MobiGesture: Mobility-Aware Hand Gesture Recognition for Healthcare

Hongyang Zhao Yongsen Ma Shuangquan Wang Amanda Watson Gang Zhou
Computer Science Department, College of William and Mary
Email: {hyzhao, yma}@cs.wm.edu, {swang10, aawatson}@email.wm.edu, gzhou@cs.wm.edu

***Abstract*—Accurate recognition of hand gestures while moving is still a significant challenge, which prevents the wide use of existing gesture recognition technology. In this paper, we propose a novel mobility-aware hand gesture segmentation algorithm to detect and segment hand gestures. We also propose a Convolutional Neural Network (CNN) to classify hand gestures with mobility noises. Based on the segmentation and classification algorithms, we develop MobiGesture, a mobility-aware hand gesture recognition system for healthcare. For the leave-one-subject-out cross-validation test, experiments with human subjects show that the proposed segmentation algorithm achieves 94.0% precision, and 91.2% recall when the user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when the user is standing, walking and jogging, respectively.**

***Index Terms*—Mobility-Aware, Gesture Recognition, Gesture Segmentation, Convolutional Neural Network**

I. Introduction

Regular mobility, such as walking or jogging, is one of the most effective ways to promote health and well-being. It helps improve overall health and reduces the risk of many health problems, such as diabetes, cardiovascular disease, and osteoarthritis [1] [2] [3]. In addition, regular mobility can also improve depression, cognitive function, vision problems, and lower-body function [4] [5]. According to the evidence-based Physical Activity Guidelines released by World Health Organization [6] and the U.S. government [7], adults aged 18-64 should do at least 150 minutes of moderate-intensity or 75 minutes of vigorous-intensity aerobic activity per week, or an equivalent combination of both to keep healthy.

Nowadays, lots of people like to listen to music on their smartphones while they do aerobic activity, such as walking or jogging. Many smartphone apps track users' workouts, play music, and even match the tempo of the songs to users' paces, such as Nike+ Run Club [8], RunKeeper [9], MapMyRun [10]. However, it is inconvenient for the users to interact with these apps while walking or jogging. To change the music, users need to slow down, take out the smartphone, and then change the music. This is troublesome. Instead, it is more convenient for users to use gestures to control the music. Unlike traditional touchscreen-based interaction, hand gestures can simplify the interaction with a smartphone by reducing the need to take out the smartphone and slow down the pace.

When a user is moving, gesture recognition is difficult. The first reason is that hand swinging motions during walking or jogging are mixed with the hand gestures. It is hard to classify if a hand movement comes from hand swinging motions or a hand gesture. In addition, when the user performs a hand gesture while moving, the hand movement is a combination of the hand gesture and the body movement. The mobility noise caused by the body movement reduces the accuracy of the hand gesture recognition. Therefore, it is hard to recognize hand gestures when the user is moving. To solve the gesture recognition problem when the user is walking or jogging, two research questions need to be answered: (1) How to segment the hand gestures when the user is moving? (2) How to accurately classify the hand gestures with mobility noises?

In order to answer the first research question, we first apply an AdaBoost Classifier to classify the current body movement into moving or non-moving. If the user is not moving, we apply a threshold-based segmentation algorithm to segment the hand gestures. If the user is moving, the sensor readings are periodic and self-correlated. So, we propose a novel self-correlation metric to evaluate the self-correlation of the sensor readings. If the sensor readings are not self-correlated at the moving frequency, we regard it as a potential gesture sample. Then, a moving segmentation algorithm is applied to segment the hand gestures.

In order to answer the second research question, we design a CNN model to classify the hand gestures with mobility noises. We apply a batch normalization layer, a dropout layer, a max-pooling layer and L2 regularization to overcome overfitting and handle mobility noises.

In addition, we integrate the gesture segmentation and classification algorithms into a system called, MobiGesture. For the leave-one-subject-out cross-validation test, experiments with human subjects show that the proposed segmentation algorithm accurately segments the hand gestures with 94.% precision and 91.2% recall when a user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when a user is standing, walking and jogging, respectively.

As far as we know, two efforts have been made to study the gesture recognition problem when a user is moving. Park et al. [11] propose a Multi-situation HMM architecture. They train a HMM model for each pair of hand gesture and mobility situation. As the authors define 8 hand gestures and 4 mobility situations, 32 HMM models are trained in total. Given a hand gesture, they apply the Viterbi algorithm [12] to calculate the likelihood of each HMM model. The HMM model with the highest likelihood is selected as the classified gesture. As the

number of the hand gestures and/or the number of the mobility situations increases, their computational cost increases dramatically. Different from their work, we only train one CNN model, which consumes much less computational power and time. In addition, evaluation results show that our CNN model performs better than Multi-situation HMM on gesture classification under leave-one-subject-out cross-validation test. The second effort comes from Murao et al. [13]. They propose a combined-activity recognition system. This system first classifies user movement into one of three categories: postures (e.g., sitting), behaviors (e.g., walking), and gestures (e.g., a punch). Then, Dynamic Time Warping (DTW) is applied to recognize hand gestures for the specific category. However, their system requires five sensors to be attached to the human body for gesture recognition. Instead, we only use one sensor, and hence are less intrusive.

We summarize our contributions as follows:

1) We propose a novel mobility-aware gesture segmentation algorithm to detect and segment hand gestures.
2) We design a CNN model to classify hand gestures. This CNN model conquers mobility noises and avoids overfitting.
3) We integrate the gesture segmentation and classification algorithms into a system, MobiGesture. Our experiments results show that the proposed segmentation algorithm achieves 94.0% precision and 91.2% recall when the user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when the user is standing, walking and jogging, respectively.

The remainder of this paper is organized as follows. First, we present the motivation in Section II. Then, we introduce the system architecture in Section III. We present our mobility-aware segmentation algorithm in Section IV, and CNN model in Section V. In Section VI, we evaluate the system performance. We summarize the related works in Section VII. Finally, we draw our conclusion in Section VIII.

II. Motivation

Hand gestures can help users interact with various mobile applications on smartphones in mobile situations. One common scenario is to control a music app while walking or jogging. We define our hand gestures to be suitable for music control in Section II-A. Based on these defined gestures, we introduce the challenge of gesture recognition when the user is walking or jogging in Section II-B. Finally, we present our data collection and the data set in Section II-C. This data set is used for performance evaluation during the rest of the paper.

A. Gesture Definition

There has been substantial research on gesture recognition. Some works define gestures according to application scenarios, such as gestures in daily life [14], or repetitive motions in very specific activities [15], while others define gestures casually [11]. In this paper, we carefully define the hand gestures that are suitable for controlling a music app. Typically,

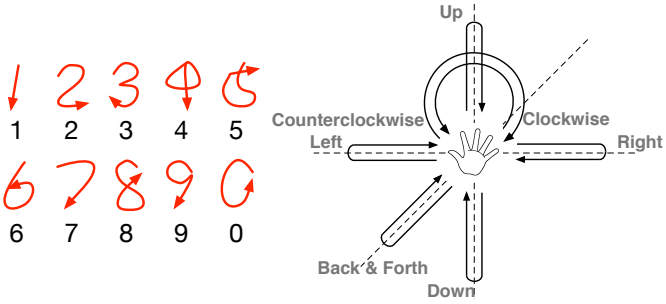


Fig. 1. 17 defined gestures for remote control

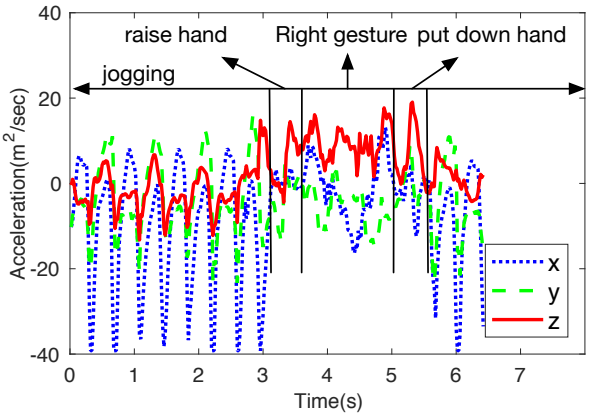


Fig. 2. Accelerometer readings of a Right gesture when a user is jogging

a music app provides the following functions to control the music: next track, previous track, volume up, volume down, play/pause, repeat on/off, shuffle on/off. Therefore, we define seven gestures corresponding to these seven functions. At the beginning, the user extends his/her hand in front of his/her body. Then he/she moves towards a certain direction and moves back to the starting point again.

These seven gestures are illustrated in Fig. 1 and are defined as follows: (1) Left gesture: move left and then move back to the starting point; (2) Right gesture: move right and then move back to the starting point; (3) Up gesture: move up and then move back to the starting point; (4) Down gesture: move down and then move back to the starting point; (5) Back&Forth gesture: move to shoulder and then extend again to the starting point; (6) Clockwise gesture: draw a clockwise circle; (7) Counterclockwise gesture: draw an counterclockwise circle. In addition, we define 10 number gestures as shown in Fig. 1 to select a specific song in the music app.

B. Challenge of Gesture Recognition under Mobility

To recognize hand gestures, a typical gesture processing pipeline consists of three steps: (1) **detect** a hand gesture from a sequence of hand movements; (2) **segment** the hand gesture; (3) **classify** the segmented hand gesture. When it comes to a mobile situation, the noises caused by body movements present several practical challenges for these three steps.

First, it is hard to **detect** gestures while a user is moving. While the user is standing without performing any gesture, the

TABLE I
CHARACTERISTICS OF FIVE PARTICIPANTS

Human Subject No.	Gender	Age	Height(cm)	Weight(kg)
1	male	29	174	62
2	female	27	167	55
3	male	28	180	73
4	male	39	170	87
5	male	30	171	68

accelerometer readings keep stable. When the user performs a gesture such as the Right gesture, the accelerometer readings change dramatically. Therefore, it is easy to detect a gesture by measuring the amplitude or deviation of the sensor readings. However, when it comes to a jogging scenario, a Right gesture and hand swinging motions are mixed together as shown in Fig. 2. Therefore, it is hard to tell whether a hand movement comes from the hand swinging motions or a hand gesture.

The second challenge is that it is hard to **segment** hand gestures while the user is moving. To perform a hand gesture while walking or jogging, the user needs to raise his/her hand, perform a gesture, and then put down his/her hand. To segment hand gestures while the user is moving, we need to not only filter out hand swinging motions caused by body movements, but also accurately exclude the hand-raising and hand-lowering movements. If the starting point and end point of a hand gesture is not precisely determined, it is hard to classify hand gestures accurately. As shown in Fig. 2, it is difficult to find the starting point and end point of a Right gesture while jogging.

The third challenge is that it is hard to **classify** hand gestures when a user is moving. After gesture segmentation, a segmented hand gesture sample includes not only the gesture the user performs, but also the noises caused by the body movements. Additionally, when the user performs a hand gesture while walking or jogging, (s)he needs to keep the walking/jogging pace while performing this gesture. The effort to keep the moving pace influences the shape of the hand gesture that the user performs. Therefore, the hand gesture performed when the user is standing is slightly different from the same type of hand gesture performed when the user is walking or jogging. Both the mobility noises and the gesture differences reduce the accuracy of gesture classification.

C. Dataset

We used a UG wristband [16] to collect 17 hand gestures from 5 human subjects, which is shown in Fig. 3. The UG wristband sampled the accelerometer and gyroscope readings at 50 Hz. The data collection experiment contained three independent steps. (1) Each participant performed each gesture 10 times while standing. (2) Each participant performed each gesture 10 times while walking on a treadmill. (3) Each participant performed each gesture 10 times while jogging on a treadmill. In total, 2550 hand gestures were collected. While walking or jogging on a treadmill, different participants tended to walk or jog at different speeds. In our experiment, the speed of walking ranged from 2 miles/hour to 3 miles/hour, and the speed of jogging ranged from 4 miles/hour to 6 miles/hour. We took video of each participant as they completed these tasks to serve as ground truth. The characteristics of our participants are shown in Table I.

III. SYSTEM ARCHITECTURE

The system architecture of the MobiGesture is shown in Fig. 4. We apply a novel mobility-aware segmentation module to partition the raw accelerometer and gyroscope readings into segments so that each segment contains one complete hand gesture. In the mobility-aware segmentation module, we first detect whether or not the user is moving. We extract a series of time-domain and frequency-domain features from accelerometer readings and apply an AdaBoost Classifier to classify the current body movement into moving or non-moving. If the user is not moving, sensor readings are clean and do not contain any mobility noise. In this case, we apply a simple threshold-based segmentation algorithm to segment the hand gestures.

On the other hand, if the user is walking or jogging, the sensor readings are periodic and self-correlated. We perform Fast Fourier Transform (FFT) on accelerometer readings and compute the dominant frequency, which is the frequency of walking or jogging. Based on the dominant frequency, we propose a novel self-correlation metric, SC. This metric represents the self-correlation characteristics of accelerometer readings at the given frequency. When the user is walking or jogging, the sensor readings are self-correlated at the dominant frequency. Once the sensor readings are no longer self-correlated at the dominant frequency, we regard it as a potential gesture sample. Then, a moving segmentation algorithm is applied to partition the accelerometer and gyroscope readings into segments based on SC metric.

As the 17 predefined gestures are different from each other and users tend to perform the gestures at different speeds, the duration of each gesture is different. Therefore, the size of each segment is different. We apply a Cubic Spline Interpolation algorithm [17] to rescale the size of each segment so that each segment contains the same data points. Finally, we design a 9-layer Convolutional Neural Network to recognize hand gestures. The Convolutional Neural Network is designed to be anti-overfitting and robust to mobility noises.

IV. MOBILITY-AWARE SEGMENTATION

A simple way to segment hand gestures from a sequence of hand movements is to use a hand-controlled button to clearly indicate the starting point and the end point of each individual gesture. However, in order to do so, the user must wear an external button on their fingers or hold it in their hands, which is obtrusive and burdensome. Another way is to segment gestures automatically. The motion data are automatically partitioned into non-overlapping, meaningful segments, such that each segment contains one complete gesture. Automatic segmentation when a user is moving faces a few challenges. First, when the user is moving, the hand gestures are mixed with the mobility noises, which leads to inaccurate segmentation. In addition, the segmentation should extract the hand movement caused by the hand gestures rather than the hand

(a) Placement of a UG wristband

(b) Coordinate System of a UG wristband

Fig. 3. UG Wristband

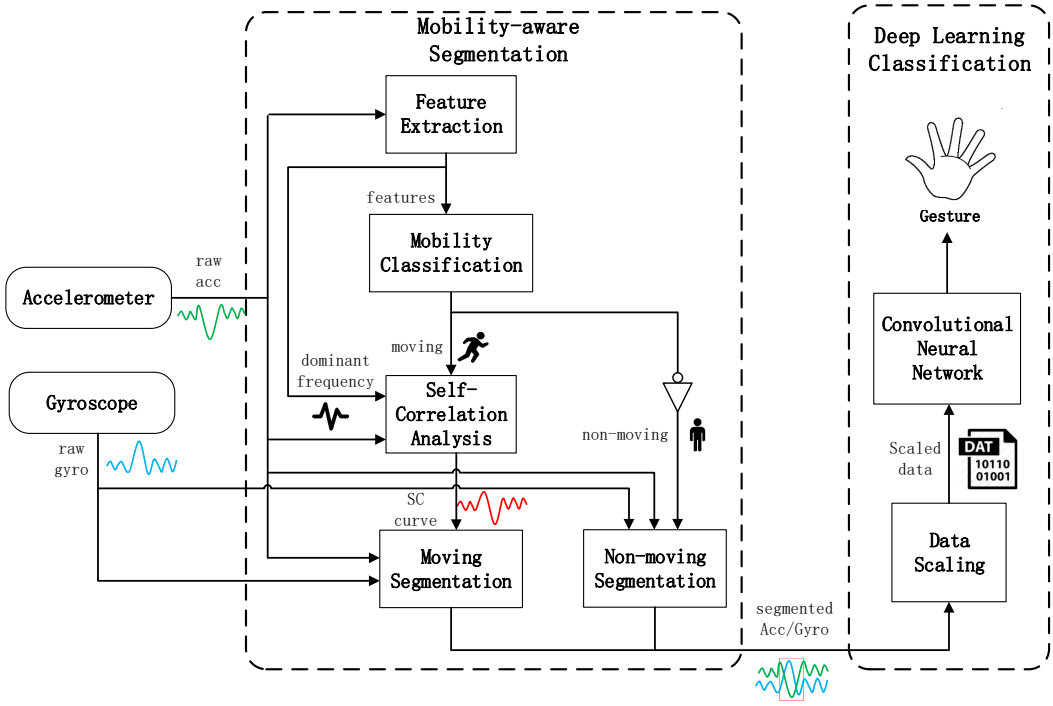


Fig. 4. System Architecture

movement caused by the body movement. Otherwise, the extracted segments contain non-gesture noises, or miss useful gesture information, which leads to inaccurate classification. To deal with these challenges, we propose a mobility-aware segmentation algorithm. we first classify the body movement into non-moving or moving. Then, we propose a non-moving segmentation algorithm and a moving segmentation algorithm to segment hand gestures for two different moving scenarios.

A. Feature Extraction

It is difficult to accurately detect the mobility situation solely based on a wristband. The reason is that the sensors in the wristband measure the combination of hand motion, gravity, and body movement. In order to accurately detect if the user is moving or not, additional sensors that are tightly attached on the body are required. However, this requirement is intrusive.

Instead of attaching additional sensors on the body, we infer the body movement based on the sensor readings from the wristband. When the user is walking, the hands are pointing to the ground with the palm facing towards the user. When the user is jogging, the hands are pointing forward with the palm facing towards the user. The orientation of the hand is fixed and stable during walking or jogging. The sporadic occurrence of a hand gesture influences the orientation of the hand in a short time. However, the orientation of the hand is stable for most of the time during walking or jogging. This motivates us to use the orientation of the hand to infer the body movement. In addition, when a user is walking or jogging, the user swings his/her hands periodically. The sporadic occurrence of a hand gesture does not influence the dominant frequency of walking or jogging. This motivates us to use the frequency of hand swinging motions to infer the body movement.

We apply a sliding window with an overlapping of 50% for the accelerometer readings. The window size is 5 seconds. We compute a series of time-domain and frequency-domain features for each time window.

For the time-domain features, we compute the mean of the accelerometer readings of the X-axis, Y-axis, and Z-axis accordingly, the mean of the pitch, and the mean of the roll to represent the orientation of the hand for each time window. The pitch and roll are computed as

$$Pitch = \arctan\left(\frac{Acc_y}{\sqrt{(Acc_x)^2 + (Acc_z)^2}}\right), \quad (1)$$

$$Roll = -\arctan\left(\frac{Acc_x}{Acc_z}\right), \quad (2)$$

where Acc_x, Acc_y, Acc_z are the accelerometer readings of the X-axis, Y-axis, and Z-axis for each time window.

For the frequency domain, we first compute the amplitude of accelerometer readings as

$$Acc = \sqrt{(Acc_x)^2 + (Acc_y)^2 + (Acc_z)^2}, \quad (3)$$

where Acc_x, Acc_y, Acc_z are the accelerometer readings of the X-axis, Y-axis, and Z-axis for each time window. Then, we perform Fast Fourier transform (FFT) for all the amplitude of the accelerometer readings within each time window. We find the dominant frequency, which has the largest amplitude in the frequency domain. Finally, the dominant frequency and the amplitude of the dominant frequency are chosen as frequency-domain features.

TABLE II
COMPARISON OF MACHINE LEARNING ALGORITHMS FOR MOBILITY CLASSIFICATION

Test	Algorithm	Precision	Recall	F-Measure
5-fold	AdaBoost	93.5%	93.5%	93.5%
	Naive Bayes	91.6%	91.4%	91.5%
	SVM	93.5%	93.5%	93.3%
	J48	96.0%	96.0%	96.0%
	RandomForest	**96.9%**	**96.9%**	**96.9%**
LOSO	AdaBoost	**94.9%**	**94.6%**	**94.4%**
	Naive Bayes	93.6%	91.4%	91.5%
	SVM	93.6%	92.5%	92.2%
	J48	91.0%	88.7%	89.0%
	RandomForest	93.7%	92.2%	92.2%

B. Mobility Classification

We apply the WEKA machine-learning suite [18] to train five commonly used classifiers. The classifiers include AdaBoost (run for 100 iterations), Naive Bayes, SVM (with polynomial kernels), J48 (equivalent to C4.5 [19]), and Random Forests (100 trees, 4 random features each). To evaluate the performance of the proposed algorithms, we apply two tests: 5-fold cross-validation and leave-one-subject-out (LOSO) cross-validation. The 5-fold cross-validation test uses all the gesture data to form the dataset. It partitions the dataset into 5 randomly chosen subsets of equal size. Four subsets are used to train the model. The remaining one is used to validate the model. This process is repeated 5 times such that each subset is used exactly once for validation. The leave-one-subject-out cross-validation test uses the gesture data from four subjects to train the classification model, and then applies this model to test the gesture samples from the remaining subject. Precision, recall, and F-measure are considered as the evaluation metrics.

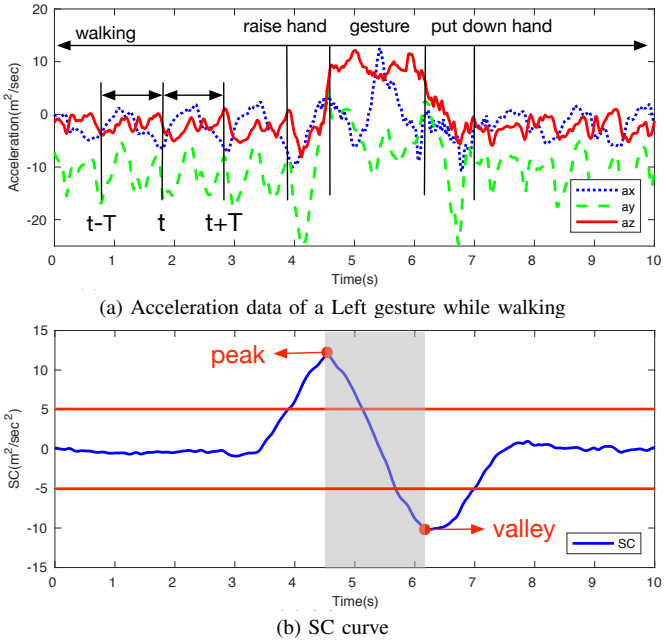
The classification results for these five algorithms are shown in Table II. Under the 5-fold cross-validation test, RandomForest performs the best. The precision, recall, and F-measure are 96.9%, 96.9%, and 96.9%, respectively. Under the leave-one-subject-out cross-validation test, AdaBoost performs the best. The precision, recall, and F-measure are 94.9%, 94.6%, and 94.4%, respectively. We favor the leave-one-subject-out cross-validation test over the 5-fold cross-validation test to avoid overfitting. Therefore, we choose AdaBoost classifier to classify the body movement.

C. Non-Moving Segmentation

When the user is not moving, we apply a lightweight threshold-based detection method to identify the starting and end points of the hand gestures. To characterize a user's hand movement (HM), a detection metric is defined using the gyroscope sensor readings as

$$HM = \sqrt{Gyro_x^2 + Gyro_y^2 + Gyro_z^2}, \tag{4}$$

where $Gyro_x, Gyro_y, Gyro_z$ are the gyroscope readings of the X-axis, Y-axis, and Z-axis. When the user's hand is stationary, the HM is very close to zero. The faster a hand moves, the larger the HM is. When the HM is larger than a threshold, i.e. 50 degree/second, we regard it as the starting point of a hand movement. Once the HM is smaller than this threshold for a certain period of time, i.e. 200 ms, we regard it as the end point of the hand movement. The time threshold is necessary. Without it, the HM may fall below this threshold occasionally, leading to unexpected splitting of this gesture [20] [21]. As a gesture does not last shorter than 260 ms or longer than 2.7 seconds in our dataset, we drop a segment if the length of this segment is shorter than 260 ms or longer than 2.7 seconds.



(a) Acceleration data of a Left gesture while walking

(b) SC curve

Fig. 5. Segmentation when a user is moving

D. Self-Correlation Analysis

When the user is walking or jogging, the sensor readings are periodic and self-correlated at the frequency of walking or jogging. Once the user performs a gesture while walking or jogging, the sensor readings are neither periodic nor self-correlated. Based on this observation, we propose a novel self-correlation metric SC to measure the self-correlation of the accelerometer readings as

$$SC(t) = \sum_{i \in \{x,y,z\}} \sum_{j=1}^{T} \left[Acc_i\left(t+j\right) - Acc_i\left(t+j-T-1\right) \right] / T, \tag{5}$$

where Acc_i $(i \in \{x, y, z\})$ are the accelerometer readings of the X-axis, Y-axis, and Z-axis. T is the cycle of the walking or jogging, which is computed as the inverse of the dominant frequency. t is the current time. If the accelerometer readings are self-correlated at the dominant frequency, the SC is very close to zero. If the accelerometer readings are not self-correlated at the dominant frequency, the SC is either a large positive value or a large negative value. Fig. 5(a) shows the accelerometer readings of a Left gesture when a user is walking. The computed SC curve is in Fig. 5(b).

From Fig. 5(a) and (b), we find that the SC is very close to zero when the user swings his/her hand during walking. The

SC begins to increase when the user raises his/her hand. The peak of the SC occurs when the user finishes raising his/her hand and begins to perform a Left gesture. The valley of the SC occurs when the user finishes performing a Left gesture and begins to put down his/her hand. After the user puts down the hand and continues to swing the hand, the SC goes back to zero. We find that the peak and the valley of the SC curve are good indicators of the starting point and the end point of the Left gesture. The reason is that when the user raises his/her hand or puts down his/her hand, the orientation of his/her hand changes a lot. This change greatly increases or reduces the SC metric.

When the user is walking, the hand is pointing to the ground. Impacted by the force of gravity, the accelerometer readings of the Y-axis are always negative. When the user is jogging, the hand is pointing forward with the palm facing towards the user. In this case, the accelerometer readings of the X-axis are impacted by gravity and always have negative values. However, when the user raises his/her hand and performs the gesture, the palm faces the ground. The accelerometer readings of the Z axis are impacted by gravity and have positive values. Therefore, when the user is walking or jogging, the sum of the accelerometer readings in 3 axes are always negative. When the user raises his/her hand, the sum of the accelerometer readings increases and reaches the maximum right after raising hand. As the SC is computed by the difference of the accelerometer readings in two adjacent time window (window size is the cycle of walking or jogging), SC reaches the peak right after raising hand. Similarly, SC reaches the valley right after putting down hand. Therefore, we use the peak and the valley of the SC curve as the starting point and the end point of the Left gesture.

E. Moving Segmentation

We segment the hand gestures when the user is moving by searching for the peak and valley of the SC. We compute the SC metric from the accelerometer readings. If the SC value is larger than 5 m^2/sec^2, we start to search for the peak of the SC within a 4 second time window. Once the peak is found, we regard it as the starting point of the hand gesture, and begin to search for the valley of the SC. We define the valley of the SC to be the smallest SC within a 4 second time window and is lower than a threshold, -5 m^2/sec^2. Once the SC valley is found, we regard it as the end point of the hand gesture. We extract the accelerometer and gyroscope readings between the starting point and the end point as the segment. As a gesture does not last longer than 2.7 seconds in our dataset, we drop a segment if we cannot find the valley of the SC after the peak of the SC for 2.7 seconds.

Fig. 6 and Fig. 7 show the performance of the moving segmentation under different window sizes and different SC thresholds accordingly. Three evaluation metrics are considered: precision, recall, and F-measure. As the window size or the SC threshold increases, the segmentation precision increases and the recall decreases. When the window size is 4 s and the SC threshold is 5 m^2/sec^2, the F-measure is at its highest value: 93.7%. Therefore, we choose 4 s as the time window size and 5 m^2/sec^2 as the SC threshold to segment the hand gestures when the user is moving.

TABLE III
COMPARISON OF THE GESTURE SEGMENTATION PERFORMANCE

Scenario	Algorithm	Precision	Recall	F-Measure
Non-moving	MobiGesture 5-fold	92.2%	93.5%	92.8%
	MobiGesture LOSO	92.6%	90.1%	91.3%
	E-gesture	98.0%	97.1%	97.5%
Moving	MobiGesture 5-fold	93.5%	94.6%	93.7%
	MobiGesture LOSO	94.0%	91.2%	92.2%
	E-gesture	8.5%	18.3%	11.3%

F. Performance

We compare our gesture segmentation algorithm with E-gesture [11], which is the state-of-the-art. E-gesture segments the hand gestures based on the amplitude of the gyroscope readings. A hand gesture is triggered if the amplitude of the gyroscope readings is higher than 25 degree/sec. The triggered gesture is assumed to have ended if the amplitude of the gyroscope readings is lower than 25 degree/sec for 400 ms. Different from E-gesture, we first apply the AdaBoost classifier to classify the body movement into moving or non-moving. Then, we apply two different segmentation algorithms to segment the hand gestures accordingly.

We evaluate the segmentation accuracy by checking the overlap between a segment and a hand gesture. If the middle of a hand gesture lies in a segment, this gesture is correctly segmented by that segment. The performance of MobiGesture and E-Gesture are shown in Table III. From the table, we find that MobiGesture performs stably in both moving scenarios. The F-measure in two moving scenarios are around 92%. E-gesture performs well when the user is not moving. However, it performs poorly when the user is moving. When the user is moving, the F-measure of E-gesture is only 11.3%. The possible reasons are: (1) the sensors in their wristband are different from ours; (2) their predefined hand gestures are different from ours.

We change the threshold of E-gesture from 25 degree/sec to 250 degree/sec with a step of 25 degree/sec and evaluate the performance of E-gesture. Fig. 8 shows the F-measure of E-gesture under different thresholds. When the threshold is 175 degree/sec, the F-measure of E-gesture in moving scenario is at its highest value: 74.5%. This is still much lower than the accuracy of our moving segmentation algorithm: 93.7% under 5-folder cross-validation test, and 92.2% under leave-one-subject-out cross-validation test. Therefore, their segmentation algorithm can not accurately segment the hand gestures when the user is moving. Their solution is not general enough to be extended to the new gestures and hardware platform.

When a user is standing without performing any gesture, the gyroscope readings are close to zero. The amplitude of the gyroscope readings is a good measurement to segment the hand gestures. Both E-gesture and MobiGesture use the amplitude of the gyroscope readings to segment the hand gestures. Therefore, both algorithms perform well when the user

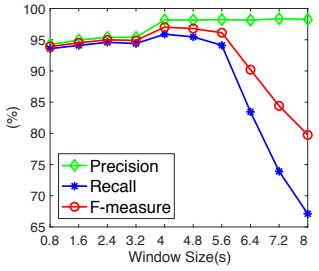


Fig. 6. Segmentation precision, recall, and F-measure under different window sizes

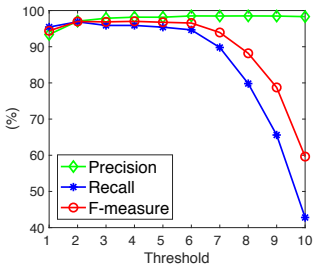


Fig. 7. Segmentation precision, recall, and F-measure under different SC thresholds

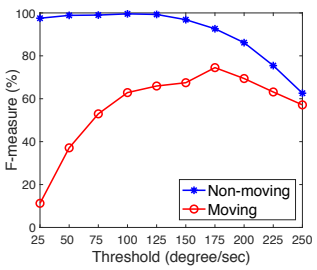


Fig. 8. F-measure of E-gesture under different thresholds

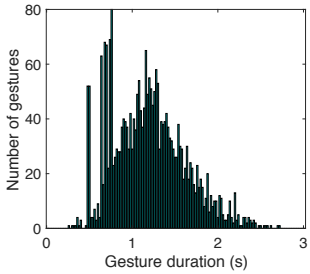


Fig. 9. The distribution of the gesture duration in our dataset

is standing. However, when a user is moving, the hand gestures are mixed with the hand swinging motions. With E-gesture, the amplitude of the gyroscope readings can not differentiate the hand gestures from the hand swinging motions. Therefore, E-gesture performs poorly in the moving scenarios. Instead, we utilize the self-correlation of the sensor readings to segment the hand gestures, which takes the hand swinging motions into consideration. Therefore, the proposed segmentation algorithm accurately distinguishes the hand gestures from the hand swinging motions.

V. Deep Learning Classification

Two approaches are popular for classifying hand gestures. One is to use conventional machine learning classifiers, such as Naive Bayes [22], Random Forest [15], and Support Vector Machines [23]. The other is to use sequential analysis algorithms, such as Hidden Markov Model (HMM) [11] and Dynamic Time Warping (DTW) [13].

In this paper, we use a 9-layer CNN as the classification algorithm. There are several advantages of the CNN over the other classifying approaches. (1) Instead of manually selecting features, CNN is able to automatically learn parameters and features. (2) CNN is very suitable for complex problems. Based on our study, we find that it is capable of handling mobility noises and reducing overfitting. (3) CNN is very fast to run in the inference stage even when the number of classes is very large.

A. Data Scaling

As 17 predefined hand gestures are different from each other and different users perform hand gestures at different speeds, the duration of each hand gesture is different. Fig. 9 shows the distribution of the gesture duration in our dataset. The maximum gesture duration is 2.7 seconds. The minimum gesture duration is 260 ms. The average gesture duration is 1.2 seconds. As the sampling rate is 50 Hz, each gesture contains 60 sample points on average.

As a CNN model requires input data with the same size, we format the segment data so that each segment has the same size. We apply the Cubic Spline Interpolation [24] to rescale the number of sample points for each segment to 60. As 3-axis accelerometer readings and 3-axis gyroscope readings are collected by each sampling, 60×6 data points

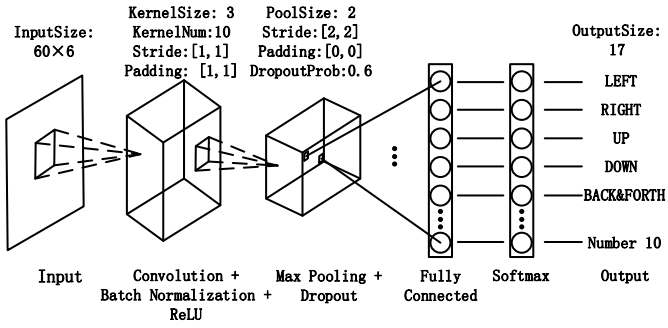


Fig. 10. Architecture and parameter settings of the 9-layer CNN

are generated after interpolation. This 60×6 data matrix is used for classification.

B. Convolutional Neural Network

We design a 9-layer CNN as the classification algorithm. A CNN consists of an input and an output layer, as well as multiple hidden layers. The output of the i-th layer of a n-layer neural network is given by:

$$\boldsymbol{y}^{(i)} = \sigma^{(i)}\left(\boldsymbol{W}^{(i)}\boldsymbol{x}^{(i)} + \boldsymbol{b}^{(i)}\right), \tag{6}$$

where $\boldsymbol{y}^{(i)}$ is the output, $\boldsymbol{x}^{(i)}$ is the input, $\sigma^{(i)}$ is the activation function, $\boldsymbol{W}^{(i)}$ is the weight matrix, and $\boldsymbol{b}^{(i)}$ is the bias vector [25]. $\boldsymbol{x}^{(0)}$ is the original input, which is a matrix of the accelerometer and gyroscope sensor data. $\boldsymbol{y}^{(n)}$ is the final output, which is one of 17 predefined hand gestures. The output of the $(i-1)$-th layer is the input of the i-th layer, i.e., $\boldsymbol{x}^{(i)} = \boldsymbol{y}^{(i-1)}$.

Fig. 10 shows the architecture and parameter settings of the CNN. It includes the following 9 layers:

1) Input Layer: The input layer is the entrance to the CNN. It provides data for the following layers. After data scaling, we get a 60×6 data matrix. This matrix is supplied to the input layer.

2) Convolutional Layer: The convolutional layer divides the input data into multiple regions. For each region, it computes a dot product of the weights and the input, and then adds a bias term. A set of weights that are applied to a region is called a kernel. The kernel moves along the input data vertically and horizontally, repeating the same computation for each region. The step size with which it moves is called a

stride. We use ten 3×3 kernels and stride of 1 in both vertical and horizontal directions. To preserve the output size of the convolutional layer, we use a padding of 1 in both vertical and horizontal directions. It adds rows or columns of zeros to the borders of the original input.

3) Batch Normalization Layer: Batch normalization is used to speed up network training, reduce the sensitivity to network initialization, and improve the generalization of the neural network when the training dataset contains data from different users. To take full advantage of batch normalization, we shuffle the training data after each training epoch.

4) ReLU Layer: Convolutional and batch normalization layers are usually followed by a nonlinear activation function. We choose a Rectified Linear Unit (ReLU) as the activation function. It performs a threshold operation on each input, where any input value less than zero is set to zero. ReLU is easy to compute and optimize. It provides fast and effective training for deep neural networks. It has been shown more effective than traditional activations, such as logistic sigmoid and hyperbolic tangent, and is widely used in CNN [25].

5) Max-pooling Layer: The max-pooling layer reduces the number of connections to the following layers by down-sampling. It partitions the input into a set of non-overlapping rectangles. For each rectangle, it outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer reduces the number of parameters to be learned in the following layers, and hence reduces overfitting.

6) Dropout Layer: As a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is dropout. It randomly removes some nodes from a neural network with a given probability. All the incoming and outgoing edges to a dropped-out node are also removed. The dropout probability in our system is 0.6.

7) Fully-connected Layer: The fully-connected layer connects all of its neurons to the neurons in the previous layer, i.e., the dropout layer. It combines all the features learned by the previous layers to classify the input. The size of the output of the fully-connected layer is equal to the number of hand gesture classes, i.e., 17 in our experiments.

8) Softmax Layer: The softmax layer applies a softmax activation function to the input. The softmax activation function normalizes the output of the fully connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer.

9) Classification Layer: The probabilities returned by the softmax activation function are the input to the classification layer . The classification layer assigns this input to one of the 17 hand gestures, and computes the loss function.

As in many other learning systems, the parameters of a CNN model are optimized to minimize the loss function. We apply the Stochastic Gradient Descent with Momentum [26] to learn the CNN parameters (weights $\boldsymbol{W}$ and biases $\boldsymbol{b}$). It updates the parameters of the CNN by taking small steps in the direction of the negative gradient of the loss function:

$$\theta_{l+1} = \theta_l - \alpha \nabla E(\theta_l) + \gamma(\theta_{l+1} - \theta_l), \tag{7}$$

where θ is the parameter vector, l is the iteration index, α is the learning rate, $E(\theta)$ is the loss function, and γ is the momentum term [25]. The momentum term γ controls the contribution of the previous gradient step to the current iteration. We use a momentum term of 0.9 and a learning rate of 0.03.

Very large weights can cause the weight matrix $\boldsymbol{W}$ to get stuck in a local minimum easily since gradient descent only makes small changes to the direction of optimization. This eventually makes it hard to explore the weight space, which leads to overfitting. To reduce overfitting, we use L2 regularization, which adds an extra term into the cost function to penalize large weights. The regularized loss function is:

$$E_R(\theta) = E(\theta) + \lambda \Omega(\boldsymbol{W}), \tag{8}$$

where λ is the regularization factor, and $\Omega(\boldsymbol{W}) = \boldsymbol{W}^T\boldsymbol{W}/2$ is the regularization function. The regularization factor in our system is 0.03.

C. Performance

We apply both the 5-fold cross-validation and leave-one-subject-out cross-validation to evaluate the performance of our CNN model. Accuracy is considered as the evaluation metric. It is defined as the number of correctly classified instances divided by the number of all testing instances. Under the 5-fold cross-validation test, the accuracy of the gesture classification when the user is standing, walking, and jogging are 92.2%, 90.1%, and 88.6%, respectively. The gesture classification accuracy when the user is jogging is only 3.6% lower than that when the user is standing. Therefore, we conclude that the moving scenarios do not influence the classification accuracy significantly in our system under the 5-fold cross-validation test.

Under the leave-one-subject-out cross-validation test, the accuracy of the gesture classification when the user is standing, walking, and jogging are 86.6%, 84.6%, and 75.1%, respectively. The gesture classification accuracy when the user is jogging is 11.5% lower than that when the user is standing. In contrast to the 5-fold cross validation test, the gesture classification is heavily influenced by the moving scenarios under the leave-one-subject-out cross-validation test. This is reasonable as the leave-one-subject-out test brings noises from different body sizes and different ways of performing the same type of hand gesture. The combination of these noises and mobility noises significantly influences the gesture classification performance.

We compare our gesture classification algorithm with E-gesture [11]. E-gesture proposes a Multi-situation HMM model for gesture classification. During training, a HMM model is built and trained for each pair of gesture and mobility situation. During testing, E-gesture computes the Viterbi scores [12] for each of the HMM models, and the best candidate is selected to be the classification result. We call this method Multi-situation HMM.

TABLE IV
COMPARISON OF THE GESTURE CLASSIFICATION PERFORMANCE

Test	Scenarios	Algorithms	Accuracy
5-fold	Standing	Multi-situation HMM	97.8%
		CNN	92.2%
	Walking	Multi-situation HMM	96.2%
		CNN	90.1%
	Jogging	Multi-situation HMM	96.5%
		CNN	88.6%
LOSO	Standing	Multi-situation HMM	70.5%
		CNN	86.6%
	Walking	Multi-situation HMM	69.3%
		CNN	84.6%
	Jogging	Multi-situation HMM	60.7%
		CNN	75.1%

TABLE V
COMPARISON OF THE OVERALL PERFORMANCE

Test	Scenarios	Algorithms	Accuracy
5-fold	Standing	E-Gesture	95.8%
		MobiGesture	85.0%
	Walking	E-Gesture	8.2%
		MobiGesture	83.1%
	Jogging	E-Gesture	8.2%
		MobiGesture	82.8%
LOSO	Standing	E-Gesture	69.1%
		MobiGesture	80.2%
	Walking	E-Gesture	5.9%
		MobiGesture	79.5%
	Jogging	E-Gesture	5.2%
		MobiGesture	70.6%

We apply the 5-fold cross-validation test and leave-one-subject-out cross-validation test to evaluate the gesture classification performance. Table IV shows the gesture classification accuracy of these two algorithms under three different moving scenarios. Under the leave-one-subject-out cross-validation test, CNN is 16.1%, 15.3%, and 14.4% more accurate than Multi-situation HMM when the user is standing, walking and jogging, respectively. Under the 5-fold cross-validation test, Multi-situation HMM is roughly 7% more accurate than CNN. For Multi-situation HMM, the average accuracy is 96.8% under the 5-fold cross-validation test, and 66.8% under the leave-one-subject-out cross-validation test. There is 30% accuracy difference between these two tests. It shows that Multi-situation HMM model is overfitted. For CNN, the accuracy difference between these two tests is 8.2%. The reasonably small difference shows that overfitting is significantly reduced.

VI. PERFORMANCE EVALUATION

In this section, we first evaluate the overall performance of MobiGesture, which integrates the aforementioned segmentation and CNN algorithms. We also compare MobiGesture with state-of-the-art work. Then, we evaluate the overhead of MobiGesture and compare it with state-of-the-art work.

A. Accuracy

The overall performance of MobiGesture and E-gesture are shown in Table V. Under the 5-fold cross-validation test, E-gesture performs well when the user is standing. However, when the user is walking or jogging, the accuracy of E-gesture is very low. The reason is that E-gesture can not differentiate the hand gestures from the hand swinging motions. MobiGesture performs stably under different moving scenarios. The recognition accuracy when the user is jogging is only 2.2% lower than that when the user is standing. Under the leave-one-subject-out cross-validation test, when the user is standing, the accuracy of E-gesture is only 69.1%. It is much lower than the accuracy under 5-fold cross validation: 95.8%. It shows that the gesture classification model in E-gesture is overfitted. When the user is walking or running, E-gesture performs poorly again due to the low segmentation accuracy. The accuracy of MobiGesture under the leave-one-subject-out cross-validation test is 3.6% $\sim$ 12.2% lower than that under the 5-fold cross-validation test. The reasonably small difference shows the effectiveness of MobiGesture's anti-overfitting design.

TABLE VI
COMPARISON OF THE TIME CONSUMPTION

Algorithm	Training Time (s)	Testing Time (ms)
Multi-situation HMM	89.6	40.8
CNN	56.7	13.8

B. Time Delay

Table VI shows the time consumption of training and testing of Multi-situation HMM and CNN. For the training time, Multi-situation consumes 58% more time than CNN. For the testing time, Multi-situation HMM consumes roughly three times as much as CNN. Multi-situation HMM trains a HMM model for each pair of the hand gestures and the moving scenarios, while MobiGesture only trains one CNN model for all the hand gestures and moving scenarios. As the number of moving scenarios increases, Multi-situation HMM consumes more time for testing, while our CNN keeps the same. Therefore, CNN is more practical than Multi-situation HMM to be implemented for real-time classification.

VII. RELATED WORK

As far as we know, two efforts have been put forth to study the gesture recognition problem when the user is moving. Park et al. [11] propose a gesture recognition system with a hand-worn sensor and a mobile device. To segment hand gestures, they design a threshold-based closed-loop collaborative segmentation algorithm. It automatically adjusts the threshold according to four mobility situations: RIDE, STAND, WALK, and RUN. To recognize hand gestures, they propose a Multi-situation HMM architecture. There are several limitations in their system. For the gesture segmentation, their threshold-based segmentation algorithm cannot effectively differentiate the predefined hand gestures from the hand swinging motions in our dataset. For the gesture recognition, they train a HMM model for each pair of hand gesture and mobility situation. In total, 32 HMM models are trained in their system. As the number of the hand gestures or the number of the mobility situations increases, their computational cost increases dramatically. Different from this work, we only train one CNN model, which consumes much less computational power and time.

Additionally, evaluation results show that our CNN model performs better than Multi-situation HMM model under leave-one-subject-out cross-validation test. The second work comes from Murao et al. [13]. They propose a combined-activity recognition system. This system first classifies user activity into one of three categories: postures, behaviors, and gestures. Then DTW is applied to recognize hand gestures for the specific category. However, their system requires five sensors attached to the human body to recognize activity. Instead, we only use one sensor.

Inertial sensors-based gesture recognition has been widely studied in mobile and pervasive computing. Various approaches dealing with the recognition of gestures or events have been presented. RisQ [15] applies motion sensors on the wristband to recognize smoking gestures. Bite Counter [27] utilizes a watch-like device with a gyroscope to detect and record when an individual takes a bite of food. Porzi et al. [23] propose a smart watch-based gesture recognition system for assisting people with visual impairments. Xu et al. classify hand/finger gestures and written characters from smart watch motion sensor data [22]. FingerPad [28], uTrack [29], and Finexus [30] use magnetic sensors to recognize finger gestures. However, none of these efforts takes body movement, such as walking and jogging, into consideration.

VIII. Conclusion

In this paper, we present MobiGesture, a mobility-aware gesture recognition system for healthcare. We present a novel mobility-aware gesture segmentation algorithm to detect and segment hand gestures. In addition, we design a CNN model to classify the hand gestures with mobility noises. Evaluation results show that the proposed CNN is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when the user is standing, walking and jogging, respectively. The proposed CNN is also two times faster than state-of-the-art work.

Acknowledgments

Special thanks to our participants in our user studies, and all the anonymous reviewers. This work was supported by NSF CNS-1253506 (CAREER).

References

[1] M. C. Ashe, W. C. Miller, J. J. Eng, and L. Noreau, "Older adults, chronic disease and leisure-time physical activity," *Gerontology*, vol. 55, no. 1, pp. 64–72, 2009.

[2] I.-M. Lee and D. M. Buchner, "The importance of walking to public health." *Medicine and science in sports and exercise*, vol. 40, no. 7 Suppl, pp. S512–8, 2008.

[3] T. Prohaska, E. Belansky, B. Belza, D. Buchner, V. Marshall, K. McTigue, W. Satariano, and S. Wilcox, "Physical activity, public health, and aging: critical issues and research priorities," *The Journals of Gerontology Series B: Psychological Sciences and Social Sciences*, vol. 61, no. 5, pp. S267–S273, 2006.

[4] E. M. Simonsick, J. M. Guralnik, S. Volpato, J. Balfour, and L. P. Fried, "Just get out the door! importance of walking outside the home for maintaining mobility: findings from the women's health and aging study," *Journal of the American Geriatrics Society*, vol. 53, no. 2, pp. 198–203, 2005.

[5] S. E. Hardy, Y. Kang, S. A. Studenski, and H. B. Degenholtz, "Ability to walk 1/4 mile predicts subsequent disability, mortality, and health care costs," *Journal of general internal medicine*, vol. 26, no. 2, pp. 130–135, 2011.

[6] Global recommendations on physical activity for health. world health organization. [Online]. Available: http://www.who.int/dietphysicalactivity/factsheet_recommendations/en

[7] Physical activity guidelines for americans. u.s. department of health and human services. [Online]. Available: http://health.gov/paguidelines

[8] Nike+ run club app. [Online]. Available: https://www.nike.com/us/en_-us/c/nike-plus/running-app-gps

[9] Runkeeper app. [Online]. Available: https://runkeeper.com/

[10] Mapmyrun app. [Online]. Available: http://www.mapmyrun.com/

[11] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, and J. Song, "E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices," in *Proceedings of the ACM SenSys*. ACM, 2011, pp. 260–273.

[12] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," in *The Foundations Of The Digital Wireless World: Selected Works of AJ Viterbi*. World Scientific, 2010, pp. 41–50.

[13] K. Murao and T. Terada, "A recognition method for combined activities with accelerometers," in *Proceedings of the ACM UbiComp*. ACM, 2014, pp. 787–796.

[14] H. Junker, O. Amft, P. Lukowicz, and G. Tröster, "Gesture spotting with body-worn inertial sensors to detect user activities," *Pattern Recognition*, vol. 41, no. 6, pp. 2010–2024, 2008.

[15] A. Parate, M.-C. Chiu, C. Chadowitz, D. Ganesan, and E. Kalogerakis, "Risq: Recognizing smoking gestures with inertial sensors on a wristband," in *Proceedings of the ACM MobiSys*. ACM, 2014, pp. 149–161.

[16] H. Zhao, S. Wang, G. Zhou, and D. Zhang, "Ultigesture: A wristband-based platform for continuous gesture control in healthcare," *Smart Health*, 2018.

[17] P. Alfeld, "A trivariate cloughtocher scheme for tetrahedral data," *Computer Aided Geometric Design*, vol. 1, no. 2, pp. 169–181, 1984.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *Proceedings of the ACM SIGKDD*, vol. 11, no. 1, pp. 10–18, 2009.

[19] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[20] W.-C. Bang, W. Chang, K.-H. Kang, E.-S. Choi, A. Potanin, and D.-Y. Kim, "Self-contained spatial input device for wearable computers," in *Proceedings of the IEEE ISWC*. IEEE Computer Society, 2003, p. 26.

[21] A. Y. Benbasat and J. A. Paradiso, "An inertial measurement framework for gesture recognition and applications," in *International Gesture Workshop*. Springer, 2001, pp. 9–20.

[22] C. Xu, P. H. Pathak, and P. Mohapatra, "Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch," in *Proceedings of the ACM HotMobile*. ACM, 2015, pp. 9–14.

[23] L. Porzi, S. Messelodi, C. M. Modena, and E. Ricci, "A smart watch-based gesture recognition system for assisting people with visual impairments," in *Proceedings of the ACM IMMPD*. ACM, 2013, pp. 19–24.

[24] C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor, *A practical guide to splines*. Springer, 1978, vol. 27.

[25] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[26] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT*. Springer, 2010, pp. 177–186.

[27] Y. Dong, A. Hoover, J. Scisco, and E. Muth, "A new method for measuring meal intake in humans via automated wrist motion tracking," *Applied psychophysiology and biofeedback*, vol. 37, no. 3, pp. 205–215, 2012.

[28] L. Chan, R.-H. Liang, M.-C. Tsai, K.-Y. Cheng, C.-H. Su, M. Y. Chen, W.-H. Cheng, and B.-Y. Chen, "Fingerpad: private and subtle interaction using fingertips," in *Proceedings of the ACM UIST*. ACM, 2013, pp. 255–260.

[29] K.-Y. Chen, K. Lyons, S. White, and S. Patel, "utrack: 3d input using two magnetic sensors," in *Proceedings of the ACM UIST*. ACM, 2013, pp. 237–244.

[30] K.-Y. Chen, S. N. Patel, and S. Keller, "Finexus: Tracking precise motions of multiple fingertips using magnetic sensing," in *Proceedings of the ACM CHI*. ACM, 2016, pp. 1504–1514.