implemented step counting program are remarked in the video.

VI. Conclusion

In this paper, we have proposed a step counting algorithm for smartphone users. The proposed algorithm contains a data collection phase and a data analysis phase. The data collection phase first collects and transforms sensory data, and then finds possible start points of segments. The data analysis phase segments the gathered raw data from data collection phase, and tries to identify possible correlated segments. Those correlated segments can be taken as user's steps. In this work, we implement the designed scheme on the Android platform. The experiment results indicate that the designed scheme can outperform other schemes when the user carries the phone in a static manner. Besides, for the cases that the phone is arbitrarily operated while walking, the proposed scheme can also count steps effectively. In the future, we are going to develop an APP, which adopts the designed scheme to trace visitors' trajectories in our campus. Besides, we are now going to apply the designed scheme to analyze sport postures.

References

[1] *The Demo Video*. [Online]. Available: http://youtu.be/GCzAALQSRHg, accessed Nov. 25, 2014.

[2] *Fitbit*. [Online]. Available: https://play.google.com/store/apps/details?id=com.fitbit.FitbitMobile, accessed Nov. 25, 2014.

[3] *Noom Walk*. [Online]. Available: https://play.google.com/store/apps/details?id=com.noom.walk, accessed Nov. 25, 2014.

[4] *Runtastic Pedometer*. [Online]. Available: https://play.google.com/store/apps/details?id=com.runtastic.android.pedometer.lite, accessed Nov. 25, 2014.

[5] *The Walk: Fitness Tracker and Game*. [Online]. Available: https://itunes.apple.com/us/app/walk-fitness-tracker-game/id678971662?mt=8, accessed Nov. 25, 2014.

[6] *Zombies, Run.* [Online]. Available: https://itunes.apple.com/us/app/zombies-run!/id503519713?mt=8, accessed Nov. 25, 2014.

[7] M. Alzantot and M. Youssef, "UPTIME: Ubiquitous pedestrian tracking using mobile phones," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2012, pp. 3204–3209.

[8] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 775–784.

[9] H. Bao and W.-C. Wong, "An indoor dead-reckoning algorithm with map matching," in *Proc. 9th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jul. 2013, pp. 1534–1539.

[10] O. Bebek *et al.*, "Personal navigation via shoe mounted inertial measurement units," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2010, pp. 1052–1058.

[11] A. Brajdic and R. Harle, "Walk detection and step counting on unconstrained smartphones," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. (UbiComp)*, 2013, pp. 225–234.

[12] J. Chon and H. Cha, "LifeMap: A smartphone-based context provider for location-based services," *IEEE Pervasive Comput.*, vol. 10, no. 2, pp. 58–67, Apr./Jun. 2011.

[13] L. Hu and D. Evans, "Localization for mobile sensor networks," in *Proc. 10th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2004, pp. 45–57.

[14] W.-Y. Hu, J.-L. Lu, S. Jiang, W. Shu, and M.-Y. Wu, "WiBEST: A hybrid personal indoor positioning system," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2013, pp. 2149–2154.

[15] H.-J. Jang, J. W. Kim, and D. H. Hwang, "Robust step detection method for pedestrian navigation systems," *IET Electron. Lett.*, vol. 43, no. 14, pp. 749–751, Jul. 2007.

[16] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara, "A comparison of pedestrian dead-reckoning algorithms using a low-cost MEMS IMU," in *Proc. IEEE Int. Symp. Intell. Signal Process. (WISP)*, Aug. 2009, pp. 37–42.

[17] Y. Kim, H. Shin, and H. Cha, "Smartphone-based Wi-Fi tracking system exploiting the RSS peak to overcome the RSS variance problem," *Pervasive Mobile Comput.*, vol. 9, no. 3, pp. 406–420, 2013.

[18] C.-C. Lo, C.-P. Chiu, Y.-C. Tseng, S.-A. Chang, and L.-C. Kuo, "A Walking Velocity Update technique for pedestrian dead-reckoning applications," in *Proc. IEEE 22nd Int. Symp. Personal Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2011, pp. 1249–1253.

[19] M. Mladenov and M. Mock, "A step counter service for java-enabled devices using a built-in accelerometer," in *Proc. ACM Int. Workshop Context-Aware Middleware Services (CAMS)*, 2009, pp. 1–5.

[20] L. Ojeda and J. Borenstein, "Non-GPS navigation with the personal dead-reckoning system," in *Proc. Defense Secur. Symp. (DSS)*, Apr. 2007, pp. 1–11.

[21] K. Park, H. Shin, and H. Cha, "Smartphone-based pedestrian tracking in indoor corridor environments," *Personal Ubiquitous Comput.*, vol. 17, no. 2, pp. 359–370, 2013.

[22] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen, "Zee: Zero-effort crowdsourcing for indoor localization," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2012, pp. 293–304.

[23] H. Wang, S. Sen, A. Elgohary, M. Farid, M. Youssef, and R. R. Choudhury, "No need to war-drive: Unsupervised indoor localization," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, 2012, pp. 197–210.

[24] R. Zhang, A. Bannoura, F. Hoflinger, L. M. Reindl, and C. Schindelhauer, "Indoor localization using a smart phone," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, Feb. 2013, pp. 38–42.

[25] R. Zhang and L. M. Reindl, "Pedestrian motion based inertial sensor fusion by a modified complementary separate-bias Kalman filter," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, Feb. 2011, pp. 209–213.

Meng-Shiuan Pan (M'08) received the B.S. degree from National Chung Cheng University, Chiayi, Taiwan, in 2001, the M.S. degree from National Tsing Hua University, Hsinchu, Taiwan, in 2003, and the Ph.D. degree from National Chiao Tung University, Hsinchu, in 2008. After receiving his Ph.D. degree, he worked with HTMM Corporation, Hsinchu, for three years, and dedicated in designing and implementing 3GPP R99/R5/R6 layer 2/3 protocol stacks. Since 2011, he has been an Assistant Professor with the Department of Computer Science and Information Engineering, Tamkang University, Taipei, Taiwan. He is as an Associate Editor of the *International Journal of Distributed Sensor Networks*. His research interests include wireless sensor network, mobile computing, and LTE-A networks.

Hsueh-Wei Lin received the B.S. and M.S. degrees from Tamkang University, Taipei, Taiwan, in 2012 and 2014, respectively. His research interests include wireless sensor network and wearable computing.