IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 January 2023, accepted 20 February 2023, date of publication 10 March 2023, date of current version 4 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3255402

RESEARCH ARTICLE

A Novel and Lightweight Real-Time Continuous Motion Gesture Recognition Algorithm for Smartphones

FARHAN SUFYAN[1], SUBHASH SAGAR[2], ZUBAIR ASHRAF[3], SHOAIB NAYEL[4], MOHD SAMEEN CHISHTI[1], AND AMIT BANERJEE[5], (Member, IEEE)

[1]School of Computing Science and Engineering, Galgotias University, Greater Noida, Uttar Pradesh 203201, India
[2]School of Information Technology, Deakin University, Melbourne, VIC 3125, Australia
[3]Department of Computer Engineering and Applications, GLA University, Mathura, Uttar Pradesh 281406, India
[4]International Organization for Migration (IOM), Kabul 1004, Afghanistan
[5]Department of Computer Science, South Asian University, New Delhi 110068, India

Corresponding author: Shoaib Nayel (snayel@iom.int)

ABSTRACT Advancement in smartphones has facilitated the investigation of new modalities of human-machine interaction, including communication through touch, voice, and gestures. In-depth, the researchers examined the problem of recognizing distinct gestures (surface, hand, and motion). However, the gesture recognition algorithm pitches discontinuity while the user performs the subsequent continuous gesture. The discontinuity may occur due to the selection of a delimiter to differentiate between successive motions or the employment of a complex algorithm to boost the accuracy of gesture detection, which takes significant time to recognize the gesture before a user may enter the next gesture. Further, gesture recognition based on template matching, machine learning models, and neural networks requires a lot of storage space, processing resources, or both, which are resource-intensive for smartphones. This research proposes a novel Axis-Point Continuous Motion Gesture (APCMG) recognition algorithm that uses accelerometer sensor data to recognizes continuous motion gestures in real time. The algorithm has low computational complexity and easily implemented on resource-constrained devices with minimal computing cost, memory, and energy. The prime objective of the APCMG is to find the start and end of a gesture from a continuous stream of accelerometer sensor data and recognize the gesture in real-time. To demonstrate the APCMG efficacy, the experimental simulation of the Android application for dialing a phone number is considered. The App acknowledges 12 continuous gestures corresponding to 0 to 9 number, delete, and calls termination. The experimental simulations collected 7500 gestures samples from the 25 volunteers. The algorithm efficiently recognizes isolated and continuous gestures with 95% and 94% accuracy, respectively. The proposed algorithm efficiently recognizes isolated and continuous gestures with minimal energy consumption.

INDEX TERMS Accelerometer, axis point, gesture recognition, continuous motion gesture (CMG).

I. INTRODUCTION

A. BACKGROUND

Smartphones equipped with sophisticated sensors are ushering humans into a new paradigm of human-machine interaction (HMI), which is user-friendly and natural. Smartphones have a variety of inbuilt sensors, such as camera, microphones, accelerometer, and gyroscopes. In HMI, sensors capture human gestures, specifically voice, speech, and movement, which are an alternative way of communication with smartphones [1]. Human gestures are employed for a variety of purposes, like biometric authentication [2], hands-free interaction [3], visually blind persons [4], activity recognition [5], and fall detection [6].

Gestures are meaningful physical movements of the fingers, hands, arms, head, and face – i.e., communicative body

The associate editor coordinating the review of this manuscript and approving it for publication was Michail Kiziroglou.

movements [1]. Tracking various bodily movements and positions is highly valuable for controlling the input parameters of the system. The process through which the system becomes aware of a user's gestures is called gesture recognition [7]. Gesture recognition utilizes the readings from sensors to capture the change of position, orientation, speech, and expressions and then applies mathematical algorithms to interpret the gestures as a command to the system [8]. The two most essential components of a gesture recognition algorithm are *gesture reading* and *gesture identification*. The gesture reading phase captures input information from the sensors when the user performs the gesture. In the gesture identification phase, the collected data is analyzed using the mathematical approach to interpret the gestures performed by the user [9].

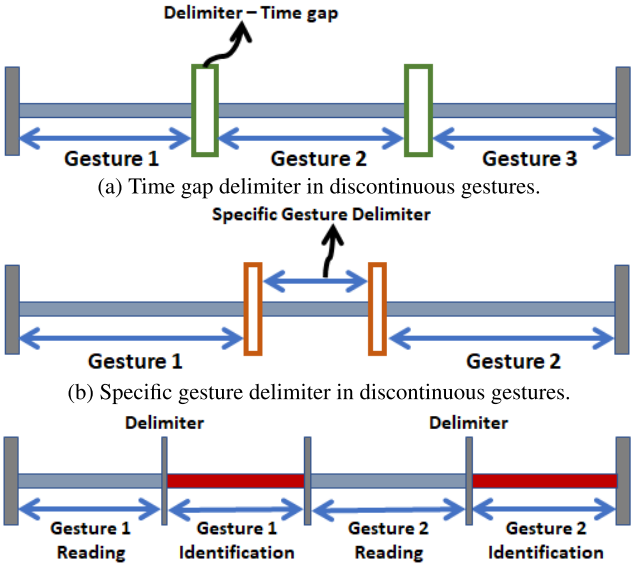
Gestures are broadly classified as 1*D*, 2*D*, and 3*D* gestures. 1*D* gesture refers to the touch buttons or taps used for dialing and increasing/decreasing volume [10]. 2*D* gestures are related to the touch-sensitive screen of a smartphone and are mainly two types: surface gesture (e.g., swipe) [11], and image-based gesture (e.g., fingerprint scan) [12]. 3*D* gestures are utilized to identify the rotation and movements of smartphones. 3*D* gestures are further classified as hand, motion, and air gestures [13]. In hand gestures, the inbuilt video camera of the device captures the hand movement and analyzes it using image processing, and computer vision [14]. Motion gesture refers to the direction of hand-held devices in physical space [15]. However, air gesture (or handwriting gesture) is a special case of motion gesture associated with alphabets, numerals, and symbols written in the air [16], [17]. The accelerometer, orientation, and gyroscope sensors realize the user's physical movement of the smartphone [18]. Special hardware, such as the Leap Motion Controller [19] and the Hand Data Glove [20], are used to collect 3*D* gesture input. In the context of the Internet of Things (IoT), gestures can be used to communicate with connected devices in a wide variety of applications [21], [22]. Gestures are particularly effective when conventional input methods, such as buttons and keyboards, are impractical or inconvenient [23]. Gesture recognition technology has the potential to significantly improve the usability of Internet of Things (IoT) devices, making them more accessible and convenient [24].

The challenges associated with gesture recognition has been extensively explored by researcher worldwide [1], [5], [25]. Techniques like template matching [26], [27], statistical models [28], machine learning [29], and deep neural networks [30] are actively engaged in developing efficient methodologies and increasing the accuracy of gesture recognition algorithms. The emergence of embedded technology shifted the preference from desktop to mobile computing. This transition makes the smartphone a default candidate for the users to perform different types of gestures (motion, surface, or voice) while interacting with various applications rather than using specialized hardware in a limited environment. Due to the advancement in hardware technology, smartphones are becoming more resourceful in terms of processing capability, memory, and battery [31]. However, despite such improvements, smartphones are insufficient for supporting computation-intensive or delay-sensitive applications and are still considered resource-constrained smart devices [32]. It is to be noted that the readings from the embedded sensor in smartphones are continuous in nature. However, the gesture recognition algorithm suffers from discontinuity [33], [34] mainly due to (a) delimiters and (b) complex gesture identification algorithms.

A *delimiter* is used to differentiate between two gesture inputs. The most commonly used delimiters are *time-gap* [35], [36] and *specific gesture* [15]. The time-gap delimiter between two gestures provides a break in the input stream from the sensors, which helps the gesture identification algorithm to analyze the input for two distinct gestures. For example, in FIGURE 1a, a user draws the gesture pattern ''Gesture 1'' and provides a time gap before entering the following input ''Gesture 2''. Hence, there is a discontinuity between the two gestures. In this case, if a user is entering ''Gesture 1'', the system must have some threshold limit to allow the user to complete the gesture. The pause above the threshold limit acts as a delimiter and initiates the recognition algorithm to analyze the input gesture. On the other hand, a specific gesture is used as a delimiter when hand pausing between consecutive gestures is impossible, for instance, in the case of handwriting and air gestures [16]. The extra delimiters break the continuity to perform specific gestures immediately after, e.g., in FIGURE 1b, the time gap delimiter is removed between Gesture 1 and Gesture 2. Still, a specific gesture delimiter is performed to differentiate between the two gestures from the continuous stream of sensor input data. Another reason for discontinuity in the gesture recognition algorithm is due to the inefficiency of the identification phase in interpreting the gesture immediately [37]. However, the discontinuity caused by the time gap and specific gesture delimiter is eliminated. The user must wait for the identification phase to complete before entering the following gestural input. For example, in FIGURE 1c, while the Gesture 1 reading phase is completed, the identification phase for Gesture 1 is started. However, the time duration to identify Gesture 1 is significant, which creates discontinuity, and the user has to wait to input Gesture 2.

B. RELATED WORK

Salalmi et al. [35] proposed a *Tesla-Rapture* gesture recognition using Tesla, a message-passing neural network graph convolution method for mmWave radar point clouds. The model exceeds the accuracy while lowering computational complexity and execution time. Moreover, idle frames were used to distinguish between the gestures in which no significant movement was noticed. Weng et al. [38] proposed a *FaceSight*, a computer vision-based hand-to-facial gesture recognition technique for augmented reality glasses. In their model, an infrared camera was used to bridge augmented reality glasses for enhanced sensing of the lower face and hand movements. The algorithm separates facial areas, detects



(a) Time gap delimiter in discontinuous gestures.

(b) Specific gesture delimiter in discontinuous gestures.

(c) Gesture identification phase in discontinuous gesture

FIGURE 1. Discontinuity in gesture recognition algorithms.

hand-to-face contact, and trains convolutional neural network models to categorize hand-to-face gestures. However, it was found that participants were instructed to remove their hands from their faces before initiating the next hand-to-face gesture, which caused discontinuity. Momotaz and Billah [36] proposed a deep-learning approach for finger gesture recognition that enables tilt-to-pan gestures as a single-handed alternative to inbuilt panning gestures for smartphone users with low vision. The model requires users to rest one finger on the screen to trigger tilt motion with a threshold time of 800 ms, which causes discontinuity.

A customizable motion gesture delimiter that distinguishes irrelevant wrist motion from smartwatch gesture input was developed by Kerber et al. [39]. The model gathered acceleration data from smartwatches and deployed the dynamic time-warping gesture identification to recognize potential motions from a stream of accelerometer readings. However, the continuity breaks between the two motion gestures by a specific movement because one must rotate the wrist outwards and back inwards with the lower arm. Ruiz and Li [15] and Dachselt and Buchholz [40] proposed the idea of specific motion gestures for delimiter for separating gestural input from normal motion. Ruiz & Li designed a DoubleFlip, and Dachselt & Buchholz employed a Throw gesture, which requires a user to move the smartphone back and forth while facing the screen to serve as a specific motion gesture for delimiter in motion-based mobile interaction. DoubleFlip and Throw gestures create delimiters that are highly resistant to false positive situations while maintaining a high detection rate, yet, suffer from discontinuity while performing two or more motion gestures. Angelini et al. [41] presented a touch-and-hold gesture for delimiter for tangible gesture interactive systems. In touch-and-hold, the user has to tap and hold the smartphone's screen while performing a particular motion and release the tap when the gesture is completed. The process needs to repeat when the user is performing the next gesture. Thus, the tapping and release of the screen for every gesture causes discontinuity between the gestures.

Template matching is perhaps the simplest way to recognize gesture movements [27]. The matching of raw sensor input data to the stored template is determined by a matching function that measures the similarity between the stored template and the input. In general, a similarity threshold exists below which the input is discarded for not belonging to any potential gestures or being too distant from the nearest template in a joint space. These strategies are simple to create and computationally straightforward. However, there are theoretical issues with the usage of templates, such as whether they should be constructed based on instances of gestures and how they should be adapted to the demands of specific users regardless of sensor calibration [42]. Template matching lacks the formal approach to training employed by neural networks and statistical classifiers. The success of neural networks and machine learning in gesture detection has garnered much interest [43]. However, thousands of labeled examples are required to train the network for accurate recognition. Too many samples will overtrain the model and force it to disregard previously learned patterns. In addition, an insufficient training data set or other issues, such as the orthogonality of the training vectors, prevent the model from becoming convergent [42]. The learning strategies typically need enormous storage space, labeling samples, computing power, and offloading [44]. Thus, executing such gesture recognition algorithms on resource-constrained smartphones is challenging [45].

C. MOTIVATION AND CONTRIBUTION

Inappropriate selection of delimiters, such as time-gap, specific gesture, and implementation of computation and storage expansive algorithms on resource-restricted smartphones, all contribute to the discontinuity issue in detecting continuous gestures. The aforementioned limitations motivate us to emphasize the development of a lightweight gesture recognition algorithm in terms of storage and processing power that can recognize continuous gestures in real-time. This paper proposes a novel axis-point continuous motion gesture (APCMG) recognition algorithm to identify the continuous motion gesture (CMG) in real time. The primary purpose of APCMG is to present a lightweight computation approach that allows efficient smartphone performance and requires no external resources to process the CMG. The proposed algorithm uses the concept of axis points, which are the specific positions in $3D$ space that indicate substantial changes in the trajectory of smartphone movement. CMGs' templates are stored within smartphones in terms of these axis points, saving storage rather than keeping the whole trajectory as a template. APCMG algorithm recognizes the CMG by spotting the axis points from the input stream and approximating them with the stored axis points ensuring a low computation

time in recognition of gestures. The significant contributions of this paper are summarized as follows:

- Introduces a new algorithm called APCMG, which uses smartphone tri-axis accelerometer data to identify continuous motion gestures (CMG).
- The delimiter used in APCMG allows for continuous gesture recognition without interruptions or the need for specific delimiting gestures.
- APCMG has low processing overhead and can be used for real-time gesture recognition on smartphones.
- APCMG has a high accuracy rate, with an average recognition rate of 95% for isolated gestures and 94% for continuous gestures.
- APCMG does not propagate errors, i.e., the algorithm confines the inaccuracy, so it does not interfere with the identification of the upcoming gestures.
- APCMG is energy-efficient, making it suitable for use on smartphones.

The rest of this paper is organized in the following manner. Section II discusses the proposed APCG algorithms and the main contributions of this work. The implementation details and analysis of essential characteristics of the proposed algorithm are explained in section III and section IV, respectively. Section V discusses performance evaluation by adopting an appropriate smartphone application. Finally, section VI concludes the paper with the future directions.

II. PROPOSED METHOD

This section discusses the proposed axis-point continuous motion gesture (APCMG) algorithm. The flow diagram of the proposed APCMG algorithm is shown in FIGURE 2, and the significant steps as are as follows:

1) ***Motion Detection:*** The accelerometer sensor is used for recognizing the motion of the smartphone in $3D$ space. It measures the acceleration due to gravity or changes in gravitational acceleration of the smartphones around the x, y &, z-axis, which is approximately 9.8 m/s^2, as shown in Figure 3.
2) ***Data preparation:*** While moving the smartphone in $3D$ space, we can define an axis-point as a precise smartphone orientation on the accelerometer axis. Axis-point occurs when the smartphone's orientation overlaps or coincides with any ($\pm x$, $\pm y$, *or* $\pm z$) axis of the accelerometer. Thus, an axis-point coinciding with x-axis, y-axis, and z-axis of the accelerometer is represented by a $3D$ co-ordinate ($\pm x$, $0 \pm \delta$, $0 \pm \delta$), ($0 \pm \delta$, $\pm y$, $0 \pm \delta$) and ($0 \pm \delta$, $0 \pm \delta$, $\pm z$), respectively, where δ is the epistemic constant. The purpose of representing the axis-points as ($\pm x$, $\pm y$, $\pm z$) is because the data received by the accelerometer is expressed via the acceleration along axes. Pictorial demonstrations of the axis-points in APCMG are shown in FIGURE 4. In FIGURE 4a, when the smartphone coincides with the x-axis, the accelerometer readings are considered axis-point. Similarly, in FIGURE 4b, when the smartphone moves from the x-axis and coincides with the y-axis, a smartphone is said to change its position and move to a new axis point, i.e., y-axis.
3) ***Read gesture:*** In the APCMG algorithm, in order to read a gesture, there should be at least a single change-of-axis. Change-of-axis is the movement of the smartphone from one axis-point to another. The movements can be performed multiple times. Graphical demos of the change-of-axis of the smartphone are illustrated in FIGURE 5. A single change-of-axis is shown in FIGURE 5a, where the smartphone is rotated from ($+y \rightarrow +x$)-axis. Similarly, in FIGURE 5b shows two change-of-axes where the smartphone is rotated from ($+y \rightarrow +x$)-axis and then ($+x \rightarrow +z$)-axis.

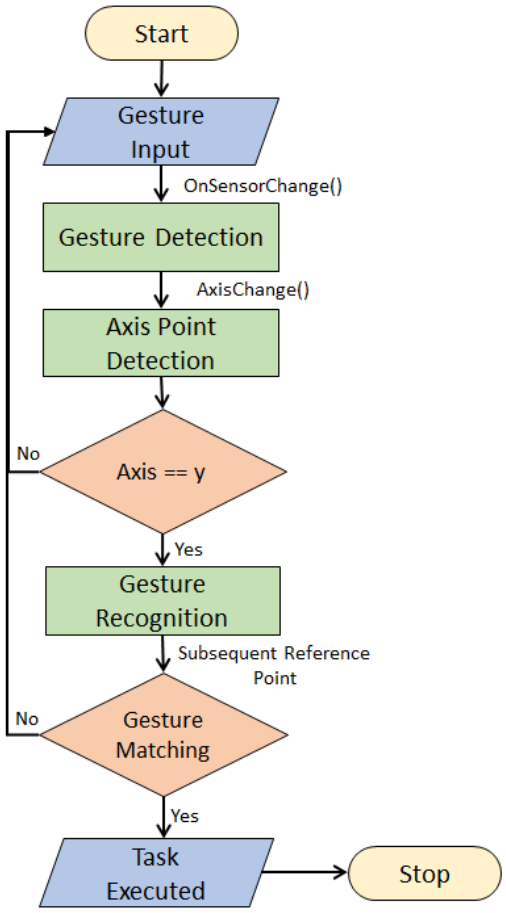


FIGURE 2. Flowchart of proposed APCMG algorithm.

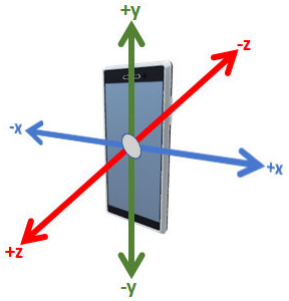
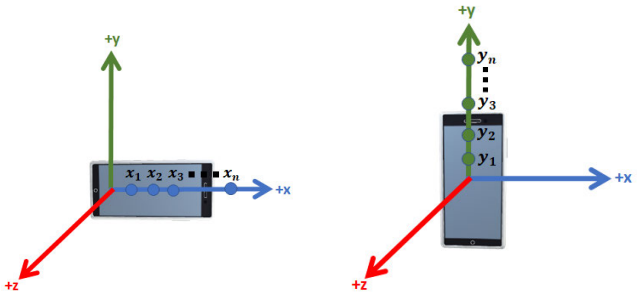
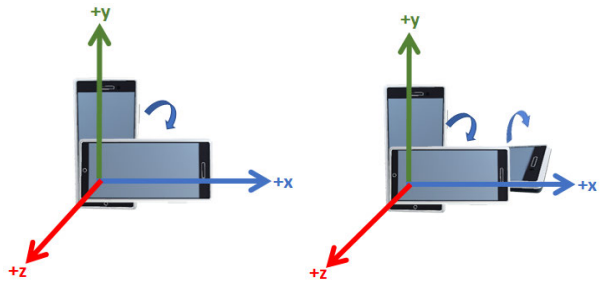


FIGURE 3. Accelerometer sensor.



(a) Smartphone on point x_i (b) Smartphone on point y_i

FIGURE 4. Pictorial demonstrations of Axis-Points of the smartphone.



(a) $2D$ change-of-axis (b) $3D$ change-of-axis

FIGURE 5. Illustration of the change-of-axis.

4) ***Gesture identification:*** The APCMG algorithm uses a delimiter to distinguish between two consecutive gestures. A delimiter's selection is crucial to maintain the continuity between the gestures. Hence, we introduce the concept of reference point in $3D$ space, from where a gesture starts and ends. We also select the reference point as our delimiter because the algorithm can identify two gestures when the user passes the smartphone from the reference point position. The accelerometer's sensitivity is very high, and it captures the changes in the position of the smartphones in real-time; hence the user is not required to place or hold their smartphone on the axis-point position to identify the end of a gesture.
5) ***Gesture recognition:*** The identification phase identifies the change-of-axes by the smartphone in $3D$ space and maintains a dynamic list for the same. As the smartphone passes through the subsequent reference point, the gesture is completed and the axis-points in the list are compared with the template of pre-recorded gestures in the table. Further, the assigned task is executed for every valid gesture. The process continues until the user stops performing the gestures.

III. IMPLEMENTATION OF APCMG ALGORITHM

The implementation of the proposed APCMG algorithm is divided into two modules, referred to as the *Read* and *Identify* modules, see FIGURE 6. The Read module runs on separate threads and receives the continuous stream of data from the accelerometer sensor. Data received from the accelerometer is in the form of float values, each corresponding to x, y, and z-axis. The task of the Read module is to store the continuous

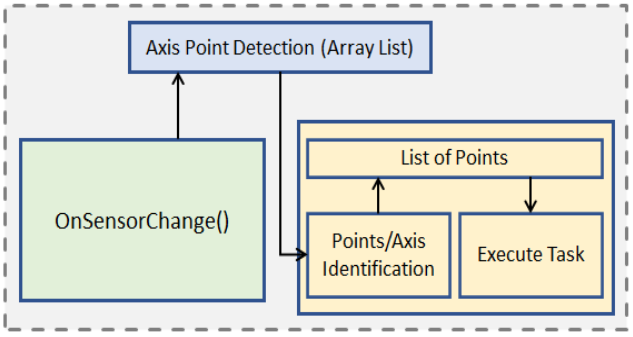


FIGURE 6. Block diagram of APCMG algorithm.

FIGURE 7. Natural way of holding smartphone.

values received from the accelerometer sensor into separate Java *ArrayList* corresponding to each axis. The identify module, running on a independent thread, consists of two functions, namely *pointIdentification()* and *executeTask()*. The pointIdentification() function determines the current position of the smartphone from the accelerometer data. If the smartphone is tantamount to any axis, an axis-point is identified and stored in a separate List (L). In our implementation, we have assumed the reference point as $+y$-axis, as shown in FIGURE 7, which is a natural way of holding the smartphone.

In APCMG, a gesture is a list of the axis-points between two reference points. If the smartphone lies at the reference point (y-axis), it is added to the list; otherwise, the function waits for the smartphone to move to the reference point, discarding the previous values and points. Hence, starting any gesture, there is only a single entry in the list, i.e., $L = \{+y\}$. A gesture consists of n axis-points in L where $n \geq 2$ denotes the least one or more change-of-axes to perform a complete gesture. For example, the accelerometer value is $(0 \pm \delta, +9.8 \pm \delta, 0 \pm \delta)$, if the smartphone is at $+y$-axis. Suppose the smartphone starts moving toward $+x$-axis then the accelerometer value of +y-axis starts decreasing and the value of $+x$-axis increase that is $(0 \pm \delta, +9.8 \pm \delta, 0 \pm \delta) \rightarrow (+9.8 \pm \delta, 0 \pm \delta, 0 \pm \delta)$. Hence, the list of the state is transformed from $\{+y\}$ to $\{+y, +x\}$. Similarly, suppose the smartphone reverts to the reference point; the list changes to $+y, +x, +y$. The subsequent reference point invokes the *executeTask()* function. The executeTask() matches the axis-points in the L with the pre-recorded gestures stored in the SQLite database and executes the assigned task for the gesture, provided it must be a valid gesture. The process continues until the user stops moving the smartphone. The correspondence

between task and gesture is application dependent and must be pre-initialized by the user. For this, we replace the executeTask() method with the *record()* method in Algorithm 1. The record function assigns a task to the gesture in list L and stores it in the *SQLite* database of the smartphone.

Algorithm 1 illustrates the recognition module of the APCMG algorithm. In Algorithm 1, the current orientation of the smartphone is identified by passing the data to *axisChange()* function as described in Algorithm 2. To change an axis, the user must completely bend the smartphone to 90° and align the smartphone to the corresponding axis. However, the perfect 90° of smartphone rotation may not always be possible for the user. Thus, the acceleration values need to be adjusted using a *threshold* (*th*) parameter, as in Algorithm 2. The *th* parameter significantly improves the efficiency of the APCMG algorithm in recognizing gestures. The *th* value can be adjusted according to the requirement of an application and the user's comfort. By analyzing the accelerometer readings, if a user rotates the smartphone around 80°, the accelerometer reading corresponds to the rotating axis is around 9 m/s^2 and the values of the other two axes lie around 1 m/s^2. Thus, by setting the $th = \pm 1$, we can incorporate $80° - 100°$ rotation of smartphone as a gesture. On increasing the value of *th*, say ± 5 or ± 6, the axisChange() function becomes too sensitive and identifies even a slight smartphone rotation as a gesture.

Algorithm 1 Continuous Gesture Recognition

```
L = NULL                                              ▷ List of axis points
prevaxis = NULL                                       ▷ Previous axis point
while true do
    P ← axisChange(x, y, z)
                                                      ▷ +y-axis
    if (P == Start/ReferencePoint && sizeOf(L) == NULL) then
        L ← appendToList(P)
        prevaxis ← P
    end if
                                                      ▷ Add next axis point
    if (sizeOf(L) && P != prevaxis) then
        prevaxis = P
                                                      ▷ Gesture Completed
        if (P == Delimeter(+y − axis)) then
            executeTask(L)
            Clear : L
        end if
        L ← appendToList(P)
    end if
end while
```

Algorithm 2 axisChange() Function

```
Input : x_n, y_n, z_n
th = 1                                                ▷ User dependent threshold.
if +x_n ≤ |0 ± δ ∓ th| && +y_n ≥ |9.8 ± δ ± th| && +z_n ≤ |0 ± δ ∓ th| then
    return +y
end if
if ... then                                           ▷ similar cases to identify other 5 axes.
end if
```

IV. ANALYSIS OF APCMG ALGORITHM

In this section, we undertake a comprehensive examination of the characteristics of the APCMG algorithm. Specifically, we analyze error propagation, mathematical and computational complexities. The key features of the proposed APCMG algorithm are as follows:

- The delimiter employed to distinguish between two gestures must not result in any interruptions in the input stream of data from the sensors. A detailed analysis of this aspect can be found in Section IV-D.
- The algorithm must detect gestures in real-time, with minimal delay in the execution of the identification phase. Further elaboration on this point can be found in section IV-E.

$$T_{recognition} = \Delta$$

A. MATHEMATICAL ANALYSIS

Suppose X, Y, Z are the axes on a 3D-plane and let $\{x_n\}$, $\{y_n\}$, $\{z_n\}$ are the sequences of numbers on X, Y & Z respectively. By experiment (Table 1), we observed that

$$|x_n| + |y_n| + |z_n| = g_n, n \in R \tag{1}$$

If the user holds the smartphone in static condition. Then, the value of $|g_n|$ is around 9.81 m/sec^2. While on the other hand when mobile is in dynamic state, the value of $|g_n|$ for every $n \in R$ lie between [10, 13].

Let x_n, y_n, z_n are the arbitrary numbers at the n^{th} iteration. Then, by our experimental setup, three conditions are possible

- If x_n is ≥ 9 then, $0 \leq y_n, z_n \leq 3$ and mobile phone lies on x-axis.
- If y_n is ≥ 9 then, $0 \leq x_n, z_n \leq 3$ and mobile phone lies on y-axis.
- If z_n is ≥ 9 then, $0 \leq y_n, x_n \leq 3$ and mobile phone lies on z-axis.

B. NUMBER OF POSSIBLE GESTURES

A n axis-point gesture can be formed by adding an extra axis-point to the previous $(n-1)$ axis-point gesture. In general, for every additional axis-point included in the gesture, the number of gestures increases by the factor of 2. Thus, the total numbers of gestures that can be formed up to n axis-point are:

$$\sum_{i=2}^{n} 2^i = 4(2^{n-1} - 1), \ \forall\, n \geq 2$$

TABLE 1. Readings of accelerometer data along different axes while performing 2D axis-point {+y,+x} gesture.

x	y	z	Sum
0.0046	9.8246	0.7045	10.5337
0.1569	9.9031	0.6522	10.7122
0.2206	9.9197	0.57345	10.7137
0.5949	9.9341	0.4608	10.9898
0.7344	8.7842	0.6310	10.1496
1.0763	8.1892	0.3876	9.65310
4.7538	7.8834	0.3738	13.0110
6.6012	5.7832	0.1589	12.5433
7.9069	4.4472	0.6310	12.9851
9.7398	2.2317	0.4734	12.4449
10.8669	0.8010	1.2318	12.8997

C. ANALYSIS OF ERROR PROPAGATION

1) CASE-1: IF AN AXIS-POINT IS NOT APPROPRIATELY RECOGNIZED:

In this case, if a particular gesture whose axis-point other than the delimiter is not appropriately recognized, the error will result in two possibilities:

(a) No gesture is identified because the list of recognized axis-points did not match the recorded gestures list.
(b) A wrong gesture is executed if the recognized axis-points in the list corresponds to a stored gesture list in the database.

For example, suppose a $2D$ axis-point gesture $\{+y, +x\}$; if $+x$-axis is not recognized correctly, no gesture is identified.

2) CASE-2: IF THE DELIMITER IS NOT APPROPRIATELY RECOGNIZED

In this case, if the adjacent gestures on either side of the delimiter are affected. Then, axis-points of adjacent gestures are combined and will match with the recorded gesture or wrong gesture executed. For example, suppose two continuous $2D$ axis-point gestures in the given list $\{+y, +x, +y, +z\}$; if the third element i.e., $+y$ is not recognized, then the list is reduced to $\{+y, +x, +z\}$, which identifies as a $3D$ axis-point gesture, then a wrong gesture is performed.

From the above cases, if the error mentioned above occurs, then at least one and at most two gestures are affected. Hence, we can say that the error is localized in the proposed algorithm and does not propagate and affect the upcoming gestures.

D. CONTINUOUS MOTION GESTURE RECOGNITION ANALYSIS

As previously mentioned in section IV, one of the criteria for the algorithm to recognize the gesture in real-time is that the delimiter between two gestures should not interrupt the input stream of data from the sensors. We start this proof by first defining the context-free grammar (CFG) of the APCMG algorithm that can accept n axis-point gesture input. The grammar accepts and specifies any gesture that can be performed using the axis-method. The constructed grammar is right linear grammar which is also regular.

S –> +yAS_1 | +yBS_2

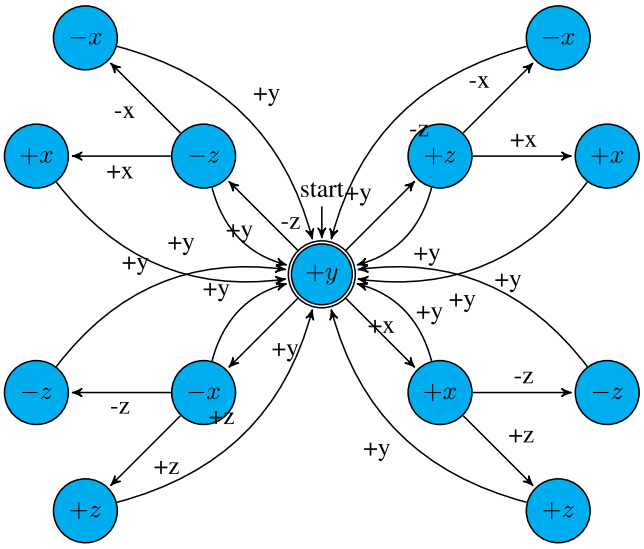


FIGURE 8. DFA of APCMG for *2D* and *3D* axis point gesture sequence.

S_1 –> BS_2 | +y
S_2 –> AS_1 | +y
A –> +x | −x
B –> +z | −z

Now, we built Deterministic state finite automata (DFA) using the above regular grammar for $2D$ and $3D$ axis-point gestures, as shown in FIGURE 8. A DFA is a type of abstract machine that is used to recognize patterns within the input. The machine reads the input symbols one at a time and transitions from one state to another based on the transition function. If the machine ends up in an acceptable state after reading the input, it is said to have recognized the pattern.

The constructed DFA shows continuity within a gesture by showing a path from one state to another for the valid axis inputs. Also, there is no epsilon (ϵ) transition in the DFA that changes the state of the smartphone to another axis without consuming any input. In addition, the constructed DFA has the same initial and final states, showing that a particular gesture starts and finishes in the same location. The final state becomes the initial state for the following gesture input. Thus, these characteristics provide evidence of the absence of discontinuity within a particular gesture or between consecutive gestures.

E. COMPUTATIONAL AND SPACE COMPLEXITY

The proposed APCMG algorithm is specifically engineered for smartphones, enabling continuous and real-time identification of motion gestures. The algorithm addresses the limitation of space requirements by storing gestures in terms of axis-points rather than entire templates. During recognition, data received from the accelerometer is parsed to extract the axis-points, which are then compared to the recorded gesture to execute the assigned task.

From the above discussion, we can infer that the algorithm can recognize gestures of any length in a deterministic time. Thus the solution to the problem lies in the $\mathcal{P}$ class. The

TABLE 2. Digits associated with different gestures in android application.

S.No	Gesture	Digit
1	$\{+y, +x\}$	0
2	$\{+y, -x\}$	2
3	$\{+y, +z\}$	1
4	$\{+y, -z\}$	3
5	$\{+y, +x, +z\}$	4
6	$\{+y, +x, -z\}$	5
7	$\{+y, -x, +z\}$	6
8	$\{+y, -x, -z\}$	7
9	$\{+y, +z, +x\}$	8
10	$\{+y, +z, +x\}$	9
11	$\{+y, -z, +x\}$	*delete*
12	$\{+y, -z, -x\}$	*call disconnect*

computational complexity of the identification phase for a single gesture is equal to the number of accelerometer readings between two reference points of the gesture, represented as c_i. In Algorithm 1, each input reading is processed to determine the axis-point. From the experiment, we found that the number of data points in a single change-of-axis lies within the range of $[35 - 50]$. Therefore, the total number of operations required for reading N gestures is $N * c$, where $c = max(c_i)$, where i are the maximum number of data points from the accelerometer in each change-of-axis. As a result, the amortized cost analysis of the APCMG algorithm is constant, i.e., $\mathcal{O}(1)$, allowing for real-time gesture identification. Amortized cost analysis is a method of determining the average cost of an operation over a sequence of operations rather than just the cost of a single operation [46].

V. PERFORMANCE EVALUATION

To estimate the effectiveness of the APCMG algorithm, the experimental simulation of the android-based application for dialing a phone number is considered. The application recognizes twelve continuous gestures; each corresponds to a number $(0 - 9)$, and the remaining two are for delete and call terminate (see Table 2). The proximity sensor is used to dial the number when the user places the phone in its ear. Next, we present a graphical representation of accelerometer data along all three axes for each $2D$ and $3D$ axis-point gesture. Further, we explain the working of the APCMG algorithm with a real-time experiment setup and the performance of detecting axis-points as the smartphone changes its orientation around different axes, as discussed previously.

A. TYPES OF GESTURES

In the APCMG algorithm, the relationships between the axis-points and types of gestures are discussed as follows:

1) 2D AXIS-POINT GESTURE

A *2D axis-point* gesture is the movement of a smartphone from the reference point $(+y)$ to any other axis. The axis-points of a 2D axis-point gesture is $\{+y, p\}$, where $p = \{+x, -x, +z, -z, -y\}$. Therefore, the total number of 2D axis-point gestures are 5. Here, we ignore the $(-y)$-axis because the combination $\{+y, -y\}$ is not possible. $(+y \leftrightarrow -y)$, $(+x \leftrightarrow -x)$ and $(+z \leftrightarrow -z)$, directly without crossing another axis. Hence, out of five, there are only four possible 2D axis-point gestures. The movement of smartphone from $(+y \rightarrow +x)$-axis and the accelerometer data received during the movement is given in FIGURE 9a. Graph depicted in FIGURE 9a shows that the smartphone is initially static on the reference point, i.e., $+y$-axis. Hence, the acceleration along the $+y$-axis is close to $+10\ m/s^2$ and readings of x-axis and z-axis is close to $0\ m/s^2$. As the smartphone moves towards x-axis, the acceleration on $+y$-axis decreases and the acceleration on x-axis starts increasing, simultaneously. When the smartphone is completely rotated to x-axis, the acceleration on the x-axis is close to $10\ m/s^2$ and acceleration on other two axes is close to $0\ m/s^2$. The solid line in the middle shows the change-of-axis from $(+y \rightarrow +x)$-axis. Similarly, mobile reverts to the reference point, acceleration on x-axis decreases and $+y$-axis increases. Algorithms 1 identify the y-axis as a delimiter to mark the end of the $2D$ axis-point gesture. Remaining $2D$ axis-point gestures and their corresponding graphs are given in FIGURE 9.

2) 3D AXIS-POINT GESTURES

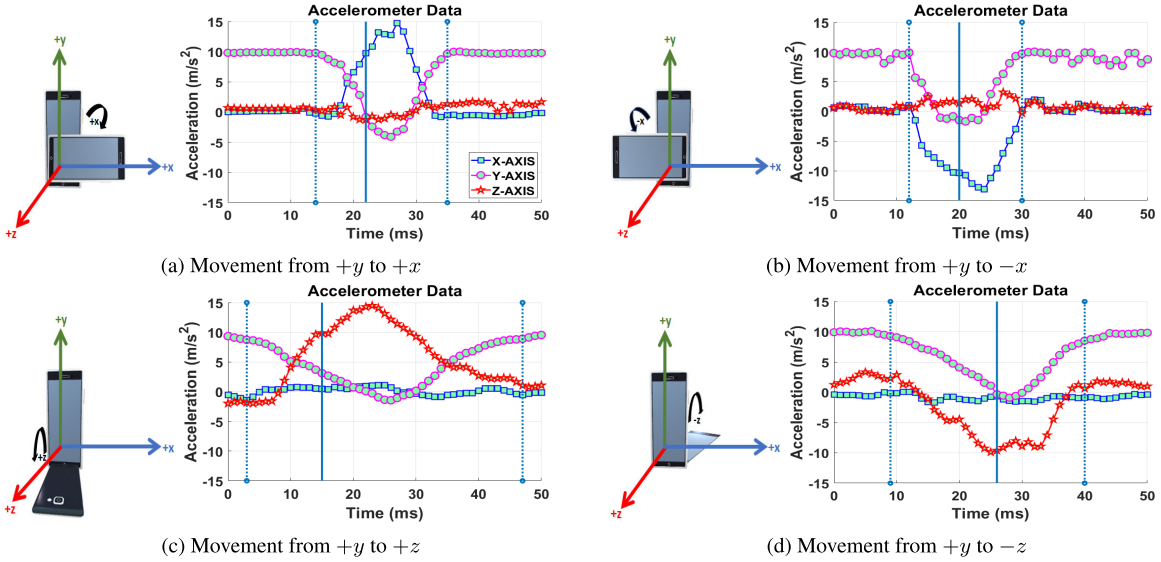
A *3D axis-point* can be expressed as $\{+y, p, q\}$. It is formed by moving the smartphone from the reference point to an axis $p = \{+x, -x, +z, -z\}$, i.e., as in the $2D$ gesture. Then, change the orientation to any of the remaining two axes to complete the $3D$ axis-point gestures. For example, to perform a $3D$ axis-point, the smartphone first moves from the $+y$-axis to any axis in p, say the $+x$-axis. After reaching the $+x$-axis, a user can either move towards $+z$ or $-z$-axis to complete a $3D$ gesture. Hence, for every $2D$ axis-point gesture user has two options for selecting the third point. So, there are eight $3D$ gestures. FIGURE 10 shows the movement of smartphone from $(+y \rightarrow +x \rightarrow +z)$-axis, forming a $3D$ axis-point gesture. The $3D$ axis-point gesture recognition is similar to the $2D$ axis-point gesture with two motions. The rest of the $3D$ gestures and their corresponding graphs are shown in FIGURE 10b-10h.

B. EXPERIMENTAL SETUP

To test our APCMG algorithm, we evaluate the smartphone application from various manufacturers with in-built accelerometer sensors. For our experiment, we recruited 25 volunteers from our university's campus, consisting of twenty males and five female postgraduate and research scholars from various disciplines in the age group $22 - 28$ years. Three volunteers are left-handed, and the remaining are right-handed. We installed Android application on the volunteer's devices to test the application on different smartphones. The result discuss below shows the average performance of each experiment.

C. RESULT EVALUATIONS AND DISCUSSION

The experimental evaluation is divided into two parts. We begin by evaluating the performance of isolated $2D$ axis-point and $3D$ axis-point gestures. We installed our android application on the volunteer's devices to test the application on different smartphones. We assess each gesture's performance, which makes volunteers familiar with the gestures



(a) Movement from $+y$ to $+x$

(b) Movement from $+y$ to $-x$

(c) Movement from $+y$ to $+z$

(d) Movement from $+y$ to $-z$

FIGURE 9. 2D axis-point demonstration of gesture recognition with their accelerometer data.

TABLE 3. Experimental results: 2*D* axis point gestures using APCMG algorithm.

Users	Gestures				Total
	{+y,+x}	{+y,-x}	{+y,+z}	{+y,-z}	
User 1	25	25	25	24	99
User 2	24	25	25	23	97
User 3	24	25	25	22	96
User 4	21	24	25	22	92
User 5	24	25	25	23	97
User 6	24	25	25	25	99
User 7	25	25	25	24	99
User 8	22	25	25	25	97
User 9	23	25	25	24	97
User 10	23	25	25	25	98
User 11	24	24	25	25	98
User 12	19	25	25	25	94
User 13	23	25	25	25	98
User 14	25	25	25	25	100
User 15	25	25	25	23	98
User 16	24	25	25	25	99
User 17	25	25	25	25	100
User 18	24	25	25	20	94
User 19	20	25	25	24	94
User 20	24	25	25	25	99
User 21	25	25	25	25	100
User 22	22	25	25	25	97
User 23	25	25	25	25	100
User 24	25	25	25	23	98
User 25	25	25	25	25	100

and allows us to test the user's comfortableness with different gestures. The second part of our evaluation consists of continuous gestures where we verify the recognition of the numeric string of different lengths entered by the volunteers.

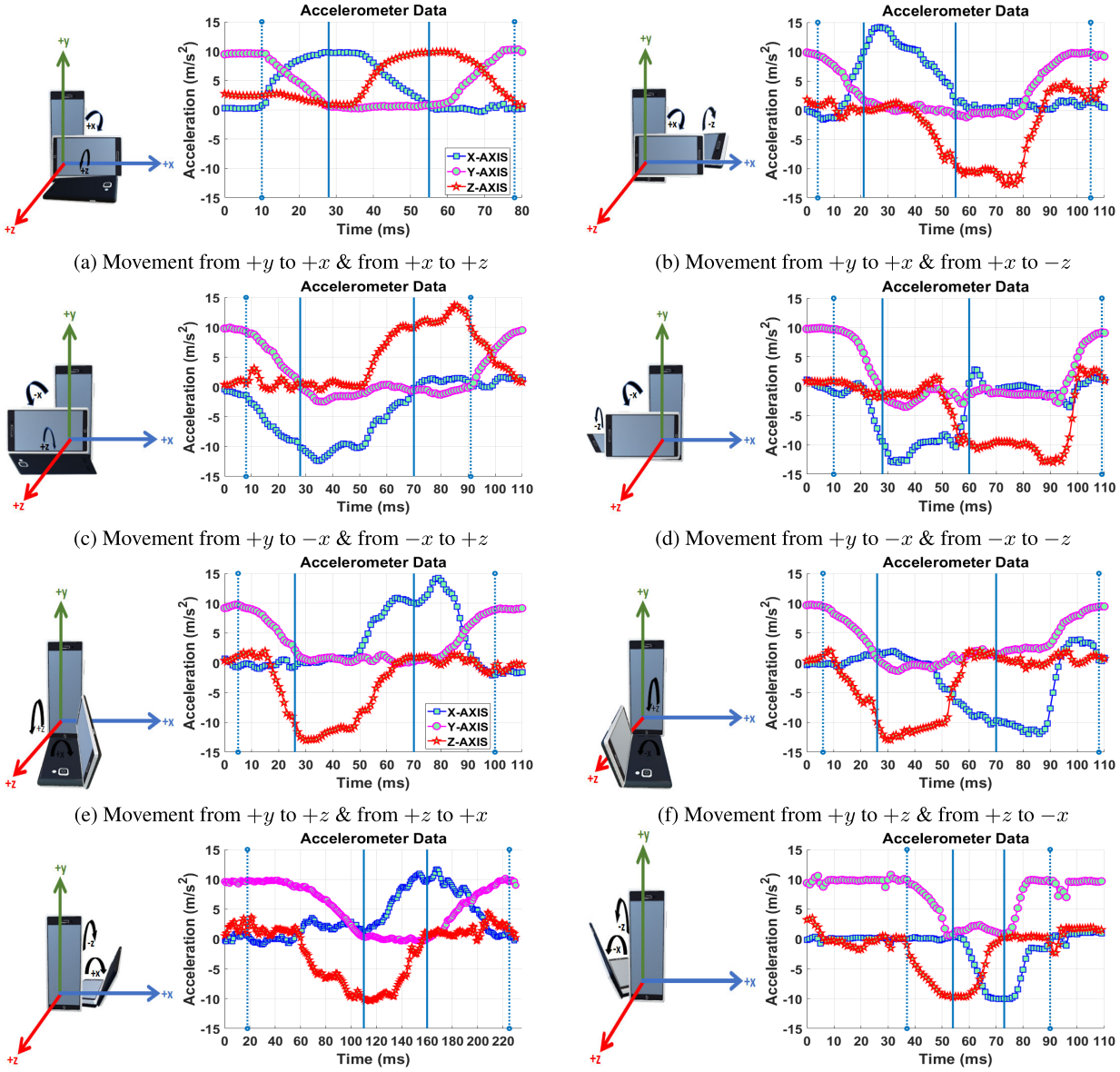
1) RESULTS OF 2D AND 3D AXIS-POINT ISLOATED GESTURES

For 2D axis-point gesture evaluation, we demonstrated four 2D axis-point gestures (see FIGURE 9) to the volunteers and gave them $2-3$ minutes to get acquainted. Then the participants are asked to repeat each 2*D* axis-point gesture 25 times, in different permutations. Since there are four 2*D* axis-point gestures, each member performs a total of $4\times25=100$ gestures. The time required for the data collection task lasted approximately $8-10$ min for an individual participant. The application transfers this data to a remote server for evaluation. In total, a data set comprising of 2500 gestures is collected from all 25 participants and the total time spent for this experiment is around 3.5 hours. The experimental result shows that the APCMG algorithm successful recognition rate of 2*D* axis-point gestures is around 98%, as shown in Table 3. Such a high accuracy was achieved because of only four 2D axis-point gestures, which can be easily remembered and performed by the user.

Further, we used the same volunteers for evaluating isolated 3*D* axis-point gestures, as shown in FIGURE 10. As before, we demonstrate eight 3*D* axis-point gestures and allowed them to get familiar with them. Again every participant is asked to repeat each 3*D* axis-point gesture 25 times, in different permutations. The experiment took approximately $10-15$ minutes and produced a dataset of $25\times8=200$ gestures for each participant. In total, we collected a dataset comprising of 5000 gestures from 25 participants; the total time required for the experiment was around 6 hours. Table 4 presents the experimental result for 3*D* axis-point gestures while indicating a successful recognition rate of around 94%.

2) RESULTS OF 2D AND 3D AXIS-POINT CONTINUOUS GESTURE

The isolated gesture experiment is designed with the intent to make the volunteers familiar and comfortable with 2*D* axis-point and 3*D* axis-point gestures. Now, volunteers are asked to input three increasing-length numerical strings, comprising both 2*D* axis-point and 3*D* axis-point gestures. The three



(a) Movement from $+y$ to $+x$ & from $+x$ to $+z$

(b) Movement from $+y$ to $+x$ & from $+x$ to $-z$

(c) Movement from $+y$ to $-x$ & from $-x$ to $+z$

(d) Movement from $+y$ to $-x$ & from $-x$ to $-z$

(e) Movement from $+y$ to $+z$ & from $+z$ to $+x$

(f) Movement from $+y$ to $+z$ & from $+z$ to $-x$

(g) Movement from $+y$ to $-z$ & from $-z$ to $+x$

(h) Movement from $+y$ to $-z$ & from $-z$ to $-x$

FIGURE 10. 3D axis-point demonstration of gesture recognition with their accelerometer data.

strings are (a) 0187 (b) 340752 (c) volunteer's mobile number (10 digit number). Also, volunteers are asked to continuously carry out the gesture corresponding to the character until the application ultimately recognizes the complete string; this may include the wrong input or not successfully recognized gestures.

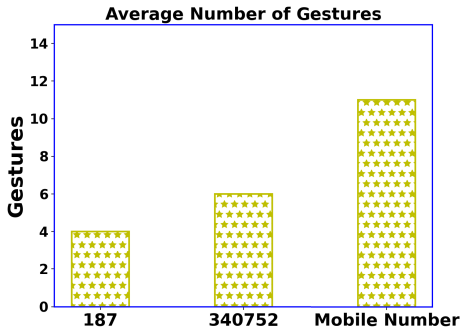
Figure. 11a shows the average number of gestures required by the volunteers to input the given two strings and their mobile number. The average time taken by the users to input the strings is shown in Figure 11b. The results depict that the average number of gestures and average time increases with increasing string size. Strings (a), (b) and (c) requires an average of 5, 7 and 12, gestures and an average time of 5, 8 and 15 seconds. The overall recognition rate of continuous gesture is 94%, which is calculated by successful gestures out of total gestures performed. The result suggests that volunteers can easily learn and memorize the digits associated with the different gestures and can comfortably input the given strings and their mobile number in a relatively quick time.

D. RESULTS OF POWER CONSUMPTION

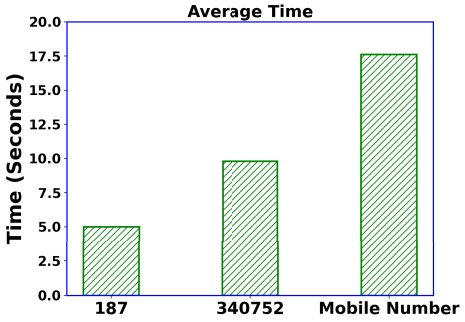
To recognize gestures within the smartphones, the gesture recognition algorithm should be lightweight and not consume too much battery life. Otherwise, if the application drains the battery life, it is not wise to install it on the smartphones for gesture recognition. To measure the performance of the APCMG algorithm, we install our android application on various smartphones from different configurations and manufacturers. During the experiment, the smartphone was fully

TABLE 4. Experimental results: 3*D* axis point gestures using APCMG algorithm.

Users	Gestures								Total
	{+y,+x,+z}	{+y,+x,-z}	{+y,-x,+z}	{+y,-x,+z}	{+y,+z,+x}	{+y,+z,-x}	{+y,-z,+x}	{+y,-z,-x}	
User 1	23	25	24	23	25	25	24	22	191
User 2	23	22	25	22	93	25	22	21	183
User 3	22	24	23	24	21	21	23	20	178
User 4	21	25	25	24	25	25	24	21	190
User 5	21	24	24	23	25	25	24	20	186
User 6	23	22	23	20	25	25	21	23	182
User 7	23	24	24	22	25	24	22	21	185
User 8	24	20	20	24	25	25	23	22	183
User 9	24	24	24	25	25	25	22	22	191
User 10	24	25	24	24	25	24	24	20	190
User 11	23	22	25	24	25	23	24	21	187
User 12	23	24	25	25	25	25	23	25	195
User 13	25	23	25	24	25	24	21	25	192
User 14	25	23	25	21	23	25	24	22	188
User 15	24	24	25	23	24	22	23	23	188
User 16	25	24	24	24	25	25	22	22	191
User 17	25	21	25	23	24	23	22	22	185
User 18	24	24	24	24	25	25	21	23	190
User 19	25	23	24	24	25	25	20	24	190
User 20	25	20	24	23	23	25	25	24	189
User 21	23	25	23	22	24	25	24	23	189
User 22	25	23	23	25	24	24	25	22	190
User 23	25	25	24	21	25	22	20	23	185
User 24	23	24	24	24	25	25	23	24	192
User 25	24	21	24	23	25	25	21	23	182



(a) Average number of gestures required for the successfully complete the strings using continuous gestures



(b) Average number of time required for the successfully complete the strings using continuous gestures

FIGURE 11. Average number of gestures and time graph in continuous gesture evaluation experiment.

charged, and the display was on; we found that the total battery consumption by the application on different devices was around 6% − 8%. Hence, the algorithm does not gobble battery of smartphones; indeed, it is a power effective method as well, so the user can perform various tasks using the APCMG algorithm by recognizing CMG for an extended period of time.

VI. CONCLUSION AND FUTURE WORK

This paper proposes a novel axis-point continuous motion gesture (APCMG) recognition algorithm using axis-points from the accelerometer sensor of the smartphone in real-time. The two effective procedures of the APCMG algorithm are the read and the recognition process. The read process reads the gesture from the smartphone's orientation, and the recognition process identifies the gesture. The APCMG algorithm is designed explicitly for resource-constrained devices with low computational complexity; the amortized cost analysis is constant, i.e., $O(1)$. Further, storage requirements and energy consumption are also minimal for the APCMG algorithm. The mathematical and empirical analysis of the algorithm evidence that it did not allow a particular gesture's discontinuity and error propagation to affect subsequent gestures' recognition while receiving continuous input data from the sensor. The experimental study was conducted by designing an Android App for the APCMG recognition algorithm of dialing phone numbers using hand-held smartphone movement with in-built accelerometer sensors. The application recognizes twelve continuous gestures corresponding to numbers $0-9$, delete, and call terminate. A team of 25 volunteers has formed, and the outcomes reveal that users can easily remember both 2*D* and 3*D* gestures. The APCMG algorithm has a high recognition rate of 98% and 94% for the isolated 2*D* and 3*D* gestures and 94% for continuous 2*D* and 3*D* gestures as well. However, one of the major limitations for the user is remembering the axis rotation as the number of

axis-points increases in a given gesture, i.e., 4*D*, 5*D* axis-point or 6*D* axis-point gestures, etc. Hence, in the future, we intend to reduce the complexity of remembering the points in a gesture for the user. In addition, we also include the analysis of the confusion matrix for true acceptance rate and false acceptance rate, analysis of the equal error rate from the receiver operating characteristics curve, and other results depending on the application under consideration.

REFERENCES

[1] N. Mohamed, M. B. Mustafa, and N. Jomhari, "A review of the hand gesture recognition system: Current progress and future directions," *IEEE Access*, vol. 9, pp. 157422–157436, 2021.

[2] X. Jiang, X. Liu, J. Fan, X. Ye, C. Dai, E. A. Clancy, D. Farina, and W. Chen, "Enhancing IoT security via cancelable HD-sEMG-based biometric authentication password, encoded by gesture," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16535–16547, Nov. 2021.

[3] K.-B. Park, S. H. Choi, J. Y. Lee, Y. Ghasemi, M. Mohammed, and H. Jeong, "Hands-free human–robot interaction using multimodal gestures and deep learning in wearable mixed reality," *IEEE Access*, vol. 9, pp. 55448–55464, 2021.

[4] S. M. Aslam and S. Samreen, "Gesture recognition algorithm for visually blind touch interaction optimization using crow search method," *IEEE Access*, vol. 8, pp. 127560–127568, 2020.

[5] B. van Amsterdam, M. J. Clarkson, and D. Stoyanov, "Gesture recognition in robotic surgery: A review," *IEEE Trans. Biomed. Eng.*, vol. 68, no. 6, pp. 2021–2035, Jun. 2021.

[6] P.-Y. Tsai, Y.-C. Yang, Y.-J. Shih, and H.-Y. Kung, "Gesture-aware fall detection system: Design and implementation," in *Proc. IEEE 5th Int. Conf. Consum. Electron.-Berlin (ICCE-Berlin)*, Sep. 2015, pp. 88–92.

[7] L. Jiashan and L. Zhonghua, "Dynamic gesture recognition algorithm combining global gesture motion and local finger motion for interactive teaching," *IEEE Access*, early access, Mar. 12, 2021, doi: 10.1109/ACCESS.2021.3065849.

[8] G. Li, H. Wu, G. Jiang, S. Xu, and H. Liu, "Dynamic gesture recognition in the Internet of Things," *IEEE Access*, vol. 7, pp. 23713–23724, 2018.

[9] H. A. Jaber, M. T. Rashid, and L. Fortuna, "Robust hand gesture identification using envelope of HD-sEMG signal," in *Proc. Int. Conf. Inf. Commun. Technol.*, New York, NY, USA, Apr. 2019, pp. 203–209.

[10] H. Shin, J.-M. Lim, C. Oh, M. Kim, H.-T. Jeong, and J. Son, "Performance comparison of tap gestures on small-screen touch devices," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2015, pp. 120–121.

[11] N. Magrofuoco, P. Roselli, and J. Vanderdonckt, "Two-dimensional stroke gesture recognition: A survey," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–36, Jul. 2021.

[12] Y. Meng, D. S. Wong, R. Schlegel, and L.-F. Kwok, "Touch gestures based biometric authentication scheme for touchscreen mobile phones," in *Information Security and Cryptology*, M. Kutyłowski and M. Yung, Eds. Berlin, Germany: Springer 2013, pp. 331–350.

[13] Y. Wang, S. Wang, M. Zhou, Q. Jiang, and Z. Tian, "TS-I3D based hand gesture recognition method with radar sensor," *IEEE Access*, vol. 7, pp. 22902–22913, 2019.

[14] S. Zhang and S. Zhang, "A novel human-3DTV interaction system based on free hand gestures and a touch-based virtual interface," *IEEE Access*, vol. 7, pp. 165961–165973, 2019.

[15] J. Ruiz and Y. Li, "DoubleFlip: A motion gesture delimiter for mobile interaction," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, 2011, pp. 2717–2720.

[16] J. Zhang, Y. Li, H. Xiong, D. Dou, C. Miao, and D. Zhang, "HandGest: Hierarchical sensing for robust-in-the-air handwriting recognition with commodity WiFi devices," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 19529–19544, Oct. 2022.

[17] G. Li and H. Sato, "Sensing in-air signature motions using smartwatch: A high-precision approach of behavioral authentication," *IEEE Access*, vol. 10, pp. 57865–57879, 2022.

[18] H. P. Gupta, H. S. Chudgar, S. Mukherjee, T. Dutta, and K. Sharma, "A continuous hand gestures recognition technique for human-machine interaction using accelerometer and gyroscope sensors," *IEEE Sensors J.*, vol. 16, no. 16, pp. 6425–6432, Aug. 2016.

[19] K. Rasheed, S. Saad, L. Shahzad, S. Ammad, A. Ali, and I. Badshah, "Application of gesture data recognition in a human-interactive leap motion sensor chair," in *Proc. Int. Conf. Data Anal. Bus. Ind. (ICDABI)*, Oct. 2021, pp. 567–571.

[20] M. Lee and J. Bae, "Deep learning based real-time recognition of dynamic finger gestures using a data glove," *IEEE Access*, vol. 8, pp. 219923–219933, 2020.

[21] D. Wu, Y. Zeng, R. Gao, S. Li, Y. Li, R. C. Shah, H. Lu, and D. Zhang, "WiTraj: Robust indoor motion tracking with WiFi signals," *IEEE Trans. Mobile Comput.*, early access, Dec. 9, 2021, doi: 10.1109/TMC.2021.3133114.

[22] R. Gao, M. Zhang, J. Zhang, Y. Li, E. Yi, D. Wu, L. Wang, and D. Zhang, "Towards position-independent sensing for gesture recognition with Wi-Fi," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol.*, vol. 5, no. 2, pp. 1–28, Jun. 2021.

[23] R. Kang, A. Guo, G. Laput, Y. Li, and X. Chen, "Minuet: Multimodal interaction with an Internet of Things," in *Proc. Symp. Spatial User Interact.*, New York, NY, USA, Oct. 2019, pp. 1–10.

[24] S. He, A. Zhang, and M. Yan, "Voice and motion-based control system: Proof-of-concept implementation on robotics via Internet-of-Things technologies," in *Proc. ACM Southeast Conf.*, New York, NY, USA, 2019, pp. 102–108.

[25] K. Kudrinko, E. Flavin, X. Zhu, and Q. Li, "Wearable sensor-based sign language recognition: A comprehensive review," *IEEE Rev. Biomed. Eng.*, vol. 14, pp. 82–97, 2021.

[26] J. Zhang, Y. Li, W. Xiao, and Z. Zhang, "Online spatiotemporal modeling for robust and lightweight device-free localization in nonstationary environments," *IEEE Trans. Ind. Informat.*, early access, Nov. 2, 2022, doi: 10.1109/TII.2022.3218666.

[27] M. Okawa, "Template matching using time-series averaging and DTW with dependent warping for online signature verification," *IEEE Access*, vol. 7, pp. 81010–81019, 2019.

[28] A. Calado, P. Roselli, V. Errico, N. Magrofuoco, J. Vanderdonckt, and G. Saggio, "A geometric model-based approach to hand gesture recognition," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 10, pp. 6151–6161, Oct. 2022.

[29] J. Li, S. Ray, V. Rajanna, and T. Hammond, "Evaluating the performance of machine learning algorithms in gaze gesture recognition systems," *IEEE Access*, vol. 10, pp. 1020–1035, 2022.

[30] M. Chmurski, M. Zubert, K. Bierzynski, and A. Santra, "Analysis of edge-optimized deep learning classifiers for radar-based gesture recognition," *IEEE Access*, vol. 9, pp. 74406–74421, 2021.

[31] A. Banerjee, F. Sufyanf, M. S. Nayel, and S. Sagar, "Centralized framework for controlling heterogeneous appliances in a smart home environment," in *Proc. Int. Conf. Inf. Comput. Technol. (ICICT)*, Mar. 2018, pp. 78–82.

[32] F. Sufyan and A. Banerjee, "Computation offloading for distributed mobile edge computing network: A multiobjective approach," *IEEE Access*, vol. 8, pp. 149915–149930, 2020.

[33] H. Zhao, S. Wang, G. Zhou, and D. Zhang, "Ultigesture: A wristband-based platform for continuous gesture control in healthcare," *Smart Health*, vol. 11, pp. 45–65, Jan. 2019.

[34] Y. Zhao, Y. Zhao, H. Tu, Q. Huang, W. Zhao, and W. Jiang, "Motion gesture delimiters for smartwatch interaction," *Wireless Commun. Mobile Comput.*, vol. 2022, Jul. 2022, Art. no. 6879206.

[35] D. Salami, R. Hasibi, S. Palipana, P. Popovski, T. Michoel, and S. Sigg, "Tesla-rapture: A lightweight gesture recognition system from mmWave radar sparse point clouds," *IEEE Trans. Mobile Comput.*, early access, Feb. 23, 2022, doi: 10.1109/TMC.2022.3153717.

[36] F. Momotaz and S. M. Billah, "Tilt-explore: Making tilt gestures usable for low-vision smartphone users," in *Proc. 34th Annu. ACM Symp. User Interface Softw. Technol.*, New York, NY, USA, Oct. 2021, pp. 1154–1168.

[37] P. Zhao, "A review on machine learning and gesture recognition," in *Proc. Int. Conf. Comput. Data Sci. (CDS)*, Aug. 2020, pp. 425–428.

[38] Y. Weng, C. Yu, Y. Shi, Y. Zhao, Y. Yan, and Y. Shi, "FaceSight: Enabling hand-to-face gesture interaction on AR glasses with a downward-facing camera vision," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, New York, NY, USA, May 2021, pp. 1–14.

[39] F. Kerber, P. Schardt, and M. Löchtefeld, "WristRotate: A personalized motion gesture delimiter for wrist-worn devices," in *Proc. MUM*, New York, NY, USA, 2015, pp. 218–222.

[40] R. Dachselt and R. Buchholz, "Natural throw and tilt interaction between mobile phones and distant displays," in *Proc. CHI Extended Abstr. Hum. Factors Comput. Syst.*, New York, NY, USA, Apr. 2009, pp. 3253–3258.
[41] L. Angelini, D. Lalanne, E. Hoven, O. Khaled, and E. Mugellini, "Move, hold and touch: A framework for tangible gesture interactive systems," *Machines*, vol. 3, no. 3, pp. 173–207, Aug. 2015.
[42] R. Watson, "A survey of gesture recognition techniques," Dept. Comput. Sci., Trinity College Dublin, Dublin, Ireland, Tech. Rep. TCD-CS-93-11, 1993. [Online]. Available: https://www.scss.tcd.ie/publications/tech-reports/tr-index.93.php
[43] D. E. Alonso-Blas and S. Genaim, "On the limits of the classical approach to cost analysis," in *Proc. 19th Int. Static Anal. Symp.* Berlin, Germany: Springer-Verlag, 2012, pp. 405–421.
[44] A. Jaramillo-Yánez, M. E. Benalcázar, and E. Mena-Maldonado, "Real-time hand gesture recognition using surface electromyography and machine learning: A systematic literature review," *Sensors*, vol. 20, no. 9, p. 2467, Apr. 2020.
[45] F. Sufyan and A. Banerjee, "Computation offloading for smart devices in fog-cloud queuing system," *IETE J. Res.*, vol. 69, no. 3, pp. 1509–1521, 2023, doi: 10.1080/03772063.2020.1870876.
[46] R. E. Tarjan, "Amortized computational complexity," *SIAM J. Algebr. Discrete Methods*, vol. 6, no. 2, pp. 306–318, Apr. 1985.

FARHAN SUFYAN received the Ph.D. degree in computer science from South Asian University (SAU), New Delhi, India. He is currently an Assistant Professor with the Department of Computer Applications, School of Computing Science and Engineering, Galgotias University, Uttar Pradesh, India. His research interests include the Internet of Things, cloud computing, edge computing, and blockchain.

SUBHASH SAGAR received the Ph.D. degree from the School of Computing, Macquarie University, Sydney, in 2022. He is currently working as an Associate Research Fellow with the School of Information Technology, Deakin University, Melbourne. Before moving to Macquarie University, he was worked as a Faculty Member with the Department of Computer Science, National University of Computer and Emerging Sciences, Karachi, Pakistan, from 2017 to 2019. He has published a number of papers in conferences and journals including IEEE TNSM, SenSys, ICC, and GLOBECOM. His current research interests include the Social Internet of Things, Trust Management, and Cybersecurity for Federated Learning. He is a member of ACM.

ZUBAIR ASHRAF received the Ph.D. degree in computer science from South Asian University, New Delhi, India, in February 2020. He is currently an Assistant Professor with the Department of Computer Engineering and Applications, GLA University, Uttar Pradesh, India. His research interests include multi-criteria decision making, evolutionary optimization, nature-inspired intelligent computation, deep learning, and fuzzy systems. He is the IEEE Young Professional and an Active Member of several societies, including the IEEE Computational Intelligence Society and EUSFLAT. He is also an Active Reviewer of journals such as IEEE TRANSACTIONS ON FUZZY SYSTEM, *Soft Computing*, *Applied Soft Computing*, *Journal of Applied Mathematics*, and the *International Journal of Intelligent Systems*.

SHOAIB NAYEL received the master's degree in computer science from the South Asian University, India. He is currently with the International Organization for Migration (IOM) as a Data Analyst. He has a strong passion for utilizing data, applied mixed-method research, and contextual knowledge to make a positive impact on the lives of people in need, particularly in the field of humanitarian migration. His work has contributed significantly to improving the effectiveness of programs, policies, and interventions for vulnerable populations affected by forced displacement, conflict, and natural disasters. He has published research articles in peer-reviewed journals and a sought-after speaker on topics related to the Internet of Things (IoT) and machine learning. He is also an Active Member of the research community and also committed to staying up-to-date with the latest developments in his field.

MOHD SAMEEN CHISHTI received the Ph.D. degree in computer science from South Asian University (SAU), New Delhi, India. He is currently an Assistant Professor with the Department of Computer Applications, School of Computing Science and Engineering, Galgotias University, Uttar Pradesh, India. His research interests include blockchain, web 3.0, and metaverse.

AMIT BANERJEE (Member, IEEE) received the Ph.D. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2009. After that, he worked for two years as an Engineer with SoC Technology Center, Industrial Technology Research Institute (ITRI), Taiwan. He is currently an Assistant Professor with the Department of Computer Science, South Asian University (SAU), New Delhi, India. He has authored or coauthored papers in peer-reviewed journals and conferences, including IEEE TRANSACTIONS. His current research interests include distributed computing, the Internet of Things, and edge computing.