2020 16th International Conference on Mobility, Sensing and Networking (MSN) | 978-1-7281-9916-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/MSN50589.2020.00054

Magic Wand : Towards Plug-and-Play Gesture Recognition on Smartwatch

Zhipeng Song
School of Software
Tsinghua University
songzp18@mails.tsinghua.edu.cn

Zhichao Cao
Dept. of Computer Sci. & Eng.
Michigan State University
caozc@msu.edu

Zhenjiang Li
Dept. of Computer Sci.
City University of Hong Kong
zhenjiang.li@cityu.edu.hk

Jiliang Wang
School of Software
Tsinghua University
jiliangwang@tsinghua.edu.cn

***Abstract*—We propose *Magic Wand* which automatically recognizes 2D gestures (e.g., symbol, circle, polygon, letter) performed by users wearing a smartwatch in real-time manner. Meanwhile, users can freely choose their convenient way to perform those gestures in 3D space. In comparison with existing motion sensor based methods, Magic Wand develops a white-box model which adaptively copes with diverse hardware noises and user habits with almost zero overhead. The key principle behind Magic Wand is to utilize 2D stroke sequence for gesture recognition. Magic Wand defines 8 strokes in a unified 2D plane to represent various gestures. While a user is freely performing gestures in 3D space, Magic Wand collects motion data from accelerometer and gyroscope. Meanwhile, Magic Wand removes various acceleration noises and reduces the dimension of 3D acceleration sequences of user gestures. Moreover, Magic Wand develops stroke sequence extraction and matching methods to timely and accurately recognize gestures. We implement Magic Wand and evaluate its performance with 4 smartwatches and 6 users. The evaluation results show that the median recognition accuracy is 94.0% for a set of 20 gestures. For each gesture, the processing overhead is tens of milliseconds.**

I. Introduction

Smartwatch becomes more and more popular in our daily life, and they can enable quick and featured interaction with diverse mobile and Internet of Things (IoT) applications. Particularly, wrist gesture recognition [8] [3] [5] [13] is a meaningful interaction interface. When a user wearing a smartwatch in his/her wrist freely performs familiar 2D gestures (i.e., symbol, circle, polygon, letter) in the air, various functions and applications (e.g., motion sensing games, VR games, mobile application shortcuts, IoT device control) can be enabled, if those gestures can be accurately recognized.

As we know, motion sensors (e.g., accelerometer and gyroscope) are available on most smartwatches (e.g., Apple Watch, LG Watch Sport, Moto 360, Samsung Gear S3, Huawei Watch, Huawei Watch 2 Pro). Many works are proposed to enable wrist gesture recognition by using those motion sensors. However, they are still far from a plug-and-play manner due to three limitations as follows.

1) *Unscalable gestures.* Given a set of pre-defined wrist gestures, some works [3] [4] [11] [19] [20] generate gesture templates or train a gesture classifier for gesture recognition. Faced with diverse user habits (e.g., speed, size, rotation of wrist and elbow, etc.), these works need much configuration overhead to achieve a general recognition model, which limits the scalability of gesture selection. Specifically, Xu et. al. [20] assume that users perform letter gestures on a vertical surface with his/her index finger. RisQ [11] only targets to recognize smoking gesture.

2) *Non-real-time recognition.* With accelerometer and gyroscope, wrist trajectory can be recovered by using coordinate rotation transformation and acceleration double integral in 3D space. Hopefully, the trajectory can be used for ubiquitous gesture recognition. Some methods [13] [14] are proposed for wrist location tracking. With MUSE [14], the median error of wrist location is 8.8cm when users perform simple gestures like eating, smoking, walking, lecturing, etc. However, to control the error of wrist motion tracking, the computational complexity is usually unaffordable to support real-time data processing. For example, MUSE [14] needs an edge-computer or the cloud to achieve second-level end-to-end latencies. For supporting more mobile applications, however, it is desirable to guarantee millisecond-level end-to-end latency.

Overall, these limitations significantly hinder plug-and-play gesture recognition on smartwatch. In this paper, we address those limitations and propose *Magic Wand*, a plug-and-play wrist gesture recognition system on smartwatch. Regarding to diverse user habits, with Magic Wand, a user can perform gestures as usual by performing only one starting gesture which is just like raising user's hand before using Kinect. The key idea is that the DoFs (degrees of freedom) of wrist motion is approximately stable and planar while a user is performing different gestures in a specific scenario. With the starting gesture, we initialize a temporal gesture transformation function to represent the DoFs of wrist motion. It converts a 3D trajectory to an equivalent 2D trajectory on a unified plane without losing much information.

Moreover, we are inspired by widely used speech recognition, which also needs to address diverse speaking variances (e.g., speed, volume, accent). A word can be represented by a phoneme sequence, which further serves as a natural feature in speech recognition. We observe that 2D stroke is the basic element for performing different gestures. No matter what user habits are, the 2D stroke sequence of a gesture is constant so that can be exploited as a natural feature for gesture recognition.

To achieve stroke sequence based wrist gesture recognition, we define 8 strokes in a unified 2D plane (called *stroke plane*)

978-1-7281-9916-0/20/$31.00 ©2020 IEEE
DOI 10.1109/MSN50589.2020.00054

to represent various gestures (Section II). Magic Wand uses accelerometer and gyroscope to sense users' wrist motion. Since the accelerometer is quite noisy while performing gestures, we design a light-weight noise filtering model, which can be efficiently initialized by the starting gesture. We extract a stroke sequence from the measured acceleration sequences of wrist motion. Then, we can timely and accurately recognize the corresponding gesture by matching the extracted stroke sequence with those pre-defined stroke sequence templates. Overall, with smartwatch, Magic Wand achieves plug-and-play gesture recognition and supports real-time interaction for various applications.

We implement Magic Wand in Android and conduct extensive experiments to evaluate the performance of Magic Wand with six users and four smartwatches. The evaluation results show that the median recognition accuracy is 94.0% for a set of 20 gestures. Our contributions are summarized as follows.

- We propose an innovative stroke sequence based wrist gesture recognition system. There is no constraint of user habits. Various styles of gestures can be naturally supported.
- We present the system design of Magic Wand, which addresses several practical challenges to achieve accurate and real-time wrist gesture recognition.
- We implement Magic Wand in Android and evaluate its performance with 6 users and 4 smartwatches. The results show a high recognition accuracy for a set of 20 gestures.

The rest of paper is organized as follows. Section II presents system overview. Section III presents the detailed system design. Section IV shows the implementation of Magic Wand. Section V shows the evaluation results. Section VI introduces the related work. Section VII concludes this work.

II. System Overview

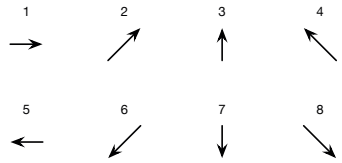
A. System Goals

Just like using Kinect to play motion sensing game, Magic Wand aims to achieve plug-and-play wrist gesture recognition on smartwatch with the following goals.

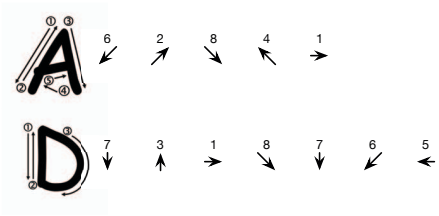
- Magic Wand should require no user-specific training (e.g., performing each gesture several times at the beginning).
- Magic Wand should require no scenario-limited wrist motion (e.g., fixing the wrist, elbow and shoulder postures in the air).
- Magic Wand should be able to recognize kinds of gestures (e.g., geometric shape, symbol, number, letter) for various applications and achieve a high recognition accuracy in real-time manner.

B. System Design Principle

As shown in Figure 1(a), we define 8 unique strokes pointing to 8 different directions in stroke plane. Those strokes are indexed from 1 to 8 (called *stroke index*) in counterclockwise order. The directions of two adjacent strokes have 45° difference. We can use stroke sequences to represent various gestures. Figure 1(b) shows how to use those 8 strokes to



(a) 8 unique strokes in stroke plane



(b) Stroke sequences of two letter gestures

Fig. 1: (a) the illustration of the stroke definition. (b) two letter gestures **A** and **D** represented by a stroke sequence, separately.

represent two letter gestures **A** and **D**. ①-⑥ of **A** and ①-③ of **D** show the predetermined gesture trajectories. Then, **A** and **D** can be represented by a stroke sequence "62841" and "7318765", respectively. For line trajectory (e.g., ① of **A**), we can use a stroke "6", which has the similar direction, to represent it. For curve trajectory (e.g., ③ of **D**), we can use a stroke sequence "18765", which consists of all possible directions, to approximate the curve. Given the predetermined trajectory of any gesture, we can easily generate its stroke sequence by following the same rules.

C. System Architecture

To achieve stroke sequence based gesture recognition in practice, Figure 2 shows the architecture of Magic Wand, which consists of four components as follows.

1) *Acceleration Noise Filtering* (Section III-A). We use motion sensors (e.g., accelerometer and gyroscope) to record the motion of a wrist gesture. The acceleration noises, however, incur significant error of trajectory recovery [13] so that hinder an easy way for stroke extraction. According to our observation, one part of acceleration noises comes from the accelerometer itself and can be calibrated by complicated process [15] [21]. The other part is the gravity which is hard to completely removed. This component filters out those acceleration noises from raw acceleration sequence in a light-weight way. Finally, we fetch the gesture acceleration sequence in a unified coordinate system X-Y-Z.

2) *Gesture Transformation* (Section III-B). Since a user usually performs gestures with user-specific DoFs of wrist motion in 3D space. To convert the user diversity to a unified 2D stroke plane, this component reduces the dimensions of the gesture acceleration sequence from coordinate system X-Y-Z to 2D coordinate system X_w-Y_w (i.e., writing plane). Just like raising user's hand before using Kinect, a user performs a predetermined starting gesture before using Magic Wand.

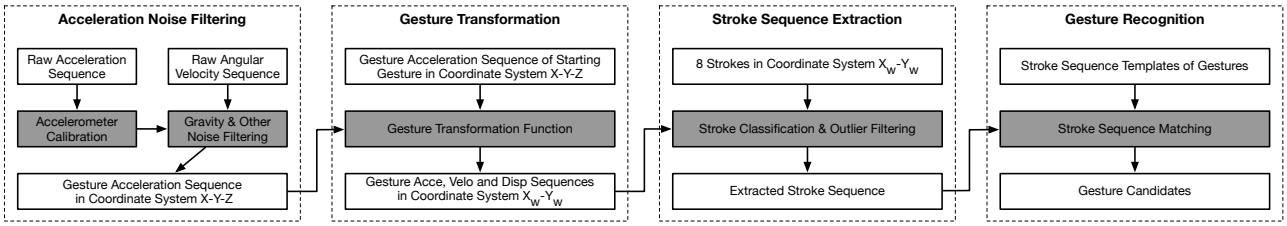


Fig. 2: The illustration of system architecture. The grey rectangles show the key functions.

We use the starting gesture to determine the writing plane and initialize the corresponding gesture transformation function. Given another 3D wrist motion performed by the same user in the same scenario, we can obtain its gesture acceleration sequence in X_w-Y_w with the gesture transformation function. We further calculate corresponding velocity and displacement sequences to represent the directions and trajectory of wrist motion in X_w-Y_w.

3) *Stroke Sequence Classification* (Section III-C). This component extracts stroke sequence given the wrist motion on writing plane. We compare the motion direction and the defined stroke direction to classify strokes. Due to the potential distortion of gesture transformation, the bias of gesture performance and residual acceleration noise, the error strokes inevitably appear. We develop a heuristic method to filter out those error strokes.

4) *Gesture Recognition* (Section III-D). This component calculates the similarity between the extracted stroke sequence and the stroke sequence templates of all possible gestures, then the gesture is recognized as the one with the highest similarity. Specifically, we utilize DTW to efficiently calculate the similarity between two stroke sequences and adaptively set the cost between different pairs of strokes.

III. System Design

In this section, we introduce the detailed design of the four components in Magic Wand.

A. Acceleration Noise Filtering

A part of acceleration noise is from the gravity. Besides the gravity, the acceleration noise contains another part (called *inherent accelerometer noise*) due to the lack of MEMS IMU calibration [15] [21].

1) Accelerometer Calibration: To model the inherent accelerator noise, we assume the noise comes from two factors, named scale and bias along each axis respectively. Specifically, the acceleration measured by accelerator $a_m(t) = (a_x^m(t), a_y^m(t), a_z^m(t))^T$ can be represented by ground truth acceleration $a(t) = (a_x(t), a_y(t), a_z(t))^T$ as follow:

$$a_m(t) = Ka(t) + b \tag{1}$$

where $K = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{pmatrix}$ denotes 3-axis scale factors and $b = (b_x, b_y, b_z)^T$ denotes 3-axis bias factors. We assume both K and b are constant for an accelerometer.

To initialize the parameters in a light-weight way, we observe that the accelerometer of a smartwatch records the gravity when it is static. We use $g = [g_x, g_y, g_z]^T$ to indicate the gravity in initial smartwatch coordinates. When the smartwatch moves in 3D space, we can use the angular velocity measured by gyroscope to compute the rotation matrix $R_{3\times3}$ [17] [11] which can be used to convert the current smartwatch coordinates to the initial smartwatch coordinates. Denote $g(t_i)$ as the gravity measured at t_i in current smartwatch coordinates, we have:

$$g = R(t_i)g(t_i) \tag{2}$$

Note that the rotation matrix $R(t_i)$ is an orthogonal matrix, thus $R(t_i)^{-1} = R(t_i)^T$. Combine Equation 1 and Equation 2, when the smartwatch is static at time t_i, the output acceleration comes as:

$$a_m(t_i) = Kg(t_i) + b = K(R^{-1}(t_i)g) + b \tag{3}$$

and it further gives:

$$K^{-1}a_m(t_i) - R^T(t_i)g - K^{-1}b = 0 \tag{4}$$

that is:

$$[A_m(t_i), -R^T(t_i), -I] \begin{bmatrix} k \\ g \\ b' \end{bmatrix} = 0 \tag{5}$$

where

$$A_m = \begin{pmatrix} a_x^m & 0 & 0 \\ 0 & a_y^m & 0 \\ 0 & 0 & a_z^m \end{pmatrix}, k = \begin{bmatrix} k_x^{-1} \\ k_y^{-1} \\ k_z^{-1} \end{bmatrix}, b' = K^{-1}b \tag{6}$$

Moreover, given N static acceleration measurements $\{a_m(t_1), a_m(t_2), \cdots, a_m(t_N)\}$, we define a stack matrix X as below:

$$X = \begin{bmatrix} A_m(t_1), -R^T(t_1), -I \\ A_m(t_2), -R^T(t_2), -I \\ \vdots \\ A_m(t_N), -R^T(t_N), -I \end{bmatrix} \tag{7}$$

Then according to Equation 5, we can obtain vector $[k, g, b']^T$ by solving a minimization problem as:

$$x_0 = arg \min_{||x_0||=1} ||Xx_0|| \tag{8}$$

To ensure high computation efficiency, we adopt closed-form solution by calculating the eigenvectors of $X'X$, and x_0 is the eigenvector that corresponds to the minimal eigenvalue of

$X'X$. Once we get x_0, since g represents gravity, we can scale the vector x_0 to $[k, g, b']^T$ according to gravity magnitude and direction, and then we get K and b and use them to filter inherent accelerator noise.

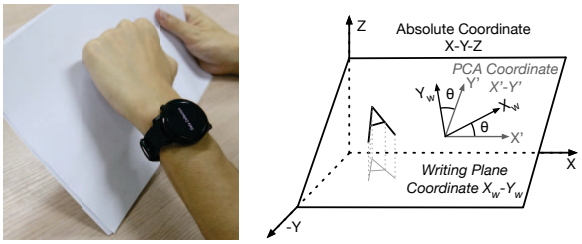
In practice, before using Magic Wand, a user needs to perform a calibration gesture for initializing the model of accelerometer calibration. According to the variance of acceleration data in a short period, we can use a threshold based method to extract the static acceleration measurements and generate the matrix X. In this way, we can calibrate the accelerator in a light-weight way and real-time manner.

2) Gravity Filtering: After accelerometer calibration, given the raw acceleration sequence of a wrist motion, we further filter out the gravity. With the same rotation matrix R (Section III-A1) measured by continuous angular velocity readings, we transform all acceleration readings measured at different time to the equivalent ones in X-Y-Z coordinate system (called *absolute coordinate system*) [17] which is the smartwatch coordinate system at the starting point. For a wrist motion, $\{a(0), a(1), ..., a(n)\}$ indicates the measured 3-axes acceleration sequence in X-Y-Z. $a(i)$ $(i \in [0, n])$ contains gravity $\{g_x, g_y, g_z\}$, which is constant along 3-axes in X-Y-Z, and the acceleration of wrist motion $a(i)$. The velocity of wrist motion is usually zero when a user starts and finishes performing a gesture. Thus, the gravity is the mean of the sum of $\{a(0), a(1), ..., a(n)\}$. By using the mean removal [7][17], we get rid of gravity and fetch the pure acceleration of wrist motion in X-Y-Z.

B. Gesture Transformation

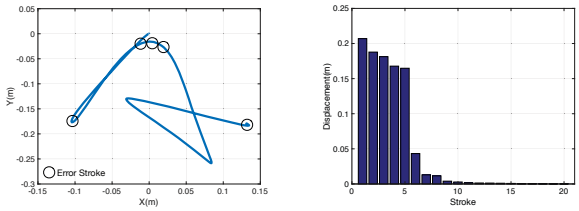
As shown in Figure 3(a), a user sits in front of a desk and performs a letter gesture **A** (black **A**) on a sloping plane. The writing plane should be parallel with the sloping paper which is unknown in X-Y-Z. If we directly select X-Y plane as the writing plane for simplicity, the obtained letter gesture is the grey **A** which is the projection of black **A** on X-Y plane. We can see obvious stroke distortion which may degrade recognition accuracy. We use a starting gesture (e.g., letter gesture **A**) to initialize a gesture transformation function which can transform the acceleration sequence in the X-Y-Z to an ideal writing plane (as X_w-Y_w) while reserving the most gesture information. Moreover, the transformation function should guarantee the stroke directions on writing plane are aligned to that on stroke plane. Therefore, other gestures performed by the user in the same scenario can share the same writing plane and gesture transformation function.

1) Writing Plane Selection: We use *Principle Component Analysis* (PCA) [2] to determine the writing plane. Specifically, given the acceleration sequence $a = \{a_x, a_y, a_z\}$ in X-Y-Z, PCA uses an orthogonal transformation, denoted as $\mathbf{x} = (x_1, x_2, x_3)^T, \mathbf{y} = (y_1, y_2, y_3)^T, \mathbf{z} = (z_1, z_2, z_3)^T$, to transform it to three principle components along X', Y' and Z' axes (called *PCA Coordinate System*). PCA maximizes the acceleration values on X' and Y' axes. X'-Y' plane naturally keeps the information of the original gesture as much as possible. Thus X'-Y' plane can be selected as the writing plane.



(a) Scenario (b) Gesture Transformation

Fig. 3: A scenario used to explain the principle of gesture transformation.



(a) Trajectory of **A** (b) Stroke displacement

Fig. 4: The trajectory of letter gesture **A** on their writing plane and the corresponding displacement of extracted strokes.

Z' axis points to the normal direction of the selected writing plane. Finally, we can calculate the acceleration sequence a' in X'-Y'-Z' with a projection matrix $(\mathbf{x}, \mathbf{y}, \mathbf{z})$.

However, as shown in Figure 3(b), we usually cannot directly use the X' and Y' axes of PCA coordinate system as the X_w and Y_w axes of writing plane due to the unaligned stroke directions. A rotation of angle θ may exists between the X'/Y' axis and X_w/Y_w axis. In X'-Y' coordinate system, the direction of all strokes is rotated θ. If θ is large, a stroke may be misclassified. For example, if θ is 45°, strokes {1,2,3,...,7,8} will be classified as {2,3,4,...,8,1}. Therefore, to align the stroke directions on X'-Y', we need derive the rotation angle θ between coordinate system X'-Y' and X_w-Y_w.

2) Rotation Angle Calculation: Given the acceleration sequence $a' = \{a'_x, a'_y\}$ in PCA coordinate system X'-Y' and the stroke sequence template of the starting gesture, we enumerate all possible θ $(0 < \theta \leq 360)$ with a step Δ_θ. For each θ, we rotate X'/Y' for θ to generate new acceleration sequence $a'(\theta)$, which is further taken as the input to extract a stroke sequence (Section III-C). We calculate the similarity between the extracted stroke sequence and the stroke sequence template of the starting gesture (Section III-D). Finally, we choose the θ^* with the highest similarity as the rotation angle.

After a user performs the starting gesture, we derive the projection matrix and the rotation angle θ^* as his/her gesture transformation function in the scenario.

C. Stroke Sequence Extraction

The velocity indicates the instantaneous direction of wrist motion, namely the stroke direction. We convert the acceleration sequence to the corresponding velocity sequence $\{v(0), v(1), ..., v(n)\}$ on writing plane. Given an instantaneous velocity $v(t)$ at time t, we find the stroke $s(t)$ whose direction is the most similar with $v(t)$ as the instantaneous stroke. With the velocity sequence of wrist motion, we extract a sequence of strokes $\{s(0), s(1), ..., s(n)\}$. If two adjacent strokes $s(t)$ and $s(t+1)$ are the same, they can be merged as one stroke $s(t)$. We filter out the redundant strokes from $\{s(0), s(1), ..., s(n)\}$ and achieve a short stroke sequence $\{s(t_0), s(t_1), ..., s(t_m)\}$ where $t_i \in [0, n]$ ($i \in [0, m]$) is the time of stroke change. However, due to the gesture distortion, some error strokes may exist. Figure 4(a) shows the trajectory of a letter gesture **A**. The black circles show several error strokes that appear during the transition between two strokes. We notice that the displacements of these error strokes are much shorter than the correct ones. Hence, we filter out those strokes whose displacement is too short.

We use $D(i)$ to indicate the displacement of the stroke $s(t_i)$, where $i \in [0, m]$, on writing plane. Then, we sort the stroke displacement sequence $\{D(0), D(1), ..., D(m)\}$ in descending order as $\{D_0, D_1, ..., D_m\}$. Figure 4(b) shows the displacement of all 20 extracted strokes for the letter gesture **A** shown in Figure 4(a). Because setting a fixed threshold cannot adapt to various gesture sizes, we develop a heuristic method to filter out the error strokes. We define two metrics. One metric $\rho(i)$ indicates the degree of the displacement difference between D_i and D_{i+1}. $\rho(i)$ can be calculated as:

$$\rho(i) = \frac{D_i - D_{i+1}}{D_{i+1}} \tag{9}$$

If $\rho(i)$ equals to 1, D_i is twice as long as D_{i+1}. The larger $\rho(i)$ is, the longer D_i is in comparison with D_{i+1}. Hence, if D_i is the boundary between correct strokes and error strokes, $\rho(i)$ should be large. In Figure 4(b), we can see $\rho(5)$ is very large. The other metric $\epsilon(i)$ indicates how much gesture stroke information is reserved if we filter out the strokes whose displacement is shorter than D_i. We calculate $\epsilon(i)$ as:

$$\epsilon(i) = \frac{\sum_{k=0}^{i} D_k}{\sum_{k=0}^{m} D_k} \tag{10}$$

The larger $\epsilon(i)$ is, the more gesture information is reserved. In Figure 4(b), we can see the $\epsilon(5)$ is large enough to reserve the most of gesture information.

We set two thresholds ρ_0 (e.g, 1) and ϵ_0 (e.g., 0.9) for those two metrics, respectively. We find the first $D(i)$ which simultaneously makes $\rho(i) > \rho_0$ and $\epsilon(i) > \epsilon_0$. Then, we filter out those strokes whose displacement is shorter than $D(i)$. In this way, we adaptively set the displacement boundary of error strokes, while reserving the most of gesture stroke information. In Figure 4(b), the five strokes of letter gesture **A** can be extracted with the heuristic method. Although most error strokes can be filtered out, two cases will blind the heuristic method: an error stroke remains when its displacement is as large as the correct ones; a correct stroke is wrongly filtered out when its displacement is too small.

Fig. 5: The system implementation of Magic Wand.

D. Gesture Recognition

To convert an extracted stroke sequence to a gesture, we find the stroke sequence template which has the highest similarity with the extracted stroke sequence. Due to possible error strokes, the length of an extracted stroke sequence may be different with its corresponding stroke sequence template.

We utilize Dynamic Time Warping (DTW) for calculating the similarity between these two stroke sequences. In this algorithm, $C(i, j)$ is denoted to indicate the non-negative cost between stroke i and stroke j. DTW further finds the matching function to minimize the DTW cost using dynamic programming. We use the calculated DTW cost as the similarity between two stroke sequences. The smaller the DTW cost is, the higher the similarity is.

IV. Implementation

We implement Magic Wand in Android. As shown in Figure 5, our prototype system consists of three parts, a smartwatch, a smartphone (e.g., Google Pixel) and a notebook (e.g., Macbook Pro). The smartwatch connects the smartphone via Bluetooth. The smartphone is connected with the notebook by a cable. With Magic Wand, users can perform wrist gestures as usual in various scenarios as shown in Figure 5. The smartwatch records the readings of motion sensors, then sends the data to the notebook for gesture recognition in real time manner. We use four smartwatches (Moto 360, LG Watch Sport, Huawei Watch and Huawei Watch 2 Pro) in our experiments. The sampling rate of motion sensors is approximate 50Hz for Moto 360 and 100Hz for the other three. We utilize interpolation to ensure the stable sampling rate. Facing different gestures and diverse wrist moving speed, the data size of a gesture is about 5K - 20K bytes. For a gesture, the transmission time over bluetooth link is about tens of milliseconds.

In Figure 1(a), the stroke set contains 8 strokes. With more strokes, we can depict more details in stroke sequence templates so that distinguish more gestures. However, it also risks increasing the probability of the error strokes in the extracted stroke sequences so that degrades the recognition accuracy. To balance the tradeoff, we adopt the 8 strokes in our implementation. For stroke sequence extraction, we empirically set ρ_0 and ϵ_0 as 1 and 0.8.

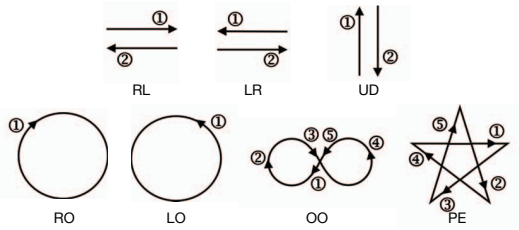


Fig. 6: The 7 geometric gestures and stroke order.

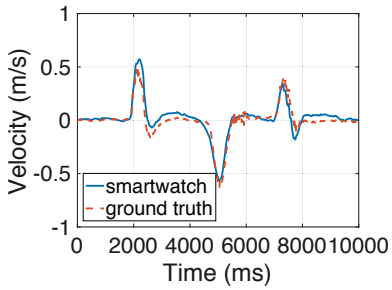


Fig. 7: The velocity comparison along VICON X-axis.

V. Evaluation

We invite six volunteers (3 females and 3 males, 20-30 years old) to evaluate the performance of Magic Wand. We define a set of 20 gestures which include 7 geometric gestures {RL (right-left), LR (left-right), UD (up-down), RO (clockwise-circle), LO (counterclockwise-circle), OO (infinity) and PE (pentagram)} as shown in Figure 6, 3 number gestures {1, 3, 8} and 10 letter gestures {A, C, D, E, H, K, N, Q, R, Z}. We further generate the corresponding stroke sequence templates by following the rules in Section II-B.

To verify the performance of Magic Wand for different users and smartwatches, we gather the data sequences as follows. Each volunteer performs each gesture 10 times following the predetermined stroke order with both Moto 360 and Huawei 2 Pro in his/her comfortable wrist motion scenario. Two male volunteers also perform each gesture 10 times with the other two smartwatches. For different smartwatch, each volunteer performs a calibration gesture to initialize the parameters of our inherent accelerometer noise model at the beginning. For different users, they perform all 20 gestures with their convenient ways and habits. For example, for different users, the shortest average gesture performance time is 1.5 seconds, but the longest one is 2.4 seconds.

A. Acceleration Noise Filtering

We verify the performance of our methods for acceleration noise filtering. A volunteer wears a Moto 360 smartwatch and performs a calibration gesture in the air. When the calibration gesture is performed, we use raw acceleration data sequences and our noise filtering methods to compute the corresponding velocity, as well as make use of a VICON motion capture system [1] as ground truth. We take the VICON coordinates as the reference coordinates and align the smartwatch coordinates to them before performing the calibration gesture.

TABLE I: The recognition accuracy in terms of different gesture scales.

Gesture Scale long (cm), wide (cm)	1/2 A4 (21, 14.9)	1/4 A4 (14.9, 10.5)	1/8 A4 (10.5, 7.5)
Recognition Accuracy (%)	97.0	98.0	98.0

The X-axis velocity of the calibration gesture is shown in Figure 7. We can see that with our acceleration noise filtering, there is little difference from the velocity computed by smartwatch accelerometer to the ground truth. The velocity error does not accumulate with time. The result verifies that our methods of acceleration noise can work efficiently and guarantee the accuracy of later stroke extraction.

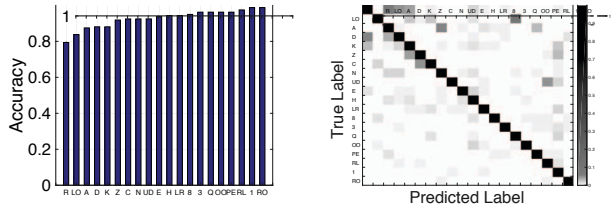
B. Overall Gesture Recognition

1) Accuracy and Error: With total 3200 gathered data sequences of the 20 gestures, we select 16 different gestures (e.g., 'A', 'E', 'K', 'D') as the starting gesture in 16 scenarios with different users/smartwatches. The rest 3184 gestures are used for testing. We show the recognition accuracy of all 20 gestures in ascending order in Figure 8(a). We can see the median recognition accuracy is 94.0%. There are 5 gestures whose recognition accuracies are lower than 90% and the worst is 79.4% for letter gesture 'R'.

We further examine the distribution of recognition error. The results are shown in Figure 8(b). We can see that gesture 'R' can be misclassified as 'A' (6.9%), 'D' (5.0%) and 'K' (4.4%). The reason is that they have similar stroke sequence. thus the possible error of stroke extraction leads the wrong gesture recognition, The same situation holds for 'LO' misclassified as 'Q' (9.4%) and 'A' misclassified as 'N' (5.6%). We also notice that 5.0% of gesture 'UD' is recognized as 'RL'. The reason is that these two gesture are sensitive to the direction of writing plane. The possible error of writing plane rotation leads bias of gesture recognition. Finally, the distribution of recognition error is asymmetric. For example, some 'R' gestures are recognized as 'A' (6.9%), but none of 'A' gestures are recognized as 'R'. This reflects that the extraction accuracy is different for straight stroke and curve stroke. It is easier to perform 'A' by following the pre-determined stroke sequence. Some strokes of the curve stroke may be filtered if the curve is not well performed.

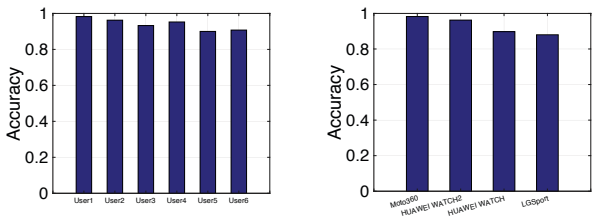
2) Multiple Users and Smartwatches: We further evaluate the influence of different smartwatches and users on the recognition accuracy. Figure 9(a) shows the average recognition accuracy regarding to different users. The recognition accuracies of different users are between 90.0% and 98.3%. We can see the recognition accuracies of different users are high. The difference mainly comes from whether users are familiar with the stroke order, but not depends on user habits. This verifies the stroke sequence based gesture recognition can adapt to various user habits.

Figure 9(b) shows the average recognition accuracy regarding to different smartwatches. The recognition accuracy of



(a) Accuracy (b) Error

Fig. 8: The recognition accuracy (a) and error distribution (b) of all 20 gestures.



(a) User (b) Smartwatch

Fig. 9: The recognition accuracy regarding to different users (a) and smartwatches (b).

different smartwatches are between 88.0% and 98.3%. The performance of Moto 360 is better than the other three smartwatch. The major difference is the motion sensor sampling rate of Moto 360 is 50Hz, which is lower than 100Hz of the other three smartwatches. The high sampling rate may record more velocity glitches which have different motion directions with others in a long stroke trajectory. These glitches can split the long stroke trajectory into short ones during stroke sequence extraction so that increase the probability of error strokes. Hence, Moto 360 with low sampling rate have better recognition accuracy.

C. Recognition Method Comparison

We compare Magic Wand with two state-of-the-art approaches in terms of gesture recognition accuracy and the computation efficiency. One approach (called **Accel DTW**) [4] [6] directly matches the measure acceleration sequences with the reference gesture acceleration sequences. The other approach (called **Naive Bayes**) [12] [18] [20] utilizes the characteristics (e.g., energy, frequency, variation, etc.) of measured acceleration readings as gesture features and Naive Bayes as the gesture classifier. We adopt the 9 characteristics used by [20].

For the set of 20 gestures, we use the data sequences of 1200 gestures performed by six volunteers with Moto 360 smartwatch as the test set. With Magic Wand, we select the same 6 starting gestures with Section V-B for the 6 users. With Accel DTW, we select 360 acceleration sequences as reference sequences for different gestures. With Naive Bayes, we randomly select 600 acceleration sequences of different gestures performed by different users in different scenarios for classifier training. For all three approaches, we filter out the acceleration noises from the acceleration sequences of all gestures with our method.

1) Recognition Accuracy: Table II shows the average recognition accuracy among different approaches. We can see that With Naive Bayes, the recognition accuracy is the lowest, namely 48.2%. This indicates the acceleration characteristics cannot well represent a large set of gestures. With Accel DTW and Magic Wand, the average recognition accuracies of all 20 gestures are 99.2% and 97.4%, respectively. We can see Accel DTW has higher recognition accuracy. The reason is the long acceleration sequence is more representable than our short stroke sequences for the set of 20 gestures.

TABLE II: The performance comparison among different gesture recognition approaches.

	Naive Bayes	Accel DTW	Magic Wand
Accuracy (%)	48.2	99.2	97.4
Time (s)	0.014	5.58	0.014

2) Computation Efficiency: Table II shows the average computation time used to recognize each gesture. Due to long acceleration sequence and the resulted high computation complexity of DTW, the computation overhead of Accel DTW is extremely high. It consumes 5.58s to recognize a gesture, but Magic Wand only consume 0.014s. In comparison with Magic Wand, Accel DTW has two fatal limitations. To adapt to diverse users' habits, it needs all users' overhead to perform 360 gesture references in advance, but Magic Wand only needs 6 starting gestures. Moreover, Accel DTW cannot achieve real-time gesture recognition due to high computation complexity. Magic Wand can achieve real-time gesture recognition while keeping the recognition accuracy high.

VI. Related Work

Wrist gesture recognition acts as a meaningful interaction interface between human and devices. Recently, with the smartphone/smartwatch motion sensors, many works are proposed to recognize various gestures represented by wrist motion. We summarize those methods as follows.

Template matching based approaches. Some works [4] [6] [8] [3] recognize wrist gestures by matching the extracted feature sequence with pre-configured gesture templates. There are two ways to generate gesture templates. On one hand, [4], [6] and [8] directly use the entire acceleration sequence as the gesture template, then DTW severs the matching function. On the other hand, P3 [3] uses 6 strokes (e.g., 2 curve strokes and 4 line strokes) to represent letter gestures, then a decision tree is utilized for stroke sequence matching. Magic Wand proposes a new stroke sequence based template matching. In comparison with P3 [3], 8 unique strokes that point to 8 different directions are more flexible to represent different kinds of gestures and avoid the ambiguity of two different gestures. Moreover, to ensure that the space of gesture template is finite,

Magic Wand can automatically convert free 3D wrist motion to a unified writing plane with low overhead, but existing works usually assume a fixed DoFs of wrist and elbow motion which restricts the scalability of gestures.

Machine learning based approaches. Some works [19] [20] [11] [5] [16] extract several characteristics (e.g., duration, velocity, displacement, angle, motion energy, acceleration variation, etc.) of the acceleration and gyroscope sequence to represent different gestures. Moreover, various learning models (e.g., Decision Tree, Naive Bayes, Logistic Regression, Support Vector Machine, k-Nearest Neighbor, Hidden Markov Model, Random Forest etc.) are used to classify different gestures. For example, Risq [11] trains a decision tree with the gesture features of duration, velocity, displacement and angle to recognize smoking gesture. [20] use various features defined in [10] to represent letter gestures writing on a vertical whiteboard and trains a Naive Bayes model as the classifier. However, these low-level motion features are usually user-habit and scenario-DoF dependent. Therefore, to train the learning model, a user has to perform each gesture several times or the system developer needs to collect a large set of labeled gesture motion data under various scenario from different users. In contrast, Magic Wand utilizes high-level feature (i.e., gesture stroke sequence) so that requires neither scenario-specific wrist motion nor user-specific training.

Wrist tracking based approaches. Some works [13] [14] [9] track the trajectory of wrist movement with motion sensors. The trajectory can be further utilized for wrist gesture recognition. Specifically, ArmTrak [13] establishes a hidden Markov model (HMM) to fuse the motion sensors and the anatomy of arm joints, then tracks the wrist location by estimate the state variables. Combining with the real-time smartwatch facing orientation, MUSE [14] establishes particle filter model to estimate wrist location with the constraints of accelerometer readings and wrist motion model. However, both ArmTrak and MUSE suffer from high computation complexity so that cannot be applied in real-time applications. ArmTroi [9] further optimizes the HMM model of ArmTrak to reduce the computation complexity. In comparison, Magic Wand develops a white-box model for wrist gesture recognition. The computation complexity of the white-box model is much less than HMM model of ArmTroi. Hence, Magic Wand can be applied for the applications (e.g., motion sensing games, VR control) which are time-sensitive.

Overall, in comparison with existing methods, Magic Wand can support various kinds of gestures in a light-weight way. Moreover, Magic Wand can achieve user-independent and real-time wrist gesture recognition with high accuracy.

VII. Conclusion

In this paper, we propose Magic Wand which develops a high-level feature (i.e., stroke sequence) to achieve a plug-and-play wrist gesture recognition on smartwatch. In comparison with existing methods, Magic Wand naturally supports several kinds of gestures without any constraint of user habits. With Magic Wand, a user wearing a smartwatch can freely perform various wrist gestures and interact with many applications.

Acknowledgements

This work was supported in part by the NSFC Grant 61932013, NSFC Grant 61972218, NSFC Grant 61872081 and the ECS grant from Research Grants Council of Hong Kong (Project No. CityU 21203516).

References

[1] Motion capture systems - vicon. http://vicon.com/. Access on May 14, 2019.

[2] H. Abdi and L. J. Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[3] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter. Using mobile phones to write in air. In *Proceedings of Mobisys*, 2011.

[4] L. Ardüser, P. Bissig, P. Brandes, and R. Wattenhofer. Recognizing text using motion data from a smartwatch. In *Proceedings of PerCom Workshops*, 2016.

[5] M. Chen, G. AlRegib, and B.-H. Juang. Feature processing and modeling for 6d motion gesture recognition. *IEEE Transactions on Multimedia*, 15(3):561–571, 2012.

[6] S.-D. Choi, A. S. Lee, and S.-y. Lee. On-line handwritten character recognition with 3d accelerometer. In *Proceedings of ICIA*, 2006.

[7] Y. Jiang, Z. Li, and J. Wang. Ptrack: Enhancing the applicability of pedestrian tracking with wearables. In *Proceedings of ICDCS*, 2017.

[8] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.

[9] Y. Liu, Z. Li, Z. Liu, and K. Wu. Real-time arm skeleton tracking and gesture inference tolerant to missing wearable sensors. In *Proceedings of Mobisys*, 2019.

[10] E. Munguia Tapia. *Using machine learning for real-time activity recognition and estimation of energy expenditure*. PhD thesis, Massachusetts Institute of Technology, 2008.

[11] 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 MobiSys*, 2014.

[12] S. Sen, K. Grover, V. Subbaraju, and A. Misra. Inferring smartphone keypress via smartwatch inertial sensing. In *Proceedings of PerCom Workshops*, 2017.

[13] S. Shen, H. Wang, and R. Roy Choudhury. I am a smartwatch and i can track my user's arm. In *Proceedings of MobiSys*, 2016.

[14] S. Sheng, R. C. Romit, and G. Mahanth. Closing the gaps in inertial motion tracking. In *Proceedings of MobiCom*, 2018.

[15] I. Skog and P. Händel. Calibration of a mems inertial measurement unit. In *XVII IMEKO world congress*, 2006.

[16] E. Thomaz, I. Essa, and G. D. Abowd. A practical approach for recognizing eating moments with wrist-mounted inertial sensing. In *Proceedings of Ubicomp*, 2015.

[17] H. Wang, T. T.-T. Lai, and R. Roy Choudhury. Mole: Motion leaks through smartwatch sensors. In *Proceedings of MobiCom*, 2015.

[18] H. Wen, J. Ramos Rojas, and A. K. Dey. Serendipity: Finger gesture recognition using an off-the-shelf smartwatch. In *Proceedings of CHI*, 2016.

[19] Q. Xia, F. Hong, Y. Feng, and Z. Guo. Motionhacker: Motion sensor based eavesdropping on handwriting via smartwatch. In *Proceedings of INFOCOM Workshops*, 2018.

[20] 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 HotMobile*, 2015.

[21] P. Zhou, M. Li, and G. Shen. Use it free: instantly knowing your phone attitude. In *Proceedings of MobiCom*, 2014.