Ultigesture: A Wristband-based Platform for Continuous Gesture Control in Healthcare

Hongyang Zhao[a], Shuangquan Wang[a], Gang Zhou[a,*], Daqing Zhang[b]

[a]*Computer Science Department, College of William and Mary, United States*
[b]*Institut Mines-Télécom/Télécom SudParis, France*

Abstract

In recent years, wearable sensor-based gesture recognition is proliferating in the field of healthcare. It could be used to enable remote control of medical devices, contactless navigation of X-ray display and Magnetic Resonance Imaging (MRI), and largely enhance patients' daily living capabilities. However, even though a few commercial or prototype devices are available for wearable gesture recognition, none of them provides a combination of (1) fully open API for various healthcare application development, (2) appropriate form factor for comfortable daily wear, and (3) affordable cost for large scale adoption. In addition, the existing gesture recognition algorithms are mainly designed for discrete gestures. Accurate recognition of continuous gestures is still a significant challenge, which prevents the wide usage of existing wearable gesture recognition technology. In this paper, we present Ultigesture wristband, a hardware/software platform for gesture recognition and remote control. Due to its affordability, small size, and comfortable profile, Ultigesture wristband is an attractive option for mass consumption. Ultigesture wristband provides full open API access for third party research and application development. In addition, it employs a novel continuous gesture segmentation and recognition algorithm, which accurately and automatically separates hand movements into segments, and merges adjacent segments if needed, so that each gesture only exists in one segment. Experiments with human subjects show that the recognition accuracy is 99.4% when users perform gestures discretely, and 94.6% when users perform gestures continuously.

Keywords: Healthcare, open API, Ultigesture wristband, continuous gesture segmentation and recognition

1. Introduction

Healthcare is one important application scenario of gesture recognition technology. Lots of researchers and companies pay much attention to this area. According to the report published by MarketsandMarkets, the Healthcare application is expected to emerge as a significant market for gesture recognition technologies over the next five years [1]. In medicine, the ability of touch-free motion sensing input technology is particularly useful, where it can reduce the risk of contamination and is beneficial to both patients and their caregivers. For example, surgeons may benefit from touch-free gesture control, since it allows them to avoid interaction with non-sterile surfaces of the devices in use and hence to reduce the risk of infection. With the help of gesture control, the surgeons can manipulate the view of X-ray and MRI imagery, take notes of important information by writing in the air, and use hand gesture as commands to instruct robotic mechanism to perform complex surgical procedures. Wachs et al. [2] have developed a hand-gesture recognition system that enables doctors to manipulate digital images during medical procedures using hand gestures instead of touch screens or computer keyboards. In their system, a Canon VC-C4 camera and a Matrox Standard II video-capturing device are used for gesture tracking and recognition. The system has been tested during a neurosurgical brain biopsy at Washington Hospital Center.

*Corresponding author

Email addresses: hyzhao@cs.wm.edu (Hongyang Zhao), swang10@cs.wm.edu (Shuangquan Wang), gzhou@cs.wm.edu (Gang Zhou), daqing.zhang@telecom-sudparis.eu (Daqing Zhang)

Gesture recognition technology in healthcare can be mainly divided into two categories: computer-vision based gesture recognition and wearable sensor-based gesture recognition. The system developed by Wachs et al. [2] is an example of computer-vision based gesture recognition system. Though the system was tested in real-world scenarios, there still exists some disadvantages. It is expensive, needs color calibration before each use, and is highly influenced by lighting environment. Compared with computer-vision based recognition, wearable sensor-based gesture recognition technology is low cost, low power, requires only lightweight processing, no color calibration in advance, and is not interfered by lighting environment. Several wearable systems with gesture recognition technology have been proposed for healthcare application scenarios, e.g., upper limb gesture recognition for stroke patients [3] and for patients with chronic heart failure [4], glove-based sign language recognition for speech impaired patients, and for physical rehabilitation [5]. However, most wearable healthcare devices do not fit healthcare application scenarios well. There are mainly three problems in current wearable healthcare systems. (1) Not comfortable to wear. Many prototypes are too big that can not be used in reality. (2) No open Application Programming Interface (API). Most wearable healthcare prototypes do not open their APIs to public. Other developers cannot build applications based on their prototype. (3) Too expensive. Some wearable healthcare systems are quite expensive, e.g., a E4 healthcare monitoring wristband charges for $1690 with open API [6]. Additionally, most of gesture recognition prototypes can only recognize hand gestures one by one. Retrieving the meaningful gesture segments from continuous stream of sensor data is difficult for most gesture recognition prototypes [7].

To answer these problems, we address two research questions: (1) How does one design a hardware platform for gesture recognition and remote control, which is comfortable to wear, with open API, and at an affordable price? (2) How does one retrieve and recognize hand gestures from a continuous sequence of hand movements?

In this paper, we present Ultigesture (UG) wristband, a gesture recognition and remote control platform. The hardware platform integrates an accelerometer, gyroscope and compass sensor, providing powerful sensing capability for gesture recognition. We open our data sensing and gesture recognition APIs to the public, so that developers and researchers can build their applications or carry out research based on our wristband platform. Because we only integrate hardware components that are necessary for gesture recognition, our wristband is small, comfortable, and affordable. We propose is a novel, lightweight, and high-precision continuous gesture segmentation and recognition algorithm. First, we separate data from a sequence of hand movements into meaningful segments. Next, nearby segments are merged based on gesture continuity, gesture completeness and gesture symmetry metrics. The noise segments are then filtered out so that each segment contains one single gesture. Finally, we extract features from the acceleration and gyroscope data, and apply the Hidden Markov Model to recognize the gesture for each segment.

We summarize our contributions as follows:

1. We present a wristband-based platform for gesture control. We design our platform with the consideration of user experience and practicality. It is comfortable to wear, with open API, and at an affordable price.
2. We present a continuous gesture segmentation and recognition framework. We propose a lightweight, and effective data segmentation mechanism to segment potential hand gestures from a sequence of hand movements. Then, we apply Hidden Markov Model recognize hand gestures.
3. Our experiment results show that our system can recognize hand gesture with 99.4% accuracy when users perform gestures discretely. When users perform gestures continuously, our system can segment hand gesture with 98.8% accuracy and recognize hand gesture with 94.6% accuracy.

The remainder of this paper is organized as follows. First, we discuss the related work in Section 2. Then, we introduce the system framework in Section 3. The design of UG wristband is introduced in Section 4. We present our continuous gesture segmentation and recognition algorithm in Section 5. In Section 6, we evaluate the system performance. Finally, we draw our conclusion in Section 7.

2. Related Work

2.1. wristband-based platform

There have already been many wristband-based gesture recognition platforms. Some are developed by researchers in the university, while others are commercial products in the market. There are some common problems in current gesture recognition platforms. For example, most of current platforms do not open their API to the public [8][9][10][11], so they can not benefit other researchers and developers. Additionally, most platforms are too

Table 1: Comparison of wristband-based gesture recognition platform

Platform	Open API	Wearable	Affordable price
Dong et al. [8]	×	×	×
Junker et al. [9]	×	×	√
eWatch [10]	×	×	√
RisQ [11]	×	√	√
E-Gesture [12]	√	×	√
E4 [6]	√	√	×
Moto 360 (2nd Gen.) [14]	√	√	×
Ultigesture Wristband	√	√	√

large to wear on wrist [10] [12] [9]. Some researchers just attach one smart phone on their wrist, which is inconvenient for daily wearing [13]. In the market, several wearable devices provide open API and are comfortable to wear, such as smart watch [14] and E4 healthcare monitoring wristband [6]. However, these products are either not targeted at healthcare, or too expensive. Table 1 shows the comparison of wristband-based gesture recognition platforms. From the table, we find that the platforms developed by researchers [8][9][10][11][12] usually do not provide open API, and are awkward to wear. Products on the market [6][14] are quite expensive, e.g., the price of Motorola Moto 360 (2nd Gen.) is over $300 and the price of E4 wristband is $1690. The motion sensors inside these platforms usually provide a high frequency sampling rate, e.g., Motorola Moto 360 (2nd Gen.) uses InvenSense MPU-6050 motion sensor which provides up to 1 kHz sampling rate for accelerometer and 8 kHz for gyroscope [15]. However, due to operating system and power requirement, the sampling rate for smart watch is limited to a lower frequency, e.g., 50 Hz [14]. This greatly limits the gesture recognition and motion sensing study in healthcare. Therefore, we are motivated to develop a wristband-based platform for gesture recognition and control for healthcare, which provides open API access, comfortable to wear, and at an affordable price. The proposed platform in this paper is based on the previous work [16]. Our platform extends the previous work in several ways, including the firmware design of the UG wristband, the design of the UG API, the design of the Gesture Symmetry metric, and more experiments.

2.2. gesture recognition

Recently, smart wristband-based gesture recognition has been studied in mobile and pervasive computing. Various approaches dealing with the recognition of gestures or events have been presented. RisQ applies motion sensors on the wristband to recognize smoking gestures [11]. Xu et al. classify hand/finger gestures and written characters from smart watch motion sensor data [17]. Bite Counter utilizes a watch-like device with a gyroscope to detect and record when an individual takes a bite of food [18].

To recognize hand gestures, the first step is to extract potential gesture samples from a sequence of hand movements. A simple way to do this is to wear an external button on their fingers or hold it in their hand, and press this button to explicitly indicate the start and end of gestures [19][13]. In order to do this, users must wear an external button on their fingers or hold it in their hand. Unlike a wristband, wearing a button all day is burdensome and unnatural, limiting the usability of a hand gesture system. Another way to do this is to segment gestures automatically. The motion data are automatically partitioned into non-overlapping, meaningful segments, such that each segment contains one complete gesture.

Compared with button-enabled segmentation, automatic gesture segmentation provides a natural user experience. Park et al. apply a threshold-based method to detect short pauses at the start and end of gestures [12]. They assume that a gesture is triggered when the gyroscope reading is higher than a threshold, and this gesture ends when the gyroscope reading is lower than this threshold. The authors define eight discrete gestures. When users perform these eight defined gestures, the gyroscope readings are always big. However, there are still many gestures that contain some small gyroscope readings, in which case their system mistakenly divides these gestures into multiple segments. Parate et al. assume that the gestures begin and end at some rest positions [11]. They segment gestures by computing the spatio-temporal trajectory of the wrist using quaternion data and tracking rest positions. However, all the segmentation and recognition procedures in their system are implemented in smart phone. They do not demonstrate their system's feasibility and performance in resource-limited wearable devices. Other researchers propose some complex segmen-

tation methods, such as sequence analysis [9] and probability calculation [20]. However, these methods require huge computational effort, which can not be effectively and efficiently implemented in resource-limited wristbands.

3. System Overview

Our Ultigesture wristband is comprised of a wristband hardware platform and a continuous gesture segmentation and recognition framework.

The hardware platform contains a 9-axis motion sensor MPU-9250 for data sensing (including a accelerometer, a gyroscope, and a compass sensor), and a powerful Cortex-M4 processor for data processing and gesture recognition. We open our API to the public, so that other researchers and developers can build their own applications and systems upon our platform. The design philosophy of Ultigesture wristband is the open API, the comfort of wear, and affordability in price.

The framework of the continuous hand gesture segmentation and recognition is shown in Fig. 1. It contains three modules: Sensing, Data Segmentation, and Hand Gesture Recognition. In the Sensing module, accelerometer, gyroscope and compass sensor readings are collected from 9-axis Inertial Measurement Unit (IMU) in the wristband. The sampling rate is set to be 20Hz with a balanced consideration of recognition accuracy, computational cost, and power restriction in the wristband.

The Data Segmentation module segments potential gestures from a sequence of hand movements. To segment gestures from hand movements, we first apply a threshold-based method to detect the start and end of the sequence of hand movements. As found in previous work [11] [21], while performing a hand gesture, people start from a static position, and then end in another static position. Therefore, the gestures tend to lie between these static positions. We develop a novel gesture segmentation algorithm to detect these static positions, and thus separate the sequence of hand movements into multiple segments, where one gesture may lie in one segment or several adjacent segments. To avoid splitting a single gesture's data into multiple segments, three metrics are proposed as post-processing to merge the adjacent segments so that each gesture only lies in one segment. Finally, the Noise Segments Removal module is applied to remove segments with noise gestures.

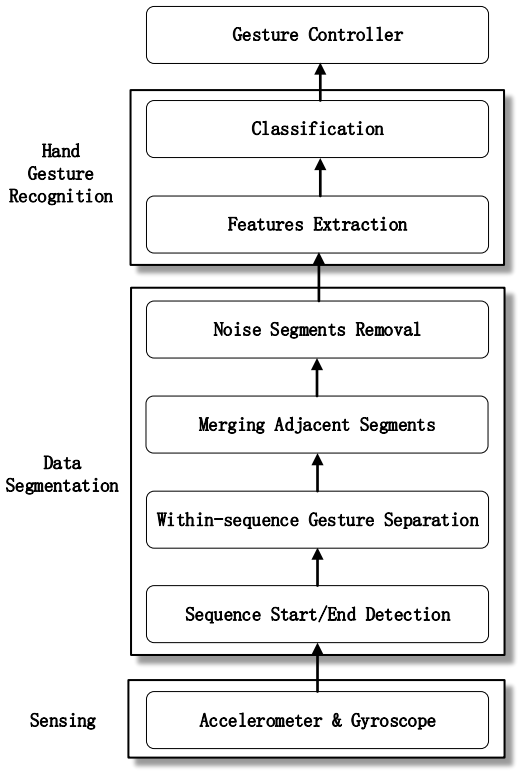

Figure 1: Continuous Gesture Segmentation and Recognition Framework

The Recognition module receives segments from Data Segmentation module and classifies each segment as one predefined gesture or noise gesture. We apply the Hidden Markov Model to classify gestures because it has shown high recognition accuracy [9]. The recognized gesture can be utilized as a command to control the medical instruments or healthcare-related devices, such as a medical computer screen, X-ray or MRI navigation.

4. Design of UG Wristband

We design our UG wristband as a platform for motion sensing study. It includes a series of hardware components, such as an accelerometer, gyroscope and compass sensor for data sensing, a powerful ARM Cortex-M4 microcon-

troller unit (MCU) for data processing, and a Bluetooth Low Energy (BLE) module for data transmission. We implement four firmware components to manage the UG wristband hardware components. The users can configure the UG wristband to work in three different modes by buttons. We open a series of BLE interfaces. Android developers can use our UG APIs to develop Android apps and manipulate multiple UG wristbands through BLE. Our smart wristband is available online [22]. Compared with existing data sensing and gesture recognition platforms, our wristband has three main advantages:

Open API. We open our data sensing APIs to Android developers. Developers can use our UG wristband as the data collector so that they can build their applications on top of our UG wristband. our APIs are designed as simple as possible so that it is easy for Android developers to use.

Comfortable to wear. We carefully design our UG wristband to make it comfortable to wear. Fig. 2 shows the appearance and the printed circuit board (PCB) design of our UG wristband. The size of the PCB is very small with 26mm length and 25mm width, which is much smaller than previous wristband platforms [10] [12]. The size of the wristband shell is: 50mm length, 30mm width, and 11.7mm thickness, which is very easy to carry.

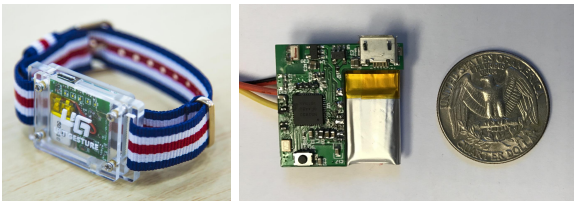

Figure 2: UG wristband

Affordable price. A practical, cheap, and open API platform is always strongly demanded by the researchers and developers in motion sensing. Some research groups use very large and expensive sensors ($2,000) [8], while others use generic smart watches (Motorola Moto 360 2nd Gen., over $300 [14]), or complex healthcare monitoring devices (E4 wristband: $1690 [6]). All these products are quite expensive for the motion sensing research and development. As our platform is designed for the motion sensing and gesture control, we only integrate necessary components into UG wristband, e.g., one 9-axis motion sensor MPU-9250 ($5), and one nRF52832 SoC ($5) that includes a Cortex-M4 processor and a BLE module. Therefore, our platform provides an affordable price for the customers to purchase and develop further.

4.1. Design of Hardware

Our smart wristband integrates an nRF52832 System on Chip (SoC) from Nordic Semiconductor as MCU. The nRF52832 SoC incorporates a powerful Cortex-M4 processor with 512kB flash and 64kB RAM, and a 2.4GHz transceiver that supports BLE protocol. The Cortex-M4 processor provides strong computation capability for complex data processing and the gesture recognition algorithm. The BLE module enables our wristband to run for a long period of time, and communicate with other medical devices that also support BLE. We carefully tune the hardware parameters for antenna so that the communication range of BLE can reach as far as 40 meters.

In terms of motion sensing, a 9-axis motion sensor MPU-9250 is embedded in our smart wristband. The MPU-9250 is a System in Package that combines two chips: the MPU-6500, which contains a 3-axis gyroscope, a 3-axis accelerometer; and the AK8963, a 3-axis digital compass sensor. Compared to other smart wristbands in the market that integrate different types of sensors, we only focus on the motion sensors that are widely used by researchers.

The capacity of the battery is a restriction for wearable devices. Our smart wristband is powered by a coin-size Li-Ion battery (3.7V, 75mAH). To account for the small capacity of the battery, we focus on the energy efficiency in our design. When turned on, our smart wristband consumes 10$\sim$20mAH, depending on how many computing functionalities are used. When turned off, our smart wristband only consumes 1uAH, which greatly prolongs the battery lifespan. The UG wristband can be charged through Micro-USB port. The charging time is around 1 hour. We also integrate 5 LEDs and 1 toggle button into UG wristband, which saves as user interfaces.

4.2. Design of Firmware

Firmware is used to control the function of various hardware components, which is embedded in flash memory of a UG wristband. We implement four firmware components to manage the UG wristband hardware components: Widgets Manager, Sensors Manager, Watchdog Manager and BLE Manager. The relationship between the firmware components and the hardware components are shown in Table 2.

Sensors Manager. The Sensor Manager is used to configure and communicate with the MPU-9250 IMU by Serial Peripheral Interface (SPI) Bus. Its functionality includes parameterizing the register addresses, initializing the sensor,

Table 2: Design of Firmware

Firmware Component	Controlled Hardware Component	Description
Sensors Manager	Accelerometer	Configure and read the accelerometer data
	Gyroscope	Configure and read the gyroscope data
	Magnetometer	Configure and read the magnetometer data
Watchdog Manager	Watchdog	Detect and recover from firmware malfunctions
Widgets Manager	Button	Detect if the button is pressed and how long it is pressed
	5 LEDs	Configure 5 LEDs to be on/off
	Battery	Measure the battery level
	Battery Charger	Detect if the battery is being charged or not
BLE Manager	BLE	Manage BLE stack and BLE communication

getting properly scaled accelerometer, gyroscope, and magnetometer data out, calibration and self-test of sensors. The sampling rates for the accelerometer, gyroscope, and compass sensor are up to 4 kHz, 1 kHz, and 8 Hz, which are configurable by the users. The measure range for these three sensors are set to be $\pm 4g$ (g is gravity), $\pm 2000^\circ/sec$, and $\pm 4800 \mu T$.

Watchdog Manager. Watchdog is an electronic timer that is used to detect and recover from firmware malfunctions. The Watchdog Manager manages the watchdog and regularly resets the watchdog timer to prevent it from elapsing. If, due to a hardware fault or program error, the Watchdog Manager fails to reset the watchdog, the watchdog timer elapses and generates a timeout signal. This timeout signal resets the firmware.

Widgets Manager. Widgets Manager is used to control the hardware widgets, including button, LEDs, battery, and battery charger. The functionality of it includes detecting if the button is pressed and how long it is pressed, configuring the 5 LEDs to be on or off, measuring the voltage of the battery, and detecting if the battery is charged or not.

BLE Manager. Every BLE device can work in one of two roles before and after BLE connection. Before connecting to another BLE device, a BLE device can work either in central role or peripheral role. The device in the central role scans for advertisement, and the device in the peripheral role makes the advertisement. After connecting to another BLE device, a BLE device can work either in server role or client role. The device in the server role stores and provides data to client, and the device in the client role requests data from server. BLE Manager is used to configure our UG wristband to work in peripheral role before BLE connection, and server role after BLE connection. It is in charge of a series of BLE operations, such as configuring BLE stack, making the advertisement, connecting to a central device, and transmitting data.

BLE provides an application-level protocol called Generic Attribute Profile (GATT) on top of a link-layer connection, which provides the general specification for data transmission over a BLE link. The GATT of UG wristband is shown in Fig. 3. There are three service: Sensor Manage (SM) Service, Widgets Manage (WM) Service, and Device Firmware Update (DFU) Service. The SM Service includes a collection of information related to sensors. It includes two characteristics: SM Service Transmit Characteristic and SM Service Receive Characteristic. The SM Service Transmit Characteristic is used to transmit sensor data to a remote BLE device. The SM Service Receive Characteristic is used to receive BLE command from a remote BLE device. A BLE command can be "sampling accelerometer at 10Hz". The WM Service includes a series of information related to widgets. It includes two characteristic: WM Service Transmit Characteristic and WM Service Receive Characteristic. The WM Service Transmit Characteristic is used to

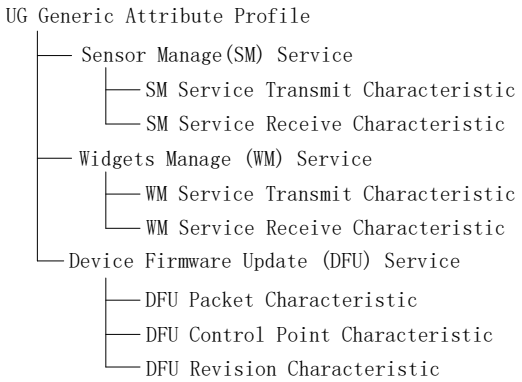

Figure 3: UG Generic Attribute Profile

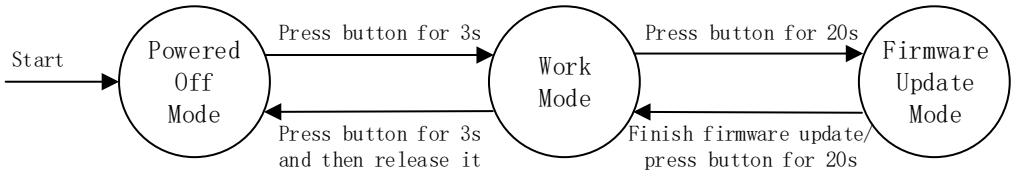

Figure 4: Mode Transition

transmit widgets' information to a remote BLE device, such as battery level. The WM Service Receive Characteristic is used to receive BLE command from a remote BLE device. A command can be "lighting the first LED". The Device Firmware Update (DFU) Service exposes necessary information to perform Device Firmware Update on the device. It includes three characteristics. The DFU Packet Characteristic is used to receive firmware. The DFU Control Point Characteristic is used to control the process of the firmware update. The DFU Revision Characteristic shows the revision of the firmware.

There are three modes UG wristband can work on: Powered Off Mode, Work Mode, and Firmware Update Mode. In Powered Off Mode, all the firmware components are turned off to save power. In Work Mode, All the firmware components are turned on. In this mode, the UG wristband can collect sensor data and send them to a remote BLE device. Firmware Update Mode is used to update firmware inside UG wristband, which fixes bugs or add new features. In Firmware Update Mode, a UG wristband only exposes Device Firmware Update Service to remote BLE devices and wait for firmware update. To update firmware in the UG wristband, a remote BLE device, such as a smart phone, needs to first write to the DFU Control Point Characteristic to enable firmware update, and then send a new firmware to DFU Packet Characteristic. After receiving the new complete firmware, the UG wristband resets the system, loads new firmware, and enters Work Mode.

We use the hardware button to switch between different modes. In Powered Off Mode, the user can press button for 3 seconds to enter Work Mode. In Work Mode, the user can press button for 3 seconds and release it to enter Powered Off Mode, or press button for 20 seconds to enter Firmware Update Mode. As we do not want the user to enter Firmware Update Mode by mistake, we set a long time, 20 seconds, to switch from Work Mode to Firmware Update Mode. In Firmware Update Mode, the user can press button for 20 seconds to exit Firmware Update Mode, or wait for the finish of firmware update. The Three modes transition is shown in Fig. 4.

4.3. Design of UG API

Generally the BLE programming on a smartphone works as a central device. It scans the peripheral devices and then connects to it. The connection is then used to access the peripheral device's characteristics or wait for notifications from peripheral device.

However, there occur some problems when an Android application tries to connects and communicates with multiple peripheral devices. Suppose an Android application wants to write to two UG wristbands at the same time, as shown in Fig. 5. The typical way goes that the Android application first needs to scan and create two Android BluetoothDevice objects: BluetoothDevice1 and BluetoothDevice2 for these two UG wristbands. Then it needs to request two Android BluetoothGatt objects: BluetoothGatt1 and BluetoothGatt2, to manage the BLE connection and communication with UG wristbands. Finally, it can write to these two UG wristbands at the same time. However, in Android, neither multiple BluetoothGatt objects are allowed to execute at the same time, nor one BluetoothGatt object is allowed to execute multiple operations at the same time. Otherwise, some unexpected results are come up. In this case, writing to two UG wristbands at the same time leads to the lost of write command, or even BLE disconnection, which is also observed by other researchers [23].

With the consideration of this issue, we provide a UGGattManager class in our APIs to coordinate multiple BLE operations, as show in Fig. 6. First, the Android application scans and creates two UGDevice objects: UGDevice1 and UGDevice2. Then, instead of creating two BluetoothGatt objects, two UGDevice objects request BLE write operations to the same UGGattManager object. The UGGattManager object queues BLE operations from the UGDevice objects

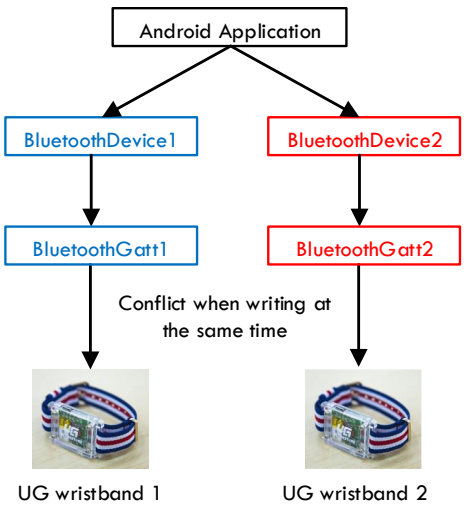

Figure 5: Typical BLE write operation

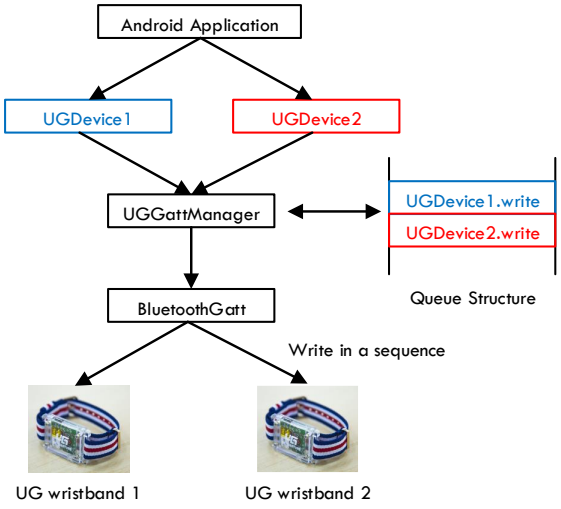

Figure 6: UG write operation

and requests one instance of BluetoothGatt. It sends BLE operations to the BluetoothGatt object from queue one by one and receives if the BLE operations are successfully executed by the BluetoothGatt object. The BLE operation is polled out of queue only when the previous BLE operation is successfully executed or time out. In this way, the BLE operations are executed in a sequence and write conflict is avoided.

To make sure all BLE operations are sent to the same UGGattManager object, we apply the singleton pattern for UGGattManager as shown in Fig. 7. UGGattManager class has its constructor as private and have a static instance of itself. It provides a static method to get its static instance to outside world. In this way, only one instance of UGGattManager class is created. Therefore, all BLE operations are sent to the only object of UGGattManager class.

```
public class UGGattManager {

   private static final UGGattManager mUGGattManager  = new
      UGGattManager ();

   private UGGattManager (){}

   public static UGGattManager getInstance(){
      return mUGGattManager ;

  ...
   }
}
```

Figure 7: Design of UGGattManager

We provide a series of open APIs to Android developers as shown in Table 3. There are mainly three classes in our open APIs: UGManager, UGDevice and UGGattManager. UGManager class is used to scan UG devices and return BLE connection status for each device. UGDevice class is used to manage the connected UGDevices. It provides a series of functions to let developers to get access to the data of the connected UG wristbands. UGGattManager manages BLE operations and communication protocol, which is invisible to developers.

5. Continuous Hand Gesture Recognition

The proposed continuous hand gesture recognition algorithm mainly contains three modules: Sensing, Data Segmentation, and Hand Gesture Recognition. The Sensing module collects the accelerometer and gyroscope sensor readings from IMU continuously, and outputs the sensor readings to the Data Segmentation module. The sampling rate of each sensor is set to be 20Hz with a balanced consideration of recognition accuracy, and computation and energy cost of the wearable device. The Data Segmentation module extracts individual gesture segments from a

Table 3: UG APIs

Class	UG APIs	Description
UGManager	UGManager(Context c, StatusChangeCallback cb)	Public constructor
	void startScan(ScanCallback cb)	Start a BLE scan for UG wristbands
	void stopScan()	Stop scanning
	Interface ScanCallback(){ void onScan (UGDevice device)}	Callback reporting a BLE device found during a device scan
	Interface StatusChangeCallback{ void onStatusChange (UGDevice device, int status)}	Callback triggered if the status of a UG wristband is changed
UGDevice	void connect()	Connect to a UG wristband
	void disconnect()	Disconnect from a UG wristband
	void startDataSensing(DataAvailableCallback cb, int rate)	Start to read sensor data from a UG wristband with certain rate
	void stopDataSensing()	Stop reading sensor data from a UG wristband
	void setLED(Byte[] ledMask)	Set the LEDs of a UG wristband to be on/off
	int getBatteryLevel()	Get the battery level of a UG wristband
	String getAddress()	Get the MAC address of a UG wristband
	Interface DataAvailableCallback{ void onDataAvailable (UGDevice device, float[] data)}	Callback reporting the sensor data received from a UG wristband
UGGattManager	N/A	Manage BLE operations and communication protocol

sequence of hand movements. The Hand Gesture Recognition module applies the HMM model to classify each individual gesture segment into one of the predefined gestures (Left, Right, Up, Down, Back&Forth, Clockwise, and Counterclockwise) or noise. The recognized gestures can be utilized to remotely control the medical instruments or healthcare related devices. In the following section, we first introduce the seven gestures defined in our system (Sec. 5.1). Then, the data segmentation module (Sec. 5.2) and the gesture recognition module (Sec. 5.3) are presented in more detail.

5.1. Gesture Definition

There has been substantial research on gesture recognition. Some work define gestures according to application scenarios, such as gestures in daily life [9], or repetitive motions in very specific activities [11], while others define gestures casually [12]. In this paper, we turn user's hand into a remote controller. We carefully design the hand gestures that best emulate a remote controller. Typically, a remote controller includes the following functions: left, right, up, down, select, play/pause, back. Therefore, we define the following seven gestures corresponding to these functions. At the beginning, the user extends his/her hand in front of his/her body. Then he/she moves towards a certain direction and moves back to the starting point again. We define the following gestures:

1. Left gesture: move left and then move back to the starting point
2. Right gesture: move right and then move back to the starting point
3. Up gesture: move up and then move back to the starting point
4. Down gesture: move down and then move back to the starting point
5. Back&Forth gesture: move to shoulder and then extend again to the starting point
6. Clockwise gesture: draw a clockwise circle
7. Counterclockwise gesture: draw an counterclockwise circle

These seven gestures are illustrated in Fig. 8. The defined hand gestures are very similar to the hand gestures defined by Wachs et al. [2]. Their gesture recognition system has been tested during a neurosurgical brain biopsy, which shows that these gestures are suitable as a remote controller for healthcare applications. To be noticed, each defined gesture ends at the starting point. Therefore, each gesture is independent from the others. Users can continuously perform the same or different gestures, which enables continuous control.

5.2. Data Segmentation

A simple way to segment hand gestures from a sequence of hand movements is to use a hand-controlled button to clearly indicate the starting point and the end point of each individual gesture. However, in order to do so, the user must wear an external button on their fingers or hold it in their hands, which is obtrusive and burdensome. Another way is to segment gestures automatically. The motion data are automatically partitioned into non-overlapping, meaningful segments, such that each segment contains one complete gesture. Automatic segmenting a continuous sensor data stream faces a few challenges. First, the segmentation should extract exactly one entire hand gesture, neither more nor less than needed. Otherwise, the extracted segments contain non-gesture noises, or miss useful gesture information, which leads to inaccurate classification. In addition, when a user performs multiple continuous gestures, the segmentation should not split a single gesture into multiple segments, or put multiple gestures into a single segment. To deal with these challenges, a continuous gesture data segmentation method is proposed, which contains three main steps: sequence start and end points detection, within-sequence gesture separation, and merging adjacent segments.

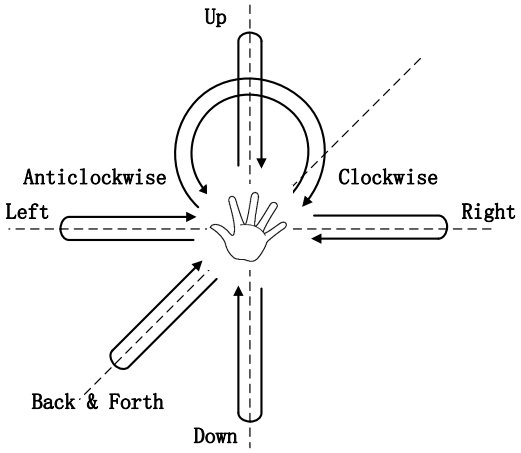

Figure 8: Seven defined gestures for remote control

5.2.1. Sequence start and end points detection

A lightweight threshold-based detection method is used to identify the start and end points of hand movements. To characterize a user's hand movement (HM), a detection metric is defined using the gyroscope sensor readings as

$$HM = \sqrt{Gyro_x^2 + Gyro_y^2 + Gyro_z^2}, \tag{1}$$

where $Gyro_x, Gyro_y, Gyro_z$ are the gyroscope readings of the X-axis, Y-axis, and Z-axis. When the user's hand is stationary, the HM is very close to zero. The faster a hand moves, the larger the HM is. When the HM is larger than a threshold, i.e. 50 degree/second, we regard it as the start point of hand movement. Once the HM is smaller than this threshold for a certain period of time, i.e. 400ms, we regard it as the end point of the hand movement. The time threshold is necessary as, in one single gesture, the HM may fall below this threshold occasionally, leading to unexpected splitting of this gesture [24][25]. Because the HM only keeps the magnitude of the vector sum of three axes and drops the direction information, this threshold-based detection method is independent of the device's orientation and therefore simplifies the gesture models.

Fig. 9 shows the gyroscope readings and the HM of one Left gesture and one Clockwise gesture. From Fig. 9(c), we see that the HM of the Left gesture falls below 50 degree/second at 1.6s. The Left gesture begins from moving left, then pauses, then moves right back to the original position. The low HM comes from the short pause in the Left gesture. The 400ms time frame prevents the Left gesture from being split into two separate hand movements.

Fig. 10 shows data processing for one continuous hand movement: raising hand horizontally→ performing Left gesture→ performing Back&Forth gesture→putting down hand. Raw gyroscope readings are shown in Fig. 10(a). The corresponding HM results for this hand movement sequence are shown in Fig. 10(b).

5.2.2. Within-sequence gesture separation

After detecting the start and end points of one sequence of hand movements, we partition this sequence of hand movements into non-overlapping, meaningful segments so that one hand gesture lies in one or several consecutive segments.

The hand gestures we defined start from and end in static positions that users feel comfortable with and choose according to their own free will. At static positions, the magnitude of hand rotation is relatively small. Therefore, the HM valley is a good indicator of the connecting point between two neighboring hand gestures. We employ a valley detection algorithm with a sliding window to detect valleys of the HM in the hand movement data. We

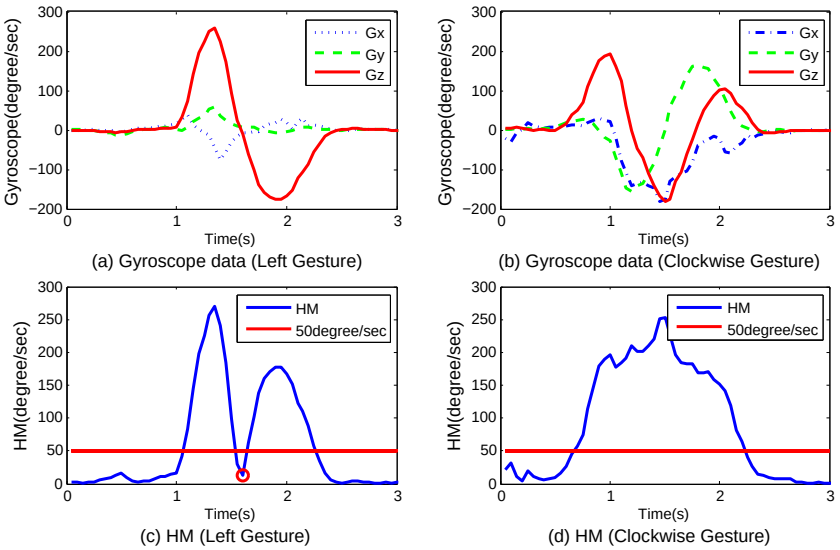

Figure 9: HM based start and end points detection

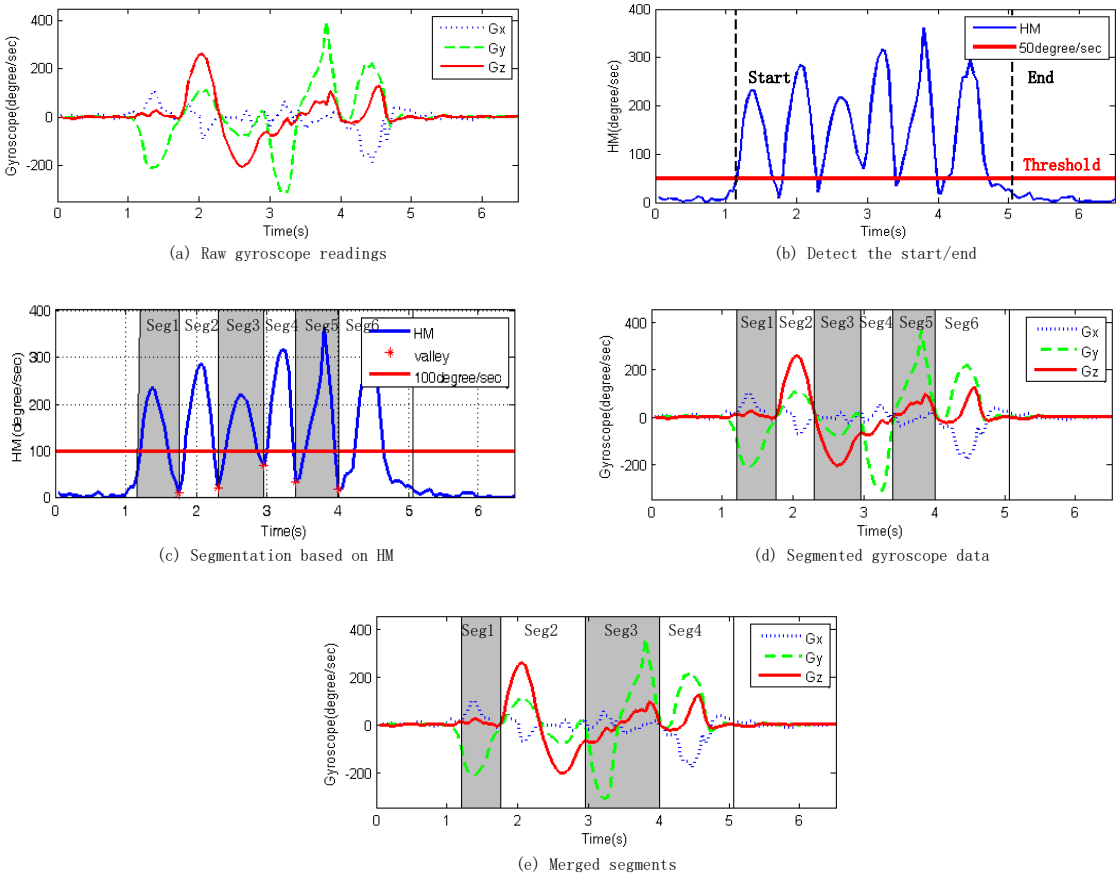

Figure 10: Data processing for one continuous hand movement

utilize valleys' positions as the segment points to partition the hand movement data into multiple and non-overlapping segments. Specifically, the sample at time $t(i)$ is a valley if it is smaller than all samples in the time window of $[t(i) - t_w/2, t(i) + t_w/2]$. Since the duration of one hand gesture is normally longer than 0.6 second, the window size t_w is set to be 0.6s.

With the window size threshold, the proposed algorithm is able to identify the HM valleys. However, sometimes there are a few false valleys which are not real switches of hand gestures. The reason is that the valley recognition algorithm only compares the HM magnitude in the time window, but does not take the absolute magnitude of the HM into consideration. A false HM valley may have large value, which indicates obvious and drastic rotation or movement. We collected the gyroscope data of a set of the continuous hand gestures which was conducted under supervision and the magnitude of HM valleys was carefully checked. The results show that, in general, the magnitude of the real HM valleys is less than 100 degree/second. Therefore, another condition, i.e. HM is less than 100 degree/second at the valleys, is added into the valley detection algorithm to eliminate the false valleys.

Fig. 10(c) shows the segmentation result based on the proposed valley detection algorithm. In total, five HM valleys are detected and six segments are generated. In this way, the raw gyroscope readings can be partitioned into six segments, as shown in Fig. 10(d). Each segment is one complete gesture or part of one complete gesture.

One question here is why we use gyroscope readings in the proposed segmentation method, rather than the accelerometer readings. The accelerometer is mainly suitable for detection of speed change. Comparatively, gyroscope is more powerful for detection of orientation change. For hand movement during conducting hand gestures, the orientation change is more significant than the speed change. Thus, gyroscope-based segmentation method is more robust and accurate than accelerometer-based segmentation method, and can provide higher segmentation accuracy [12].

5.2.3. Merging adjacent segments

For one continuous gyroscope readings stream, after segmentation, we get a series of partitioned segments. One gesture may lie in one segment or several continuous segments. In Fig. 10(d), segment 1 refers to "raise hand horizontally" movement, segment 2 and 3 belong to Left gesture, segment 4 and 5 are from Back&Forth gesture, and segment 6 is "put down hand" movement. The Left gesture and the Back&Forth gesture are both partitioned into two segments. To merge the adjacent segments so that one gesture only lies in one segment, we propose three measurement metrics to decide whether two neighboring segments should be merged: Gesture Continuity metric, Gesture Completeness metric, and Gesture Symmetry metric.

The **Gesture Continuity** metric measures the continuity of data in two neighboring segments. When two segments differ greatly in its signal shape at the connecting point, it is less likely that these two segments belong to the same single gesture. On the other hand, if two segments have similar slopes near the connecting point, these two segments may belong to one gesture. Based on this intuition, we compute the slopes near connecting points for each segments. If two slopes computed from two segments are similar, we say these two neighbor segments have similar shapes. Fig. 11 illustrates the computation of Gesture Continuity metric of a Right gesture:

For the sensor reading of each gyroscope axis, g_x, g_y and g_z (assume g_i), we do the following:

1. In g_i, we find the connecting point (t_1) between two segments $[t_0, t_1]$ and $[t_1, t_2]$, which is also a valley point in HM curve;
2. We extract the data points near connecting point (t_1) within one time window of 600ms, which is the same as the window size in valley detection algorithm. As the sampling rate is 20Hz, we pick 6 points before the connecting point (t_1) as t_a, t_b, t_c, t_d, t_e, t_f and 6 points after the connecting point (t_1) as t_g, t_h, t_i, t_j, t_k, t_l;
3. Twelve lines $\overline{t_a t_1}$, $\overline{t_b t_1}$,$\cdots$, $\overline{t_l t_1}$ are formed. For any 2 lines among the 12 lines, the angle between them is computed, and the maximum angle is defined as θ_{g_i};
4. We compute the weight w_{g_i} as the area size of the curve g_i in the time window $[t_0, t_2]$.

As there are three axes for gyroscope readings, we compute the three angles ($\theta_{g_i}, i \in \{x, y, z\}$) and three weights ($w_{g_i}, i \in \{x, y, z\}$) corresponding to the three axes. The Gesture Continuity (Con) at the connecting point t_1 is calculated as:

$$Con(t_1) = \frac{\sum(\mathrm{w}_{g_i} \cdot \theta_{g_i})}{\sum \mathrm{w}_{g_i}} \tag{2}$$

The higher the angle θ_{g_i} is, the bigger difference the signal shape is, and the less likely for the two segments to belong to the same gesture. In addition, a larger gyroscope reading of one axis indicates greater hand rotation around this axis. Accordingly, we add w_{g_i} as weights to three axes. Con is the weighted version of the angle θ_{g_i}. It ranges from 0 degree to 180 degree. Small Con stands for similar signal shapes for two neighbor segments. We merge two segments if the Gesture Continuity metric Con is small.

In Fig. 11, the Right gesture is partitioned into two segments $[t_0, t_1]$ and $[t_1, t_2]$. From the figure, we see that angle θ_{g_z} is quite small and weight w_{g_z} is very large. Therefore, the Con is small, and two segments $[t_0, t_1]$ and $[t_1, t_2]$ should be merged.

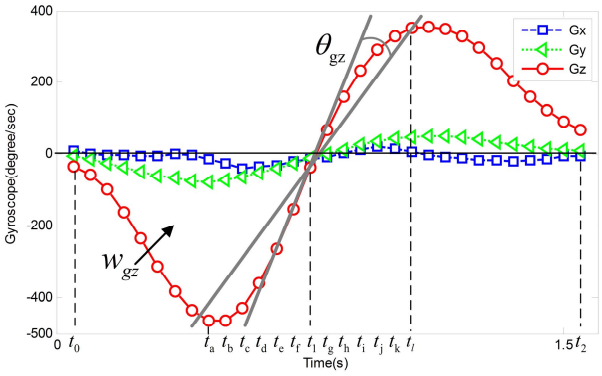

Figure 11: Computation of Gesture Continuity and Gesture Completeness metric

The **Gesture Completeness** metric measures the completeness of data in two neighboring segments if they belong to one complete gesture. To achieve continuous control, each gesture we chose to recognize starts from one user-chosen random position and ends with the same position. Even though the sensor readings vary during the procedure of a gesture, the sum of sensor readings should be close to zero for a complete gesture. Utilizing this gesture property, we calculate the Gesture Completeness (Com) metric as follows:

$$Com\,(t_1) = \frac{\left|\sum_{t_0}^{t_2} g_x\right| + \left|\sum_{t_0}^{t_2} g_y\right| + \left|\sum_{t_0}^{t_2} g_z\right|}{\sum_{t_0}^{t_2} |g_x| + \sum_{t_0}^{t_2} |g_y| + \sum_{t_0}^{t_2} |g_z|} \tag{3}$$

Here, g_x, g_y, g_z are sensor readings of each gyroscope axis, t_1 is the connecting point between two segments $[t_0, t_1]$ and $[t_1, t_2]$. Com ranges from 0 to 1. Small Com stands for that two neighboring segments belong to one gesture. We merge two segments if Com is low. In Fig. 11, we see that the sum of sensor readings for each axis is very close to zero. Therefore, Com is small and two segments $[t_0, t_1]$ and $[t_1, t_2]$ should be merged.

The **Gesture Symmetry** metric measures the symmetry of data in two neighboring segments. As all the gestures defined are symmetric, two segments that constitute a single gesture are symmetric with respect to the point connecting them. We merge two neighboring segments if they are symmetric to each other, and partition them if they are not.

We utilize Dynamic Time Warping (DTW) to calculate the Gesture Symmetry metric. DTW is an algorithm to find an optimal match between two temporal sequences. It specifies a distance metric, DTW Distance, to measure the similarity between two temporal sequences. If two temporal sequences differ a lot in shape, the DTW Distance metric is large. Otherwise, the DTW Distance metric is small. We apply Segment $[t_0, t_1]$ and the opposite of Segment $[t_1, t_2]$ as two temporal sequences, and compute DTW Distance between these two segments. As there are three axes for gyroscope readings, we compute three DTW Distances ($DTW_{g_i}, i \in \{x, y, z\}$) and three corresponding weights ($w_{g_i}, i \in \{x, y, z\}$), which are the same as Eq. 2. The Gesture Symmetry (Sym) for these two neighboring segments is calculated as:

$$Sym\,(t_1) = \frac{\sum\left(\mathrm{w}_{g_i} \cdot DTW_{g_i}\right)}{\sum \mathrm{w}_{g_i}} \tag{4}$$

Small Sym stands for that two neighboring segments are symmetric to each other with respect to the connecting point. We merge two segments if the Gesture Symmetry metric Sym is small.

We apply a threshold-based method to merge neighboring segments. If the Gesture Continuity metric, the Gesture Completeness metric, and the Gesture Symmetry metric of a connecting point are smaller than certain thresholds, we merge two segments that correlated to this connecting point. We apply a brute-force search to find the thresholds of three metrics that maximizes the segmentation accuracy.

For a given dataset of hand movement, we first compute segmenting points by valley detection algorithm and compute three metrics for each of them. After that, we get three arrays: one array with Gesture Continuity data, one

array with Gesture Completeness data, and one array with Gesture Symmetry data. We choose each possible combination of these three kinds of data as thresholds, and compute the segmentation accuracy for the whole dataset. The combination that achieves the highest segmentation accuracy is chosen as the thresholds for the Gesture Continuity metric, the Gesture Completeness metric, and the Gesture Symmetry metric.

Fig. 12 shows the Con, Com and Sym values for 100 gestures performed by one user continuously. In these 100 continuous gestures, there are 177 connecting points. Of all these connecting points, 99 of them separate two gestures, which are marked as blue stars; the other 78 connecting points are inside gestures, which are marked as red circles. The thresholds for Con, Com, and Sym are 27.29, 0.22 and 99635. The combination of these thresholds are shown as the red cubic areas in the figure. If Con, Com, and Sym for a connecting point are smaller than these three thresholds, we merge two segments correlated at this connecting point into one. From Fig. 12, we find that almost all red circles are distributed in the red cubic areas of the figure, while blue stars are quite evenly distributed in the figure. It shows that the proposed threshold-based method can effectively distinguish if the connecting points are inside one gesture or not. We merge two neighboring segments if their connecting point is inside one gesture.

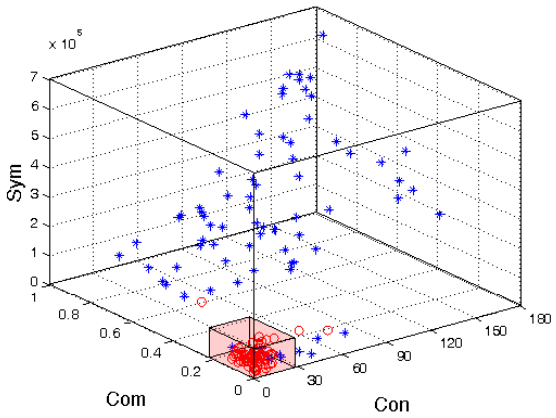

Figure 12: Con VS Com VS Sym

From Fig. 10(d) to Fig. 10(e), we find that segment 2 and 3 in Fig. 10(d) are merged into segment 2 in Fig. 10(e), which is a Left gesture. Segment 4 and 5 in Fig. 10(d) are merged into segment 3 in Fig. 10(e), which is a Back&Forth gesture. Each segment in Fig. 10(e) contains exactly one complete gesture.

5.2.4. Noise Segments Removal

We extract the following three features from each segment to classify if it is a noise segment:
(1) Duration of segment. Usually, the duration of one gesture is within a certain range. Among all the gesture data collected by us, no gesture lasts longer than 3 seconds, or shorter than 0.8 second. Therefore, if the duration of one segment is outside of these boundaries, this segment is filtered out as noise.
(2) HM of segment. The user is not supposed to perform the gestures too quickly. Therefore, HM, which measures the hand movement, is limited in a certain range. In our gesture dataset, we find that the max HM is 474 degree/second. Therefore, segments with the HM value above 474 degree/second are removed.
(3) Completeness of segment. The Gesture Completeness metric is used for segments merging as defined in Eq. (3). Here we use this metric again to remove noise segments. As each gesture defined by us starts from and ends in the same position, the Gesture Completeness metric (Com) for each gesture segment should be a small value. For all the gesture data collected, the Com values of more than 99% of gestures are smaller than 0.3. Therefore, if the Com of one segment is larger than 0.3, this segment is removed.

In Fig. 10(e), the Com value for Segments 1 to 4 are 0.98, 0.08, 0.04, 0.76, respectively. Therefore, Segment 1 and 4 are removed, Segment 2 and 3 are forwarded to the Hand Gesture Recognition module. Notably, Segment 1 and 4 are the "raise hand horizontally" movement and "put down hand" movement, which are not predefined gestures for our application.

5.3. Hand Gesture Recognition

According to the data segmentation results, we extract 6 representative features for model training and testing from the acceleration and gyroscope data of each segment: (1) raw acceleration data, (2) the first-derivative of acceleration data, (3) the integral of the acceleration data, (4) raw gyroscope data, (5) the first-derivative of gyroscope data, and (6) the integral of the gyroscope data. The first-derivative and the integral of the data are shown to be effective in improving recognition accuracy [12]. This is due to their ability to describe the main characteristics of the gesture: absolute trending (raw data), its relative change (first-derivative), and the cumulative effect (integral).

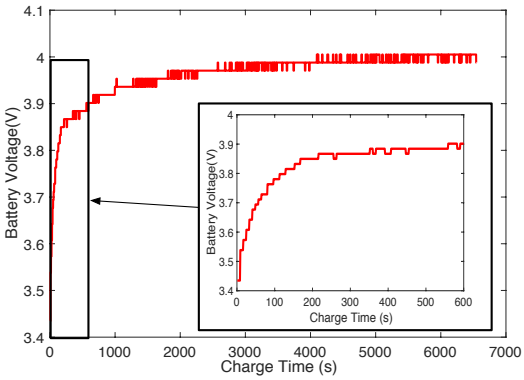

Figure 14: Battery charging curve

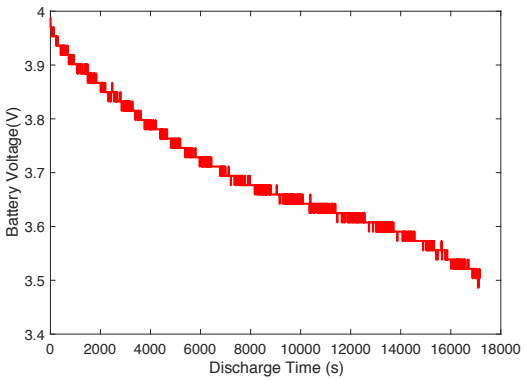

Figure 15: Battery discharging curve

We utilize the Hidden Markov Model (HMM) algorithm to train classifiers for online hand gesture recognition, as it has shown high accuracy in previous work [12][9][20]. For each gesture, an HMM model is trained based on a set of the same gesture data. We train the HMM models using the standard Baum-Welch re-estimation algorithm [26]. Each HMM model is configured with 4 states and 2 Gaussian components. To filter out non-gestural movements or undefined gestures, a noise HMM model is trained using the ergodic topology [20]. At runtime, each data segment is classified as one of the predefined gestures or noise. We use the Viterbi algorithm [27] to efficiently calculate the likelihood of the input data segment for each gesture model. Then, the gesture model with the highest likelihood is selected as the classified gesture. If the noise model is classified, we reject the gesture.

6. Performance Evaluation

In this section, we evaluate the performance of the hardware platform and the proposed algorithms. We first evaluate the packet loss rate and battery lifetime of the hardware platform in Section 6.1. Then, we introduce our dataset in Section 6.2. Next, we evaluate the threshold-based gesture segmentation algorithm in Section 6.3. Finally, we evaluate the accuracy of the gesture recognition algorithm when the users perform gestures separately and continuously in Section 6.4.

6.1. Hardware Platform

We measure the packet loss rate by sending packets from UG wristbands to a Pixel phone running Android 8.0. We configured 1 to 6 UG wristbands to send sensor data to the test device at different sampling rates, ranging from 1ms to 12 ms for 5 trials. We wrote an Android app to receive the sensor data from the UG wristbands and checked the packet loss rate. The transmission power of the UG wristbands was set to be 0dB, which is the default transmission power of the nRF52832 SoC. The distance between the UG wristbands and the Pixel phone was within 10cm. Fig. 13 shows the packet loss rate under different sampling intervals. Error bars in the figure represent the standard deviation. From the figure, we find that the packet loss rate for 1 to 3 UG wristbands reaches 0 when the sampling interval is over 4ms, and the packet loss rate for 4 to 6 UG wristbands reaches 0 when the sampling rate is over 12ms.

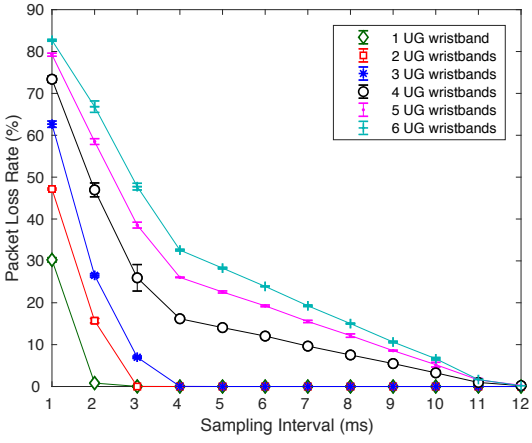

Figure 13: Packet loss rate between 1 to 6 UG wristbands and a Pixel phone.

Table 4: Characteristics of ten participants

Test Device	Human Subject No.	Gender	Age	Height(cm)	Weight(kg)
Moto 360	Human Subject 1	male	29	174	62
	Human Subject 2	male	30	171	68
	Human Subject 3	male	27	174	70
	Human Subject 4	male	28	181	81
	Human Subject 5	male	28	182	74
UG wristband	Human Subject 6	male	26	178	70
	Human Subject 7	male	28	173	73
	Human Subject 8	male	27	174	67
	Human Subject 9	male	27	180	66
	Human Subject 10	male	28	180	72

Table 5: Comparison of different metrics for Threshold-based Gesture Segmentation

Metrics	Precision	Recall	F-Measure	Accuracy
Con	90.9%	96.6%	93.6%	94.4%
Com	88.9%	97.4%	92.6%	93.9%
Sym	92.9%	82.6%	85.9%	90.3%
Con+Com+Sym	97.4%	97.4%	97.4%	97.7%

We used a 10-bit resolution analog-to-digital converter in the nRF52832 SoC to measure the voltage of the battery pin during charging and discharging. To charge a UG wristband, we used an AC/DC power adapter charger with 5.2V, 2.4A output. To discharge a UG wristband, we configured the UG wristband to sample sensor data at 20 Hz and sent the sensor data to a Pixel phone through BLE. The transmission power of UG wristband was set to be 0dB. Fig. 14 shows the battery charging curve. From the figure, we find that the voltage of the battery rises rapidly at the beginning and then gradually becomes stable. Fig. 15 shows the battery discharging curve. From the figure, we find that the voltage of the battery drops almost linearly. The lifetime of the battery is 4.47 hours. It takes only 2.8 minutes to reach 3.85V during charging, while it takes 4.0 hours to drop from 3.85V. This corresponds to 2.8 minutes of charging for 4 hours of use.

6.2. Data Collection

We collect the accelerometer and gyroscope data of seven hand gestures from 10 human subjects. The data collection experiment contains two independent steps: (1) each participant performs each gesture 10 times. During this experiment, the participants are asked to pause for 1 to 2 seconds between each gesture. (2) We randomly generate a sequence of 50 gestures. Participants are then asked to perform this sequence of gestures without pausing. We take video of each participant as they complete this task to serve as ground truth. We use a Motorola Moto 360 (2nd Gen.) smart watch to collect the gesture data from five human subjects and use a UG wristband to collect the gesture data from the other five human subjects. The characteristics of our participants are shown in Table 4. We call the gesture data collected by the Moto 360 as Moto dataset, and the gesture data collected by the UG wristband as UG dataset.

6.3. Gesture Segmentation Results

We evaluate the segmentation accuracy with leave-one-subject-out cross-validation on the Moto dataset. We choose the hand gesture data from four participants as the training dataset, and the remaining one as the test dataset. For the training dataset, we apply a brute-force search to compute the optimal thresholds for the Gesture Continuity metric, the Gesture Completeness metric, and the Gesture Symmetry metric. Then, we test these three thresholds on the test dataset. Precision, recall, F-measure and accuracy as considered as the evaluation metrics. Table 5 shows the segmentation performance under different metrics: the Gesture Continuity metric, the Gesture Completeness metric, the Gesture Symmetry metric, and all three features together. From the table, we find that the segmentation accuracy is 97.7% when using all three metrics together. When using only one metric, the Gesture Continuity metric performs

Table 6: Comparison between Machine learning Algorithms and Threshold-based Method for Gesture Segmentation

Algorithm	Precision	Recall	F-Measure	Accuracy
AdaBoost	97.2%	97.1%	97.1%	97.1%
Naive Bayes	95.9%	95.9%	95.9%	95.9%
SVM	96.3%	96.2%	96.2%	96.2%
J48	97.9%	97.9%	97.9%	97.9%
RandomForest	98.1%	98.1%	98.1%	98.1%

best. Its segmentation accuracy is 94.4%. The second one is the Gesture Completeness metric. The segmentation accuracy is 93.9%. The worst one is the Gesture Symmetry, the accuracy of which is 90.3%.

In addition to the proposed threshold-based method, another way to classify the segmenting points is to use machine learning algorithm. We apply WEKA machine-learning suite [29] to train five commonly used classifiers using all these three metrics as features. The classifiers include AdaBoost (run for 100 iterations), Naive Bayes, SVM (with polynomial kernels), J48 (equivalent to C4.5 [30]), and Random Forests (ten trees, four random features each). We run 5-fold cross-validation on this dataset for each classifiers. Precision, recall, F-measure and accuracy are used as the evaluation metrics. The classification results for these five algorithms are shown in Table 6. From the table, we find that RandomForest Performs the best. 98.1% of segments are correctly merged. The second one is J48. 97.9% segments are correctly merged. From Table 5, we find that the accuracy of the threshold-based method is 97.7%, which is very close to the RandomForest algorithm and performs better than AdaBoost, Naive Bayes, and SVM. As the threshold-based method is lightweight and has comparable performance with the machine learning algorithms, we apply the threshold-based method for gesture segmentation.

The Gesture Continuity metric is computed within a time window for a connecting point. A short time window size may result in the loss of good information, while a large time window size may incur noise. We examine the performance of the Gesture Continuity metric under different time window size, which is shown in Fig. 16. Three evaluation metrics are considered: precision, recall, and accuracy. When the time window size is smaller than 300ms or larger than 800ms, the segmentation accuracy goes down dramatically. When the time window size is 500ms or 600ms, the segmentation accuracy is highest: 94.4%. When the time window size is 500ms, the precision is larger than the recall. In this case, we will have more false negatives and less false positives. When the time window size is 600ms, the recall is larger than the precision. In this case, there are more false positives and less false negatives. In our system, false negative means that one segment is wrongly partitioned into two segments, while false positive means that two segments are wrongly merged into one. We prefer less false negatives over less false positives. This leads us to favor recall over precision. Therefore, we choose 600ms as the time window size to compute the Gesture Continuity metric.

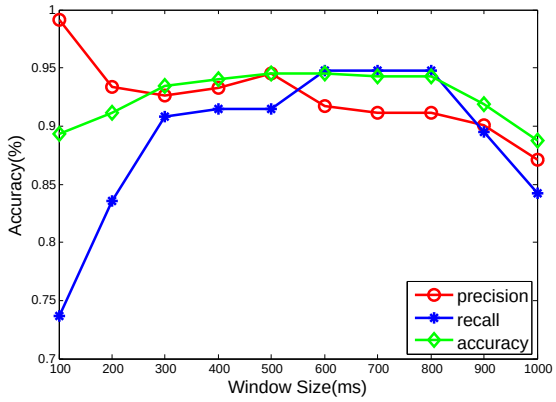

Figure 16: Precision, recall and accuracy under different window size

6.4. Gesture Recognition Results

We evaluate the performances of the proposed algorithms on both Moto dataset and UG dataset. For each dataset, we evaluate the proposed algorithms on the discrete gestures and the continuous gestures separately. First, we evaluate the gesture recognition accuracy with leave-one-subject-out cross-validation on each participant when performing gestures discretely. We use gesture samples from four participants to train the HMM models, and then apply these HMM models to classify the gesture samples from the remaining participant. Second, we use all the non-continuous

Table 7: Confusion matrix for gesture classification on the Moto dataset with just acceleration features

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	50	0	0	0	0	0	0
Right	0	48	0	0	0	0	2
Up	0	0	50	0	0	0	0
Down	0	0	0	40	0	6	4
Back&Forth	0	0	0	0	50	0	0
Clockwise	0	0	4	0	0	46	0
Counterclockwise	0	0	0	0	0	0	50

Table 8: Confusion matrix for gesture classification on the Moto dataset with just gyroscope features

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	42	6	4	0	0	0	0
Right	0	40	0	0	10	0	0
Up	0	0	50	0	0	0	0
Down	0	0	0	48	2	0	0
Back&Forth	0	0	0	0	50	0	0
Clockwise	0	0	0	0	0	50	0
Counterclockwise	0	0	0	0	0	0	50

hand gesture samples to train the HMM models, and evaluate the proposed algorithm on the continuous hand gesture samples.

Table 7 shows the confusion matrix for gesture classification on the Moto dataset with just acceleration features. The average accuracy is 95.4%. Table 8 shows the confusion matrix for gesture classification on the Moto dataset with just gyroscope features. The average accuracy is 93.7%. Table 9 shows the confusion matrix for gesture classification on the Moto dataset with both acceleration and gyroscope features. The average accuracy is 99.4%. Table 10 shows the confusion matrix for gesture classification on the UG dataset with just acceleration features. The average accuracy is 94.9%. Table 11 shows the confusion matrix for gesture classification on the UG dataset with just gyroscope features. The average accuracy is 97.7%. Table 12 shows the confusion matrix for gesture classification on the UG dataset with both acceleration and gyroscope features. The average accuracy is 98.3%. From these tables, we find that the proposed gesture classification algorithm performs well on both datasets. The gesture classification algorithm with both acceleration and gyroscope features has the best performance among these two datasets with accuracy over 98%. In the Moto dataset, the gesture classification algorithm with acceleration features is 1.7% more accurate than the gesture classification algorithm with gyroscope features. In the UG dataset, the gesture classification algorithm with gyroscope features is 2.8% more accurate than the gesture classification algorithm with acceleration features. We see there is only less than 3% accuracy difference between the two features. Therefore, we can reasonably conclude that there is no significant accuracy difference between the two features.

Fig. 17 demonstrates the segmentation accuracy after removing noise segments, recognition accuracy with both

Table 9: Confusion matrix for gesture classification on the Moto dataset with both acceleration and gyroscope features

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	50	0	0	0	0	0	0
Right	0	50	0	0	0	0	0
Up	0	0	48	0	2	0	0
Down	0	0	0	50	0	0	0
Back&Forth	0	0	0	0	50	0	0
Clockwise	0	0	0	0	0	50	0
Counterclockwise	0	0	0	0	0	0	50

Table 10: Confusion matrix for gesture classification on the UG dataset with just acceleration features

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	46	0	0	0	0	0	4
Right	0	40	0	0	0	0	10
Up	0	0	48	0	0	0	2
Down	0	0	0	50	0	0	0
Back&Forth	0	0	0	0	48	0	2
Clockwise	0	0	0	0	0	50	0
Counterclockwise	0	0	0	0	0	0	50

Table 11: Confusion matrix for gesture classification on the UG dataset with just gyroscope features

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	50	0	0	0	0	0	0
Right	0	50	0	0	0	0	0
Up	0	0	50	0	0	0	0
Down	0	0	0	50	0	0	0
Back&Forth	0	0	8	0	42	0	0
Clockwise	0	0	0	0	0	50	0
Counterclockwise	0	0	0	0	0	0	50

Table 12: Confusion matrix for gesture classification on the UG dataset with both acceleration and gyroscope features

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	50	0	0	0	0	0	0
Right	4	46	0	0	0	0	0
Up	0	0	50	0	0	0	0
Down	0	0	0	50	0	0	0
Back&Forth	0	0	2	0	48	0	0
Clockwise	0	0	0	0	0	50	0
Counterclockwise	0	0	0	0	0	0	50

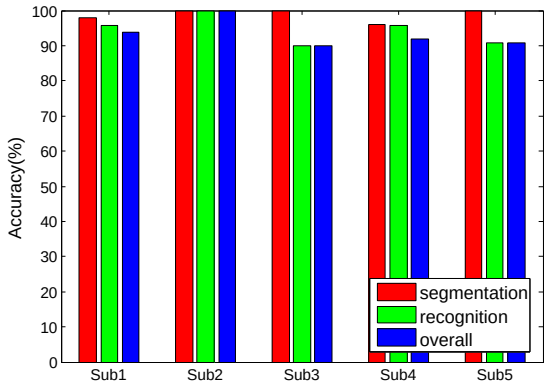

Figure 17: Segmentation and recognition accuracy of the continuous hand gesture recognition algorithm on the Moto dataset. Sub1 means human subject one

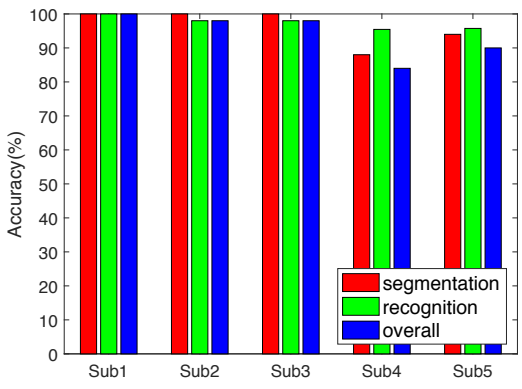

Figure 18: Segmentation and recognition accuracy of the continuous hand gesture recognition algorithm on the UG dataset. Sub1 means human subject one

Table 13: Confusion matrix for classification of continuous hand gestures on the Moto dataset

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	28	0	0	0	0	0	6
Right	0	40	0	0	0	0	0
Up	0	0	36	0	2	0	0
Down	0	0	0	29	0	0	0
Back&Forth	0	0	0	0	44	0	0
Clockwise	2	0	0	0	0	23	0
Counterclockwise	0	0	0	0	0	0	23

Table 14: Confusion matrix for classification of continuous hand gestures on the UG dataset

	Left	**Right**	**Up**	**Down**	**Back&Forth**	**Clockwise**	**Counterclockwise**
Left	34	0	0	0	0	1	0
Right	0	40	0	0	0	0	0
Up	0	0	39	0	0	0	0
Down	1	0	0	28	0	0	0
Back&Forth	0	0	0	0	39	0	2
Clockwise	0	0	0	0	0	25	0
Counterclockwise	2	0	0	0	0	0	23

acceleration and gyroscope features, and overall accuracy of the continuous gesture recognition algorithm on the Moto dataset. The overall accuracy is the product of the segmentation accuracy and recognition accuracy. The average segmentation accuracy is 98.8% (standard deviation: 1.8%), the average recognition accuracy is 95.7% (standard deviation: 4.1%), and the average overall accuracy is 94.6% (standard deviation: 4.0%). Fig. 18 demonstrates the segmentation accuracy, recognition accuracy, and overall accuracy of the continuous gesture recognition algorithm on the UG dataset. The average segmentation accuracy is 96.4% (standard deviation: 5.4%), the average recognition accuracy is 97.4% (standard deviation: 1.9%), and the average overall accuracy is 94.0% (standard deviation: 6.8%). Experimental results demonstrate that the proposed segmentation algorithm and recognition algorithm are very promising.

Table 13 shows the confusion matrix for classification of continuous hand gestures on the Moto dataset. From the table, we see that six Left gestures are mistakenly classified as Counterclockwise, two Up gestures are mistakenly classified as Back&Forth, and two Clockwise gestures are mistakenly classified as Left. Table 14 shows the confusion matrix for classification of continuous hand gestures on the UG dataset. From the table, we see that one Left gesture is mistakenly classified as Clockwise, one Down gesture is mistakenly classified as Left, two Back&Forth gestures are mistakenly classified as Counterclockwise, and two Counterclockwise gestures are mistakenly classified as Left. These misclassifications mainly come from the difference between training gesture samples and test gestures samples. We utilize gestures discretely performed by the users as the training set, and gestures continuously performed by the users as the test set. When users perform gestures continuously, sensor readings include lots of motion noises, which lead to the lower recognition accuracy.

Fig. 19 shows the classification accuracy for continuous hand gesture recognition on the Moto dataset with different features: HMM models with just acceleration features, HMM models with just gyroscope features, and HMM models with both acceleration and gyroscope features. The average accuracy for the HMM models with acceleration, gyroscope, and both acceleration and gyroscope features are: 86.0% (standard deviation: 16.6%), 82.0% (standard deviation: 7.4%), and 94.6% (standard deviation: 4.0%), respectively. Fig. 20 shows the classification accuracy for continuous hand gesture recognition on the UG dataset with different features. The average accuracy for the HMM models with acceleration, gyroscope, and both acceleration and gyroscope features are: 96.6% (standard deviation: 2.6%), 97.1% (standard deviation: 2.4%), and 97.4% (standard deviation: 1.9%), respectively. Statistically, HMM models with both the acceleration and gyroscope features are more accurate (the highest average accuracy) and more stable (the lowest standard deviation).

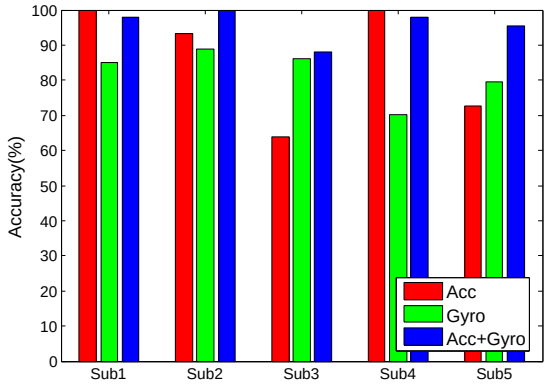

Figure 19: Classification accuracy for continuous hand gesture recognition with different HMM models on the Moto dataset. Sub1 means human subject one

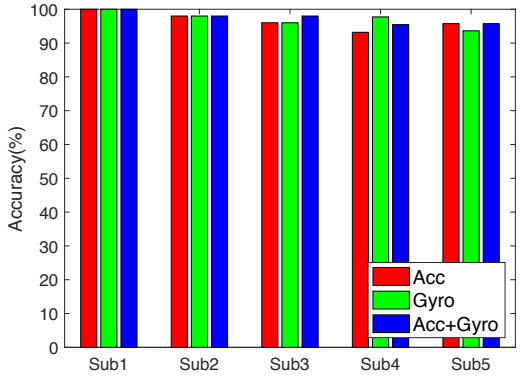

Figure 20: Classification accuracy for continuous hand gesture recognition with different HMM models on the UG dataset. Sub1 means human subject one

7. Conclusion

In this paper, we present Ultigesture, a wristband platform for gesture recognition for future remote control use in healthcare settings. We carefully design the hardware and the firmware of Ultigesture wristband. It is comfortable to wear, with open API, and at an affordable price. What's more, we propose a novel continuous gesture segmentation and recognition algorithm. For a sequence of hand movement, we separate data into meaningful segments, merge segments based on the Gesture Continuity metric, the Gesture Completeness metric, and the Gesture Symmetry metric, remove noise segments, and finally recognize hand gestures by HMM classification. Evaluation results show that the proposed algorithm can achieve over 94% recognition accuracy when users perform gestures continuously.

8. Acknowledgments

Special thanks to our participants in our user studies, Kyle for polishing the paper, and all the anonymous reviewers for their great reviews and suggestions to improve the quality of this paper. This work was supported by NSF CNS-1253506 (CAREER) and NSF CNS-1618300.

References

[1] Marketsandmarkets.com.
URL http://www.marketsandmarkets.com/Market-Reports/touchless-sensing-gesturing-market-369.html

[2] J. P. Wachs, H. I. Stern, Y. Edan, M. Gillam, J. Handler, C. Feied, M. Smith, A gesture-based tool for sterile browsing of radiology images, Journal of the American Medical Informatics Association 15 (3) (2008) 321–323.

[3] A. Tognetti, F. Lorussi, R. Bartalesi, S. Quaglini, M. Tesconi, G. Zupone, D. De Rossi, Wearable kinesthetic system for capturing and classifying upper limb gesture in post-stroke rehabilitation, Journal of NeuroEngineering and Rehabilitation 2 (1) (2005) 8.

[4] S. W. Davies, S. L. Jordan, D. P. Lipkin, Use of limb movement sensors as indicators of the level of everyday physical activity in chronic congestive heart failure, The American journal of cardiology 69 (19) (1992) 1581–1586.

[5] M. Milenkovic, E. Jovanov, J. Chapman, D. Raskovic, J. Price, An accelerometer-based physical rehabilitation system, in: Proceedings of IEEE SSST, IEEE, 2002, pp. 57–60.

[6] E. Wristband, E4 wristband (2017).
URL https://www.empatica.com/e4-wristband

[7] N. C. Krishnan, C. Juillard, D. Colbry, S. Panchanathan, Recognition of hand movements using wearable accelerometers, Journal of Ambient Intelligence and Smart Environments 1 (2) (2009) 143–155.

[8] Y. Dong, A. Hoover, E. Muth, A device for detecting and counting bites of food taken by a person during eating, in: Proceedings of IEEE BIBM, IEEE, 2009, pp. 265–268.

[9] H. Junker, O. Amft, P. Lukowicz, G. Tröster, Gesture spotting with body-worn inertial sensors to detect user activities, Pattern Recognition 41 (6) (2008) 2010–2024.

[10] U. Maurer, A. Rowe, A. Smailagic, D. P. Siewiorek, ewatch: a wearable sensor and notification platform, in: Proceedings of IEEE BSN, IEEE, 2006.

[11] A. Parate, M.-C. Chiu, C. Chadowitz, D. Ganesan, E. Kalogerakis, Risq: Recognizing smoking gestures with inertial sensors on a wristband, in: Proceedings of ACM MobiSys, ACM, 2014, pp. 149–161.
[12] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, J. Song, E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices, in: Proceedings of ACM SenSys, ACM, 2011, pp. 260–273.
[13] J. Liu, L. Zhong, J. Wickramasuriya, V. Vasudevan, uwave: Accelerometer-based personalized gesture recognition and its applications, Pervasive and Mobile Computing 5 (6) (2009) 657–675.
[14] Moto 360 2nd gen.
URL https://www.motorola.com/us /products/moto-360
[15] Invensense mpu6050 datasheet.
URL https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
[16] H. Zhao, S. Wang, G. Zhou, D. Zhang, Gesture-enabled remote control for healthcare, in: Proceedings of IEEE/ACM CHASE, IEEE/ACM, 2017.
[17] C. Xu, P. H. Pathak, P. Mohapatra, Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch, in: Proceedings of ACM HotMobile, ACM, 2015, pp. 9–14.
[18] Y. Dong, A. Hoover, J. Scisco, E. Muth, A new method for measuring meal intake in humans via automated wrist motion tracking, Applied psychophysiology and biofeedback 37 (3) (2012) 205–215.
[19] Wii controller.
URL http://wii.com/
[20] H.-K. Lee, J.-H. Kim, An hmm-based threshold model approach for gesture recognition, IEEE Transactions on pattern analysis and machine intelligence 21 (10) (1999) 961–973.
[21] C. Lee, Y. Xu, Online, interactive learning of gestures for human/robot interfaces, in: Proceedings of IEEE ICRA, Vol. 4, IEEE, 1996, pp. 2982–2987.
[22] UG smart wristband.
URL http://www.ultigesture.com/
[23] A. A. Levy, J. Hong, L. Riliskis, P. Levis, K. Winstein, Beetle: Flexible communication for bluetooth low energy, in: Proceedings of ACM MobiSys, ACM, 2016, pp. 111–122.
[24] W.-C. Bang, W. Chang, K.-H. Kang, E.-S. Choi, A. Potanin, D.-Y. Kim, Self-contained spatial input device for wearable computers, in: Proceedings of IEEE ISWC, IEEE Computer Society, 2003, p. 26.
[25] A. Y. Benbasat, J. A. Paradiso, An inertial measurement framework for gesture recognition and applications, in: International Gesture Workshop, Springer, 2001, pp. 9–20.
[26] L. E. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains, The annals of mathematical statistics 41 (1) (1970) 164–171.
[27] A. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE transactions on Information Theory 13 (2) (1967) 260–269.
[28] Invensense mpu9250 datasheet.
URL https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf
[29] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The weka data mining software: an update, Proceedings of ACM SIGKDD 11 (1) (2009) 10–18.
[30] J. R. Quinlan, C4. 5: programs for machine learning, Elsevier, 2014.