

2020

## Motion Sensors-Based Human Behavior Recognition And Analysis

Hongyang Zhao

*William & Mary - Arts & Sciences*, [cire.zhao@gmail.com](mailto:cire.zhao@gmail.com)

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Zhao, Hongyang, "Motion Sensors-Based Human Behavior Recognition And Analysis" (2020).  
*Dissertations, Theses, and Masters Projects*. Paper 1593091889.  
<http://dx.doi.org/10.21220/s2-5xdw-zx81>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

Motion Sensors-based Human Behavior Recognition and Analysis

Hongyang Zhao

Williamsburg, VA, USA

Master of Science, Zhejiang University, China, 2014

A Dissertation presented to the Graduate Faculty  
of The College of William & Mary in Candidacy for the Degree of  
Doctor of Philosophy

Department of Computer Science

College of William & Mary  
January 2020



# APPROVAL PAGE

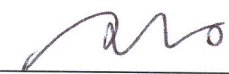
This Dissertation is submitted in partial fulfillment of  
the requirements for the degree of

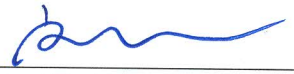
Doctor of Philosophy


  
\_\_\_\_\_  
Hongyang Zhao

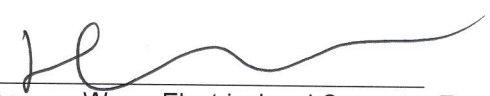
Approved by the Committee, January 2020

  
\_\_\_\_\_  
Committee Chair  
Professor Gang Zhou, Computer Science  
College of William & Mary

  
\_\_\_\_\_  
Professor Qun Li, Computer Science  
College of William & Mary

  
\_\_\_\_\_  
Assistant Professor Xu Liu, Computer Science  
College of William & Mary

  
\_\_\_\_\_  
Professor Weizhen Mao, Computer Science  
College of William & Mary

  
\_\_\_\_\_  
Associate Professor Honggang Wang, Electrical and Computer Engineering  
University of Massachusetts Dartmouth



## COMPLIANCE PAGE

Research approved by

The College of William & Mary Protection of Human Subjects Committee

Protocol number(s):

PHSC-2017-11-03-12491-gzhou

PHSC-2017-11-04-12489-gzhou

PHSC-2018-03-30-12905-gzhou

Date(s) of approval:

11/27/2017

11/27/2017

04/01/2018

## ABSTRACT

Human behavior recognition and analysis have been considered as a core technology that can facilitate a variety of applications. However, accurate detection and recognition of human behavior is still a big challenge that attracts a lot of research efforts. Among all the research works, motion sensors-based human behavior recognition is promising as it is low cost, low power, and easy to carry. In this dissertation, we use motion sensors to study human behaviors.

First, we present Ultigesture (UG) wristband, a hardware platform for detecting and analyzing human behavior. The hardware platform integrates an accelerometer, gyroscope, and compass sensor, providing a combination of (1) fully open Application Programming Interface (API) for various application development, (2) appropriate form factor for comfortable daily wear, and (3) affordable cost for large scale adoption.

Second, we study the hand gesture recognition problem when a user performs gestures continuously. We propose a novel continuous gesture recognition algorithm. It accurately and automatically separates hand movements into segments, and merges adjacent segments if needed, so that each gesture only exists in one segment. Then, we apply the Hidden Markov Model to classify each segment into one of predefined hand gestures. 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.

Third, we study the hand gesture recognition problem when a user is moving. We propose a novel mobility-aware hand gesture segmentation algorithm to detect and segment hand gestures. We also propose a Convolutional Neural Network to classify hand gestures with mobility noises. For the leave-one-subject-out cross-validation test, experiments with human subjects show that the proposed segmentation algorithm achieves 94.0% precision, and 91.2% recall when the user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when the user is standing, walking, and jogging, respectively.

Finally, we present a tennis ball speed estimation system, TennisEye, which uses a racket-mounted motion sensor to estimate ball speed. We divide the tennis shots into three categories: serve, groundstroke, and volley. For a serve, we propose a regression model to estimate the ball speed. In addition, we propose a physical model and a regression model for both groundstroke and volley shots. Under the leave-one-subject-out cross-validation test, evaluation results show that TennisEye is 10.8% more accurate than the state-of-the-art work.

## TABLE OF CONTENTS

<b>Acknowledgments</b>	v
<b>Dedication</b>	vi
<b>List of Tables</b>	vii
<b>List of Figures</b>	ix
<b>1 Introduction</b>	2
<b>1.1 Problem Statements</b> . . . . .	4
<b>1.2 Contributions</b> . . . . .	6
<b>1.3 Dissertation Organization</b> . . . . .	8
<b>2 Related Work</b>	9
<b>2.1 Wristband-based Gesture Recognition Platforms</b> . . . . .	9
<b>2.2 Continuous Gesture Recognition</b> . . . . .	10
<b>2.3 Gesture Recognition With Movement Noises</b> . . . . .	12
<b>2.4 Tennis Shots Analysis</b> . . . . .	12
<b>3 Ultigesture: A wristband-based platform for motion sensing</b>	15
<b>3.1 Introduction</b> . . . . .	15
<b>3.2 Design of UG Wristband</b> . . . . .	16
<b>3.2.1 Design of Hardware</b> . . . . .	17
<b>3.2.2 Design of Firmware</b> . . . . .	18
<b>3.2.3 Design of API</b> . . . . .	21
<b>3.3 Performance Evaluation</b> . . . . .	24

3.4 Conclusion	26
<b>4 Continuous Gesture Recognition for Remote Control</b>	<b>27</b>
4.1 Introduction	27
4.2 System Overview	29
4.3 Continuous Hand Gesture Recognition	31
4.3.1 Gesture Definition	31
4.3.2 Data Segmentation	33
4.3.2.1 Sequence start and end points detection	33
4.3.2.2 Within-sequence gesture separation	35
4.3.2.3 Merging adjacent segments	36
4.3.2.4 Noise Segments Removal	41
4.3.3 Hand Gesture Recognition	42
4.4 Performance Evaluation	43
4.4.1 Data Collection	43
4.4.2 Gesture Segmentation Results	44
4.4.3 Gesture Recognition Results	46
4.5 Conclusion	51
<b>5 MobiGesture: Mobility-aware Hand Gesture Recognition</b>	<b>53</b>
5.1 Introduction	53
5.2 Motivation	56
5.2.1 Gesture Definition	56
5.2.2 Challenge of Gesture Recognition under Mobility	57
5.2.3 Dataset	59
5.3 System Architecture	60
5.4 Mobility-aware Segmentation	61
5.4.1 Feature Extraction	62

5.4.2	Mobility Classification	63
5.4.3	Non-Moving Segmentation	64
5.4.4	Self-Correlation Analysis	65
5.4.5	Moving Segmentation	67
5.4.6	Performance	68
5.5	Deep Learning Classification	70
5.5.1	Data Scaling	70
5.5.2	Convolutional Neural Network	71
5.5.2.1	Input Layer	71
5.5.2.2	Convolutional Layer	72
5.5.2.3	Batch Normalization Layer	72
5.5.2.4	ReLU Layer	73
5.5.2.5	Max-pooling Layer	73
5.5.2.6	Dropout Layer	73
5.5.2.7	Fully-connected Layer	73
5.5.2.8	Softmax Layer	74
5.5.2.9	Classification Layer	74
5.5.3	Performance	75
5.6	Performance Evaluation	76
5.6.1	Accuracy	77
5.6.2	Time Delay	78
5.7	Conclusion	78
6	TennisEye: Tennis Ball Speed Estimation using a Racket-mounted Motion Sensor	79
6.1	Introduction	79
6.2	TennisEye Design	82
6.2.1	Overview of TennisEye	82

6.2.2	Sensor Deployment	84
6.2.3	Stroke Detection	85
6.2.4	Data Segmentation	86
6.2.5	Data Interpolation	87
6.2.6	Stroke Classification	88
6.2.7	Serve Speed Calculation	88
6.2.8	Groundstroke/Volley Speed Calculation	89
6.2.8.1	Physical Model	90
6.2.8.2	Regression Model	97
6.3	Performance Evaluation	97
6.3.1	Data Set	97
6.3.2	Ball Speed Calculation Accuracy	98
6.3.2.1	Leave-one-subject-out Evaluation	98
6.3.2.2	Self Evaluation	101
6.3.2.3	Cross Evaluation	102
6.3.2.4	Influence Factors on the Physical Model	103
6.3.3	Stroke Detection Accuracy	104
6.3.4	Stroke Classification Accuracy	104
6.3.5	TennisEye Performance	105
6.4	Discussion and Future Work	106
6.5	Conclusion	107
7	Conclusion and Future Work	108
	Bibliography	111
	Vita	122

## ACKNOWLEDGMENTS

This dissertation is written with the support and help of many individuals. I would like to thank all of them.

First and foremost, I would like to express my deepest appreciation to my advisor, Dr. Gang Zhou. Without his guidance in my research, encouragement in my life, and confidence in my abilities, this dissertation would not have been possible.

I would also like to thank the rest of my committee members, Dr. Qun Li, Dr. Xu Liu, Dr. Weizhen Mao, and Dr. Honggang Wang for serving on my dissertation committee, as well as for their insightful comments.

My sincere thanks also go to all the members of the LENS group past and present, Dr. Ge Peng, Dr. Xin Qi, Dr. David T. Nguyen, Dr. Qing Yang, Dr. Daniel Graham, Dr. Kyle Wallace, Shuangquan Wang, Yongsen Ma, Amanda Watson, Woosub Jung, George Simmons, Xiaoran Peng, Benjamin Power, Minglong Sun, and Dr. Yantao Li, for the stimulating discussions, constructive suggestions, generous assistance, and effective teamwork.

I also thank my friends, Du Shen, Xiao Liu, and so on for all the fun we have had in the past five years.

Last but not the least, I would like to thank my family. Thanks to my parents, whose love and care has made me who I am today.

This dissertation is dedicated to my beloved parents for their endless and selfless love and support.



## LIST OF TABLES

2.1 Comparison of wristband-based gesture recognition platform . . .	10
3.1 Design of Firmware . . . . .	18
3.2 UG APIs . . . . .	24
4.1 Characteristics of ten participants . . . . .	43
4.2 Comparison of different metrics for Threshold-based Gesture Seg- mentation . . . . .	44
4.3 Comparison between Machine learning Algorithms and Threshold- based Method for Gesture Segmentation . . . . .	45
4.4 Confusion matrix for gesture classification on the Moto dataset with just acceleration features . . . . .	47
4.5 Confusion matrix for gesture classification on the Moto dataset with just gyroscope features . . . . .	47
4.6 Confusion matrix for gesture classification on the Moto dataset with both acceleration and gyroscope features . . . . .	47
4.7 Confusion matrix for gesture classification on the UG dataset with just acceleration features . . . . .	48
4.8 Confusion matrix for gesture classification on the UG dataset with just gyroscope features . . . . .	48
4.9 Confusion matrix for gesture classification on the UG dataset with both acceleration and gyroscope features . . . . .	48
4.10 Confusion matrix for classification of continuous hand gestures on the Moto dataset . . . . .	50

4.11 Confusion matrix for classification of continuous hand gestures on the UG dataset . . . . .	50
5.1 Characteristics of five participants . . . . .	59
5.2 Comparison of Machine learning Algorithms for Mobility Classifi- cation . . . . .	64
5.3 Comparison of the Gesture Segmentation Performance . . . . .	68
5.4 Comparison of the Gesture Classification Performance . . . . .	76
5.5 Comparison of the Overall Performance . . . . .	77
5.6 Comparison of the Time consumption . . . . .	78
6.1 Eight impact types between a tennis ball and a racket . . . . .	93
6.2 Dataset collected with a UG sensor . . . . .	98
6.3 Dataset collected with both a UG and a Zepp sensor . . . . .	98
6.4 Comparison with State-of-the-art Ball Speed Models . . . . .	100
6.5 Comparison of Machine learning Algorithms for Stroke Classification	105

## LIST OF FIGURES

3.1 UG wristband . . . . .	16
3.2 UG Generic Attribute Profile . . . . .	20
3.3 Mode Transition . . . . .	21
3.4 Typical BLE write operation . . . . .	22
3.5 UG write operation . . . . .	22
3.6 Design of UGGattManager . . . . .	23
3.7 Packet loss rate between 1 to 6 UG wristbands and a Pixel phone. . . . .	24
3.8 Battery charging curve . . . . .	25
3.9 Battery discharging curve . . . . .	25
4.1 Continuous Gesture Segmentation and Recognition Framework . . . . .	30
4.2 Seven defined gestures for remote control . . . . .	32
4.3 <i>HM</i> based start and end points detection . . . . .	34
4.4 Data processing for one continuous hand movement . . . . .	35
4.5 Computation of Gesture Continuity and Gesture Completeness metric . . . . .	38
4.6 <i>Con VS Com VS Sym</i> . . . . .	41
4.7 Precision, recall and accuracy under different window size . . . . .	46
4.8 Segmentation and recognition accuracy of the continuous hand gesture recognition algorithm on the Moto dataset. Sub1 means human subject one . . . . .	49

4.9 Segmentation and recognition accuracy of the continuous hand gesture recognition algorithm on the UG dataset. Sub1 means human subject one . . . . .	49
4.10 Classification accuracy for continuous hand gesture recognition with different HMM models on the Moto dataset. Sub1 means human subject one . . . . .	51
4.11 Classification accuracy for continuous hand gesture recognition with different HMM models on the UG dataset. Sub1 means human subject one . . . . .	51
5.1 Ten number gestures for remote control . . . . .	57
5.2 Accelerometer readings of a Right gesture when a user is jogging	58
5.3 UG Wristband . . . . .	60
5.4 System Architecture . . . . .	60
5.5 Segmentation when a user is moving . . . . .	66
5.6 Segmentation precision, recall, and F-measure under different window sizes . . . . .	68
5.7 Segmentation precision, recall, and F-measure under different $SC$ thresholds . . . . .	68
5.8 F-measure of E-gesture under different thresholds . . . . .	69
5.9 The distribution of the gesture duration in our dataset . . . . .	69
5.10 Architecture and parameter settings of the 9-layer CNN . . . . .	72
6.1 Data Processing of TennisEye . . . . .	83
6.2 Sensor placement and coordinate system . . . . .	84
6.3 Birdview of the tennis court from a PlaySight camera . . . . .	84
6.4 Accelerometer readings in the X-axis when a player plays tennis .	85
6.5 Gyroscope readings of a serve . . . . .	87
6.6 Interpolation of the gyroscope readings in the Y-axis . . . . .	88

6.7 Physical impact process between a tennis racket and a tennis ball	91
6.8 Acceleration and gyroscope data of a forehand shot . . . . .	92
6.9 Distribution of COR values . . . . .	96
6.10 Performance of ball speed estimation models under leave-one-subject-out cross validation . . . . .	99
6.11 Performance of ball speed estimation models under self test . . .	101
6.12 Performance of ball speed estimation models under 5-fold cross validation . . . . .	102
6.13 Performance of the physical model under different incoming ball speeds . . . . .	103
6.14 Performance of the physical model under different racket speeds	103
6.15 Performance of stroke detection under different thresholds . . . .	104
6.16 Performance of TennisEye under leave-one-subject-out cross validation . . . . .	105
6.17 Cumulative distribution functions of TennisEye and Zepp . . . . .	105

## Motion Sensors-based Human Behavior Recognition and Analysis

# Chapter 1

## Introduction

In the field of ubiquitous computing, human behavior recognition is an important research topic and has been used in many human-centric services and applications, such as health monitoring [1], remote control [2], and personalized recommendation [3]. Typically, human behavior recognition can be divided into two categories: computer vision-based behavior recognition and wearable sensor-based behavior recognition. Computer vision-based behavior recognition requires a camera to capture the human behaviors, while wearable sensor-based behavior recognition requires a physical sensing device to be worn on human bodies. Compared with computer-vision based recognition, wearable sensor-based behavior recognition technology is lower cost, lower power, not influenced by lighting environment, and has no color calibration in advance. Therefore, we utilize wearable sensors to detect and recognize human behaviors in the present study.

There have already been many wearable sensor-based human behavior recognition platforms, such as eWatch [4], a wearable sensing and notification platform; E-Gesture, a hand gesture recognition platform [5]; and E4, a healthcare monitoring wristband [6]. There are three common problems in current behavior recognition platforms: (1) They are not comfortable to wear. Many platforms are too big to be used in daily life, e.g., eWatch [4]. (2) They do not provide open API. Most wearable platforms do not open their APIs to public, such as E-gesture [5]. Therefore, other developers cannot build applications based on their platforms. (3) Some platforms are very expensive, e.g., a E4

healthcare monitoring wristband charges \$1690 with open API [6]. Therefore, a wearable sensor-based behavior recognition platform that is comfortable to wear, provides open API access, and can be bought at an affordable price does not exist.

Hand gesture recognition is a trending topic in human behavior recognition research. Several wearable systems with gesture recognition technology have been proposed, e.g., upper limb gesture recognition for stroke patients [1] and for patients with chronic heart failure [7], glove-based sign language recognition [8], and wristband-based smoking gesture recognition [9]. Most gesture recognition prototypes assume that a user performs one hand gesture at a time and that the user is not moving. Instead, in many scenarios, a person tends to perform multiple hand gestures continuously or to perform hand gestures while moving. For example, a person tends to perform hand gestures continuously to remotely control a drone or smart TV. In addition, a person tends to use hand gestures to control a music app while jogging. In these scenarios, the traditional hand gesture recognition algorithms do not work. Therefore, it is essential to recognize hand gestures when a person performs multiple gestures continuously and when a person is moving.

In addition to hand gesture recognition, tennis ball speed estimation is another emerging topic in human behavior study. Aggressive tennis shots with high ball speed are the key factors in winning a tennis match. Today's tennis players are increasingly focused on improving ball speed. As a result, in recent tennis tournaments, records of tennis shot speeds are broken again and again. The traditional method for calculating the tennis ball speed uses multiple high-speed cameras and computer vision technology, such as Hawk-eye technology [10] [11] and PlaySight [12]. However, high-speed cameras are very expensive and hard to set up. Another way to calculate the tennis ball speed is to use motion sensors, which are lower in cost, lower in power usage, not influenced by lighting environment, and easier to set up. There are some commercial products on the market that assess the performance of the players and estimate the ball speed [13] [14] [15]. However, none of these commercial products open their al-



gorithms to the public. In addition, no previous publication has used motion sensors to calculate the tennis ball speed. Therefore, the smotion sensors-based tennis ball speed estimation problem remains to be explored.

## 1.1 Problem Statements

In this dissertation, we investigate how to exploit motion sensors to recognize and analyze human behaviors. Specifically, we work on the following four problems:

**(1) Developing a Wristband-based Platform for Behavior Recognition.** There have already been many wristband-based behavior recognition platforms. Some are developed by researchers at universities, while others are commercial products on the market. We find that the platforms developed by researchers usually do not provide open API, and are awkward to wear. Products on the market are quite expensive. For example, the price of Motorola Moto 360 (2nd Gen.) is over \$300 [16], and the price of E4 wristband is \$1690 [6]. Therefore, we are motivated to develop a wristband-based platform for human behavior recognition, which provides open API access, is comfortable to wear, and can be bought at an affordable price.

**(2) Recognizing Hand Gestures When a User Performs Multiple Gestures Continuously.** Most hand gesture recognition prototypes can only recognize hand gestures one by one. Accurate recognition of continuous gestures is difficult for most gesture recognition prototypes [17]. To recognize multiple continuous hand gestures, we first need to partition this sequence of hand movement into non-overlapping segments, such that each segment contains one complete gesture. Then, some classification algorithms can be applied to classify each segment into one of predefined gestures. Segmenting multiple continuous hand gestures 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, we propose a continuous gesture data segmentation and recognition algorithm. The proposed algorithm first segments a sequence of hand movements into several small segments. Then, it merges the adjacent segments so that each segment contains one complete hand gesture. Finally, we apply the Hidden Markov Model to classify the hand gestures.

**(3) Recognizing Hand Gestures When a Person is Moving.** Hand gesture is a promising mobile user interface as it can simplify the interaction with a smartphone when a user is moving. However, most gesture recognition prototypes do not take mobility into consideration. When a user is moving, gesture recognition is difficult. The first reason is that hand swinging motions during walking or jogging are mixed with hand gestures. It is hard to classify whether a hand movement comes from hand swinging motions or a hand gesture. In addition, when the user performs a hand gesture while moving, the hand movement is a combination of the hand gesture and the body movement. The mobility noises caused by the body movement reduce the accuracy of the hand gesture recognition. To deal with these two challenges, we propose a novel mobility-aware gesture segmentation algorithm to detect and segment hand gestures, and a Convolutional Neural Network (CNN) model to classify hand gestures with mobility noises.

**(4) Estimating Tennis Ball Speed Using a Racket-Mounted Sensor.** Tennis ball speed is an important metric in assessing the skill level of a tennis player. However, it is not easy to estimate the tennis ball speed based on a motion sensor. First, it is hard to build a model to accurately calculate the tennis ball speed, because it is influenced by a variety of factors: type of stroke, stroke strength, stroke angle, string tension, and ball impact position. Some of these cannot be measured by a motion sensor, making it hard to calculate the ball speed accurately. Second, it is hard to build a model that is accurate for all kinds of tennis players. Different players have different tennis skills and stroke patterns. For example, advanced players perform strokes consistently and correctly,

while beginning players perform strokes inconsistently and incorrectly. A model that is accurate for the advanced players may not be accurate for the beginning players and vice versa. To deal with these two problems, we investigate the physical impact process between a tennis racket and a tennis ball. Then, we propose two models to calculate the groundstroke speed: a physical model and a regression model. We apply the physical model to calculate the ball speed for advanced players, and the regression model to calculate the ball speed for beginning players.

## 1.2 Contributions

This dissertation proposes four contributions towards human behavior recognition and analysis. The overall contributions are as follows:

**Developing a Wristband-based Platform for Human Behavior Study.** We develop a UG wristband, a motion sensors-based platform for human behavior study. It integrates a Cortex-M4 processor, a Bluetooth Low Energy (BLE) module, a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis digital compass sensor. It is comfortable to wear, provides open API access, and can be bought at an affordable price. Specifically, we make two contributions:

- We carefully design the hardware and firmware of the UG wristband. The size of a UG wristband printed circuit board is similar to that of a quarter. Therefore, it is easy to carry. A UG wristband only integrate motion sensors that are widely used by researchers. Therefore, it is cheap. We implement four firmware components to manage the hardware components, and provide a series of interfaces to configure the UG wristband that (1) configure and read sensor readings, (2) manage the peripherals, (3) configure BLE modules, (4) update firmware.
- We provide a series of open APIs to Android developers. The developers can use our APIs to configure and access the data of the connected UG wristband through

BLE. An Android library is carefully designed so that one smartphone can connect to multiple UG wristbands without conflict.

**Recognizing Hand Gestures When a User Performs Multiple Gestures Continuously.** We propose a novel continuous gesture segmentation and recognition algorithm. Specifically, we make two contributions:

- 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 to classify hand gestures.
- Our experimental results show that our system can recognize hand gestures with 99.4% accuracy when users perform gestures discretely. When users perform gestures continuously, our system can segment hand gestures with 98.8% accuracy and recognize hand gesture with 94.6% accuracy.

**Recognizing Hand Gestures When a Person is Moving.** We propose a novel mobility-aware hand gesture segmentation algorithm to detect and segment hand gestures. Specifically, we make three contributions:

- We propose a novel mobility-aware gesture segmentation algorithm to detect and segment hand gestures. We first apply a machine learning algorithm to classify the current body movement into moving or non-moving. If the user is not moving, we apply a threshold-based segmentation algorithm to segment the hand gestures. If the user is moving, we propose a novel self-correlation metric to evaluate the self-correlation of the sensor readings. Then, we apply a moving segmentation algorithm to segment the hand gestures.
- We design a CNN model to classify the hand gestures with mobility noises. We apply a batch normalization layer, a dropout layer, a max-pooling layer, and L2 regularization to overcome overfitting and handle mobility noises.

- We integrate the gesture segmentation and classification algorithms into a system, MobiGesture. Our experimental results show that the proposed segmentation algorithm achieves 94.0% precision and 91.2% recall when the user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when the user is standing, walking, and jogging, respectively.

**Estimating Tennis Ball Speed Using a Racket-Mounted Sensor.** We explore the motion sensors-based tennis ball speed estimation problem. Specifically, we make three contributions:

- We propose a tennis ball speed calculation system, TennisEye. It is the first research publication to calculate the serve, groundstroke and volley speed of a tennis ball using a racket-mounted motion sensor.
- We propose two models to calculate the groundstroke speed: a physical model and a regression model. We apply the physical model to calculate the ball speed for advanced players, and the regression model to calculate the ball speed for beginning players.
- We evaluate the proposed system using the tennis shot data from players of different levels. Our experiment results show that TennisEye is 10.8% more accurate than the state-of-the-art work.

### 1.3 Dissertation Organization

The rest of this dissertation is structured as follows. In Chapter 2, we discuss related work. In Chapter 3, we present our motion sensors-based platform, UG wristband. In Chapter 4, we propose a continuous hand gesture recognition algorithm. In Chapter 5, we propose a novel mobility-aware hand gesture recognition algorithm. In Chapter 6, we present a tennis ball speed estimation algorithm. Finally, we conclude in Chapter 7.

## Chapter 2

# Related Work

This chapter reviews related work in wristband-based gesture recognition platforms and gesture recognition algorithms.

### 2.1 Wristband-based Gesture Recognition Platforms

There have already been many wristband-based gesture recognition platforms. Some are developed by researchers in universities, 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 [18][19][4][9], so they can not benefit other researchers and developers. Additionally, most platforms are too large to wear on wrist [4][5][19]. Some researchers just attach one smart phone on their wrist, which is inconvenient for daily wearing [20]. In the market, several wearable devices provide open API and are comfortable to wear, such as smart watch [16] and E4 healthcare monitoring wristband [6]. However, these products are very expensive. Table 2.1 shows the comparison of wristband-based gesture recognition platforms. From the table, we find that the platforms developed by researchers [18][19][4][9][5] usually do not provide open API, and are awkward to wear. Products on the market [6][16] 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 usu-

ally 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 [21]. However, due to operating system and power requirement, the sampling rate for smart watch is limited to a lower frequency, e.g., 50 Hz [16]. This greatly limits the research study in gesture recognition and motion sensing. Therefore, we are motivated to develop a wristband-based platform for gesture recognition and control, which provides open API access, is comfortable to wear, and can be bought at an affordable price.

**Table 2.1:** Comparison of wristband-based gesture recognition platform

Platform	Open API	Wearable	Affordable price
Dong et al. [18]	×	×	×
Junker et al. [19]	×	×	✓
eWatch [4]	×	×	✓
RisQ [9]	×	✓	✓
E-Gesture [5]	✓	×	✓
E4 [6]	✓	✓	×
Moto 360 (2nd Gen.) [16]	✓	✓	×
Ultigesture Wristband	✓	✓	✓

## 2.2 Continuous Gesture Recognition

Inertial sensors-based gesture recognition has been widely studied in mobile and pervasive computing. Various approaches dealing with the recognition of gestures or events have been presented. RisQ applies motion sensors on the wristband to recognize smoking gestures [9]. Bite Counter [22] utilizes a watch-like device with a gyroscope to detect and record when an individual takes a bite of food. Porzi et al. [23] propose a smart watch-based gesture recognition system for assisting people with visual impairments. Xu et al. classify hand/finger gestures and written characters from smart watch motion sensor data [24]. FingerPad [25], uTrack [26], and Finexus [27] use magnetic sensors to recognize finger gestures.

To recognize continuous 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 [28][20]. 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 [5]. 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 [9]. 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 segmentation methods, such as sequence analysis [19] and probability calculation [29]. However, these methods require huge computational effort, which can not be effectively and efficiently implemented in resource-limited wristbands.



## 2.3 Gesture Recognition With Movement Noises

As far as we know, two efforts have been put forth to study the gesture recognition problem when the user is moving. Park et al. [5] propose a gesture recognition system with a hand-worn sensor and a mobile device. To segment hand gestures, they design a threshold-based closed-loop collaborative segmentation algorithm. It automatically adjusts the threshold according to four mobility situations: RIDE, STAND, WALK, and RUN. To recognize hand gestures, they propose a Multi-situation HMM architecture. There are several limitations in their system. For the gesture segmentation, their threshold-based segmentation algorithm cannot effectively differentiate the predefined hand gestures from the hand swinging motions in our dataset. For the gesture recognition, they train a HMM model for each pair of hand gesture and mobility situation. In total, 32 HMM models are trained in their system. As the number of the hand gestures or the number of the mobility situations increases, their computational cost increases dramatically. Different from this work, we only train one CNN model, which consumes much less computational power and time. Additionally, evaluation results show that our CNN model performs better than Multi-situation HMM model under leave-one-subject-out cross-validation test. The second work comes from Murao et al. [30]. They propose a combined-activity recognition system. This system first classifies user activity into one of three categories: postures, behaviors, and gestures. Then DTW is applied to recognize hand gestures for the specific category. However, their system requires five sensors attached to the human body to recognize activity. Instead, we only use one sensor.

## 2.4 Tennis Shots Analysis

There are mainly two ways to analyze tennis shots. One way is to use computer vision technology. The other way is to use wearable motion sensors.

In computer vision technology, some researchers use one camera to study the tra-

jectory of tennis ball. For example, Yan et al. introduce a methodology to process low quality single camera videos and track a tennis ball with the aid of a modified particle filter [31]. Qazi et al. combine machine learning algorithms with computer vision techniques to predict ball trajectories [32]. Wang et al. employ a neural network approach to predict ball trajectories [33]. These one camera-based methods can only capture video data in two dimensions. The lack of the video data in the third dimension makes data not usable for high precision applications such as line calling or player performance analysis. For this reason, some researchers use multiple dimensional video data to study the tennis shots. Pingali et al. are pioneers to build a multiple camera real time ball tracking system based on six cameras [34]. So far, there have been several commercial products for three dimensional tennis ball tracking and line calling, such as Hawk-eye technology [10] [11] and PlaySight [12]. Hawk-eye technology achieves extremely precise results (mean error rating of 2.6 mm) as it employs 6 to 10 high-dimensional cameras to equip the court. This technology has been applied by many researchers [35] [36] [37]. In addition to Hawk-eye, PlaySight is another multi-camera based sports video-review and analytics system [12]. This system is equipped with six high-dimensional cameras, and uses advanced image processing and analytical algorithms to capture and log stroke type, ball trajectory, speed and spin, in-depth shot data, player movement and more. We collect the tennis data in a tennis court that is equipped with a PlaySight system. The computer vision-based technology requires that a tennis court is equipped with camera systems. However, most tennis courts are not equipped with such systems. Therefore, most players can not get access to these systems, which limits the popularity of this technology.

Recent trends show that inertial motion sensors are being used to analyze tennis shots. Some works focus on the stroke behaviors classification. They classify tennis stroke types into forehand, backhand, serve, volley, smash, top spin, and back spin [38] [39] [40] [41] [42]. A variety of machine learning algorithms have been applied, including SVM [38], longest common subsequence [40], Random Forest [41], CNN, and

Bidirectional Long Short-Term Memory Networks (BLSTM) [42]. In addition to stroke classification, some researchers assess the performance of tennis strokes and provide recommendations to improve tennis skills [43][44][45]. Srivastava et al. study the consistency of tennis shots [43]. The authors provide recommendations on wrist rotation based on shots performed by professional players. TennisMaster used Hidden Markov Model to segment a tennis serve into 8 phases [44]. By studying the power, gesture, and rhythm for each phase, the authors build a regression model which outputs the score of a serve. Sharma et al. segment a tennis serve into various key points, including start, trophy pose, cocking position, impact, and finish [45]. The authors constitute serve phases including backswing, pronation, and follow-through, by using the motion between these key points. By comparing the serve phases from a user and that from professionals, the system provide the user with corrective feedback and insights into their playing styles. Different from stroke behaviors classification and performance assessment, we investigate the interaction between the racket and the ball. To our knowledge, none of the previous works use motion sensors to estimate tennis ball speed.

In addition to these research works, there are also some commercial products on the market that assess the performance of the players, such as Zepp [13], Usense [14], Babolat [15]. These products either integrate the motion sensors inside the racket [15], or require users to attach the motion sensors onto the racket [13] [14]. They analyze the tennis data and compute the key performance metrics for each swing, such as stroke type, ball speed, ball spin, and sweet spot. However, none of these commercial products opens their algorithms to the public.

## Chapter 3

# Ultigesture: A wristband-based platform for motion sensing

### 3.1 Introduction

There are mainly three problems in current wearable 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 prototypes do not open their APIs to public. Other developers cannot build applications based on their prototype. (3) Too expensive. Some wearable 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 [17].

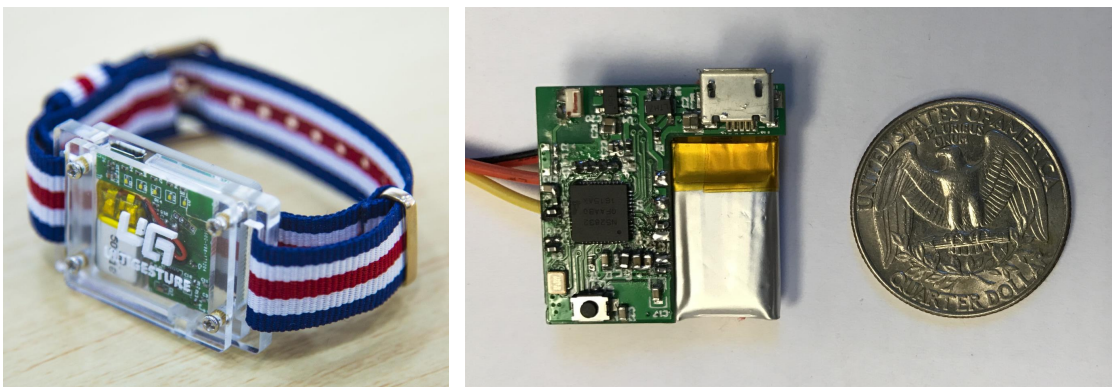
In this chapter, we present Ultigesture (UG) wristband, a motion sensing platform. The hardware platform integrates an accelerometer, gyroscope and compass sensor, providing powerful sensing capability for gesture recognition. We open our data sensing and gesture recognition APIs to the public, so that 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.

## 3.2 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 microcontroller 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 [46]. 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.



**Figure 3.1:** UG wristband

**Comfortable to wear.** We carefully design our UG wristband to make it comfortable to wear. Fig. 3.1 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 [4] [5]. The size of the wristband shell is: 50mm length, 30mm width, and 11.7mm thickness, which is very easy to carry.

**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 [18]), while others use generic smart watches (Motorola Moto 360 2nd Gen., over \$300 [16]), or complex 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.

### 3.2.1 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 10~20mAH, 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.

### 3.2.2 Design of Firmware

**Table 3.1:** 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

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 [3.1](#).

**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, 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  $\pm 4800\mu T$ .

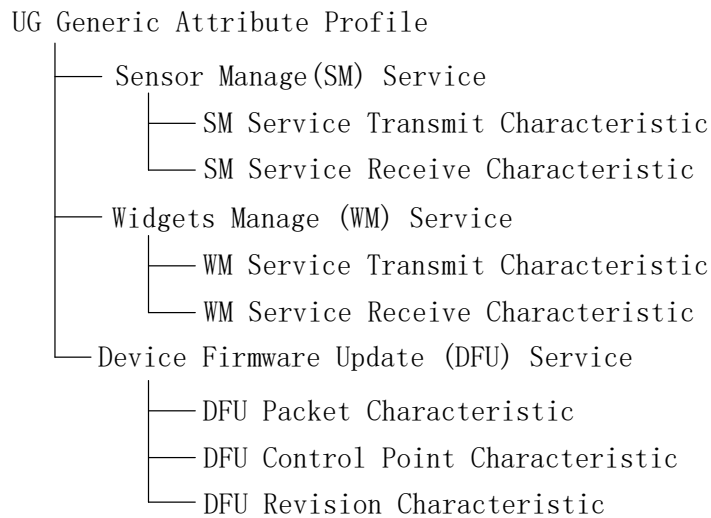
**Watchdog Manager.** Watchdog is an electronic timer that is used to detect and recover from firmware malfunctions. The Watchdog Manager manages the watchdog and regularly resets the watchdog timer to prevent it from elapsing. If, due to a hardware fault or program error, the Watchdog Manager fails 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. 3.2. 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



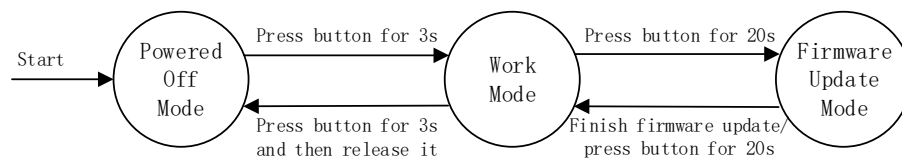


**Figure 3.2:** UG Generic Attribute Profile

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 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.



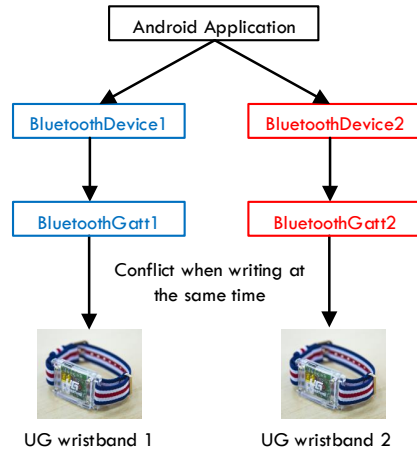
**Figure 3.3:** Mode Transition

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. [3.3](#).

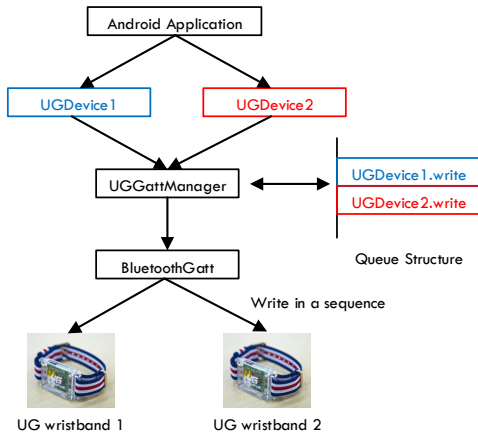
### 3.2.3 Design of API

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.



**Figure 3.4:** Typical BLE write operation



**Figure 3.5:** UG write operation

However, there occur some problems when an Android application tries to connect and communicate with multiple peripheral devices. Suppose an Android application wants to write to two UG wristbands at the same time, as shown in Fig. 3.4. 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 [47].

With the consideration of this issue, we provide a UGGattManager class in our APIs to coordinate multiple BLE operations, as show in Fig. 3.5. 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 oper-

ations to the same UGGattManager object. The UGGattManager object queues BLE operations from the UGDevice objects 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.

```
public class UGGattManager {  
  
    private static final UGGattManager mUGGattManager = new  
        UGGattManager ();  
  
    private UGGattManager (){}  
  
    public static UGGattManager getInstance(){  
        return mUGGattManager ;  
  
        ...  
    }  
}
```

**Figure 3.6:** Design of UGGattManager

To make sure all BLE operations are sent to the same UGGattManager object, we apply the singleton pattern for UGGattManager as shown in Fig. 3.6. 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.

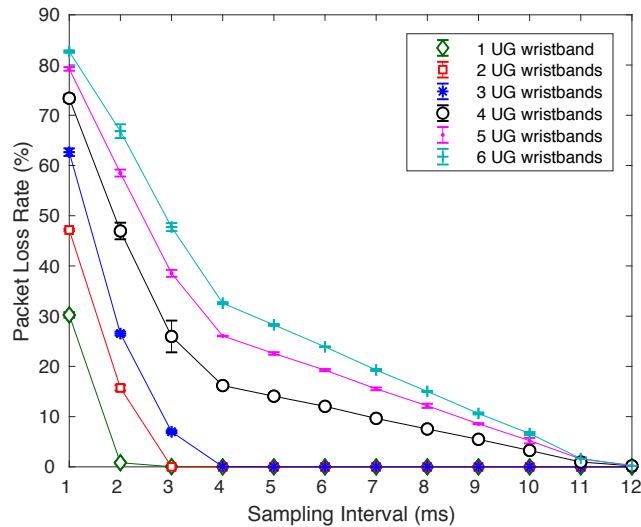
We provide a series of open APIs to Android developers as shown in Table 3.2. There are mainly three classes in our open APIs: UGManager, UGDevice and UGGattManager. UGManager class is used to scan UG devices and return BLE connection status for each device. UGDevice class is used to manage the connected 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 proto-

**Table 3.2: UG APIs**

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

col, which is invisible to developers.

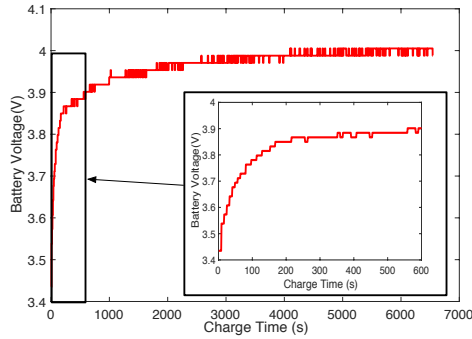
### 3.3 Performance Evaluation



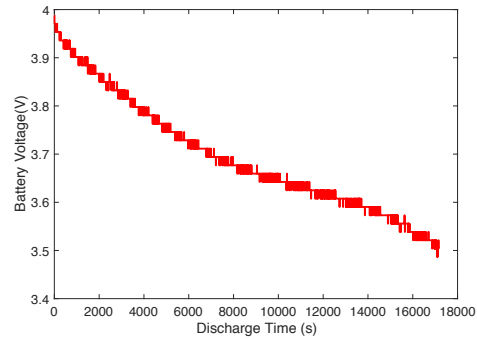
**Figure 3.7:** Packet loss rate between 1 to 6 UG wristbands and a Pixel phone.

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. 3.7 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 3.8:** Battery charging curve



**Figure 3.9:** Battery discharging curve

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. 3.8 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. 3.9 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.

### **3.4 Conclusion**

In this chapter, we present Ultigesture, a wristband platform for gesture recognition for future remote control use. We carefully design the hardware and the firmware of Ultigesture wristband. It is comfortable to wear, with open API, and at an affordable price.

## Chapter 4

# Continuous Gesture Recognition for Remote Control

### 4.1 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 [48]. 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. [49] 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.

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. [49] 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 [1] and for patients with chronic heart failure [7], glove-based sign language recognition for speech impaired patients, and for physical rehabilitation [50]. However, 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

To answer these problems, we address one research questions: How does one retrieve and recognize hand gestures from a continuous sequence of hand movements?

We propose 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 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.

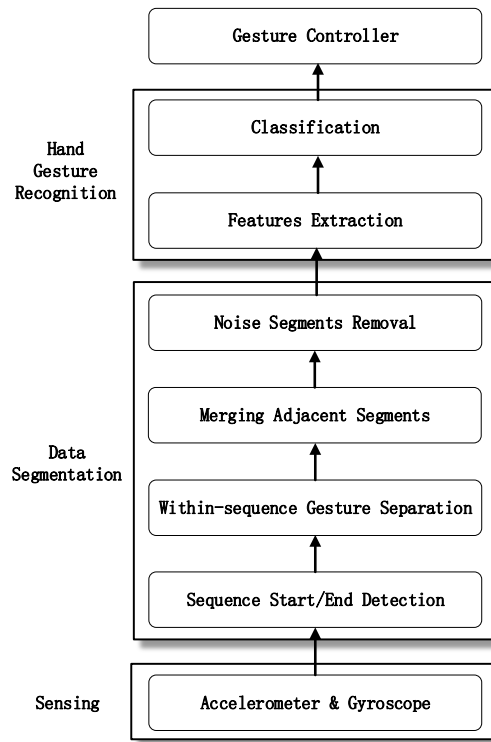
2. Our experiment results show that our system can recognize hand gesture with 99.4% accuracy when users perform gestures discretely. When users perform gestures continuously, our system can segment hand gesture with 98.8% accuracy and recognize hand gesture with 94.6% accuracy.

The remainder of this chapter is organized as follows. First, we introduce the system framework in Section 4.2. The design of UG wristband is introduced in Section 3.2. We present our continuous gesture segmentation and recognition algorithm in Section 4.3. In Section 6.3, we evaluate the system performance. Finally, we draw our conclusion in Section 6.5.

## 4.2 System Overview

The framework of the continuous hand gesture segmentation and recognition is shown in Fig. 4.1. 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 [9] [51], 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



**Figure 4.1:** Continuous Gesture Segmentation and Recognition Framework

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.

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 [19]. 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.

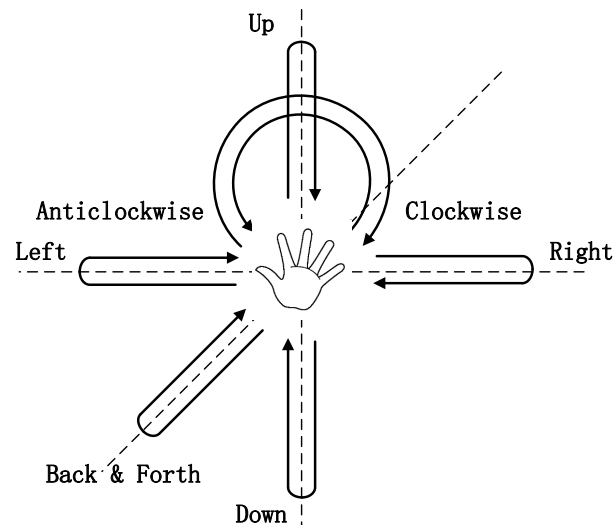
## 4.3 Continuous Hand Gesture Recognition

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 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. 5.2.1). Then, the data segmentation module (Sec. 6.2.4) and the gesture recognition module (Sec. 4.3.3) are presented in more detail.

### 4.3.1 Gesture Definition

There has been substantial research on gesture recognition. Some work define gestures according to application scenarios, such as gestures in daily life [19], or repetitive motions in very specific activities [9], while others define gestures casually [5]. In this project, 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



**Figure 4.2:** Seven defined gestures for remote control

These seven gestures are illustrated in Fig. 4.2. The defined hand gestures are very similar to the hand gestures defined by Wachs et al. [49]. 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.

### 4.3.2 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.

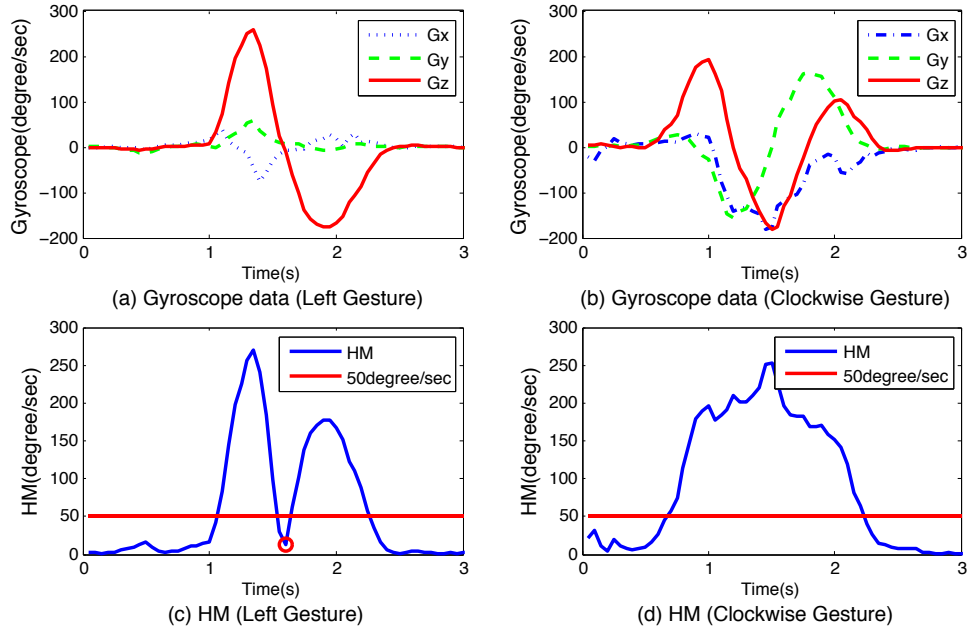
#### 4.3.2.1 Sequence start and end points detection

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

$$HM = \sqrt{Gyro_x^2 + Gyro_y^2 + Gyro_z^2}, \quad (4.1)$$

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

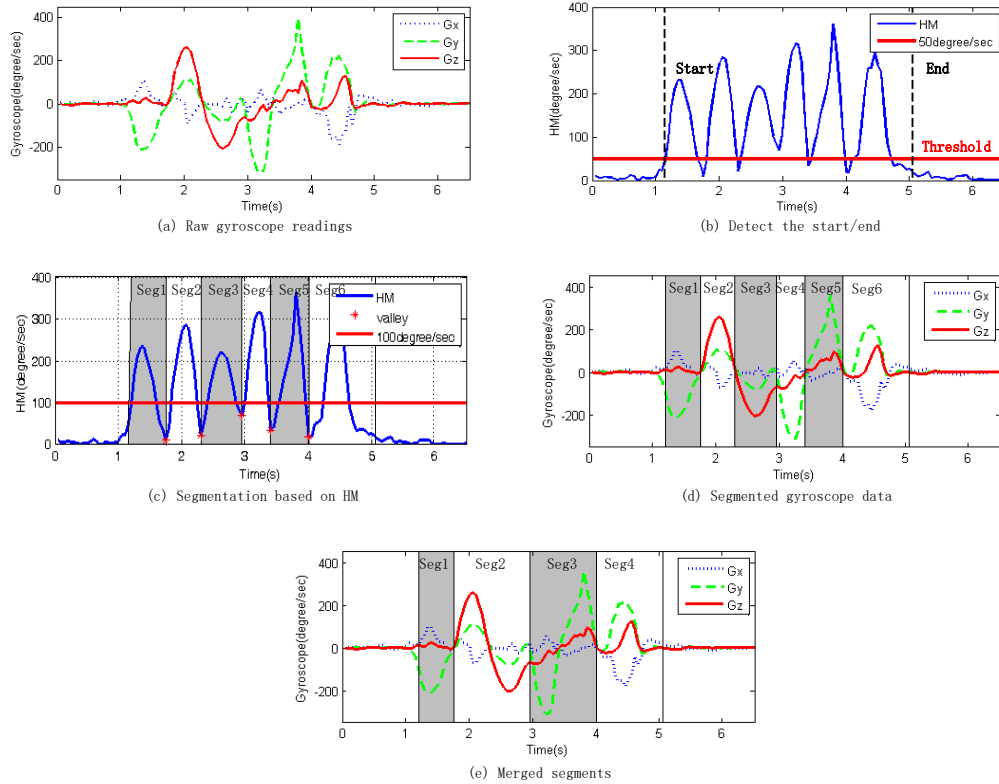
point of the hand movement. The time threshold is necessary as, in one single gesture, the  $HM$  may fall below this threshold occasionally, leading to unexpected splitting of this gesture [52][53]. Because the  $HM$  only keeps the magnitude of the vector sum of three axes and drops the direction information, this threshold-based detection method is independent of the device's orientation and therefore simplifies the gesture models.



**Figure 4.3:**  $HM$  based start and end points detection

Fig. 4.3 shows the gyroscope readings and the  $HM$  of one Left gesture and one Clockwise gesture. From Fig. 4.3(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. 4.4 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. 4.4(a). The corresponding  $HM$  results for this hand movement sequence are shown in Fig. 4.4(b).



**Figure 4.4:** Data processing for one continuous hand movement

#### 4.3.2.2 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 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. 4.4(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. 4.4(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 [5].

#### 4.3.2.3 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 seg-

ments. In Fig. 4.4(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. 4.5 illustrates the computation of Gesture Continuity metric of a Right gesture:

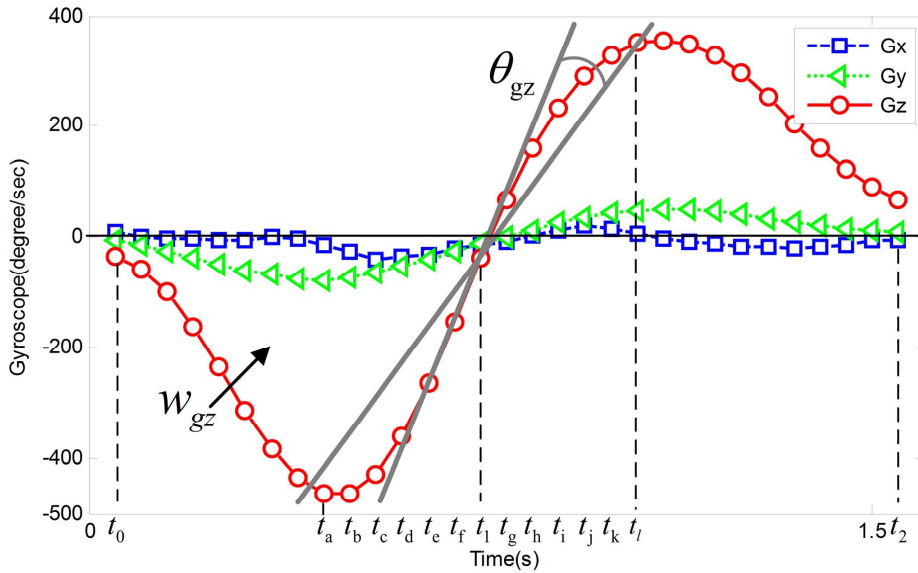
For the sensor reading of each gyroscope axis,  $g_x$ ,  $g_y$  and  $g_z$  (assume  $g_i$ ), we do the following:

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  $\overline{t_a t_1}, \overline{t_b t_1}, \dots, \overline{t_l t_1}$  are formed. For any 2 lines among the 12 lines, the angle between them is computed, and the maximum angle is defined as  $\theta_{g_i}$ ;
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:

$$Con(t_1) = \frac{\sum (w_{g_i} \cdot \theta_{g_i})}{\sum w_{g_i}} \quad (4.2)$$

The higher the angle  $\theta_{g_i}$  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  $\theta_{g_i}$ . 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.



**Figure 4.5:** Computation of Gesture Continuity and Gesture Completeness metric

In Fig. 4.5, the Right gesture is partitioned into two segments  $[t_0, t_1]$  and  $[t_1, t_2]$ . From the figure, we see that angle  $\theta_{g_z}$  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.

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:

$$Com(t_1) = \frac{|\sum_{t_0}^{t_2} g_x| + |\sum_{t_0}^{t_2} g_y| + |\sum_{t_0}^{t_2} g_z|}{\sum_{t_0}^{t_2} |g_x| + \sum_{t_0}^{t_2} |g_y| + \sum_{t_0}^{t_2} |g_z|} \quad (4.3)$$

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. 4.5, 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. 4.2. The Gesture Symmetry (Sym) for these two neighboring segments is calculated as:

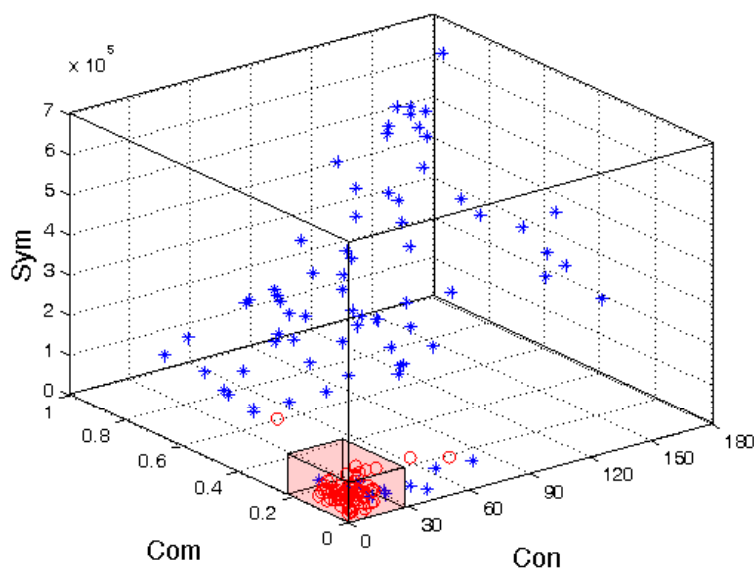
$$Sym(t_1) = \frac{\sum(w_{g_i} \cdot DTW_{g_i})}{\sum w_{g_i}} \quad (4.4)$$

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. 4.6 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. 4.6, 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



**Figure 4.6:** *Con VS Com VS Sym*

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.

From Fig. 4.4(d) to Fig. 4.4(e), we find that segment 2 and 3 in Fig. 4.4(d) are merged into segment 2 in Fig. 4.4(e), which is a Left gesture. Segment 4 and 5 in Fig. 4.4(d) are merged into segment 3 in Fig. 4.4(e), which is a Back&Forth gesture. Each segment in Fig. 4.4(e) contains exactly one complete gesture.

#### 4.3.2.4 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. (4.3). 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. 4.4(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.

### 4.3.3 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 [5]. 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).

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 [5][19][29]. 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 [54].

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 [29]. At runtime, each data segment is classified as one of the predefined gestures or noise. We use the Viterbi algorithm [55] 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.

## 4.4 Performance Evaluation

In this section, we evaluate the performance of the proposed algorithms. We first introduce our dataset in Section 6.3.1. Next, we evaluate the threshold-based gesture segmentation algorithm in Section 4.4.2. Finally, we evaluate the accuracy of the gesture recognition algorithm when the users perform gestures separately and continuously in Section 4.4.3.

### 4.4.1 Data Collection

**Table 4.1:** 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

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 5.1. 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.

#### 4.4.2 Gesture Segmentation Results

**Table 4.2:** Comparison of different metrics for Threshold-based Gesture Segmentation

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

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 4.2 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 best. Its segmentation accuracy is 94.4%. The second one is the Gesture Completeness met-

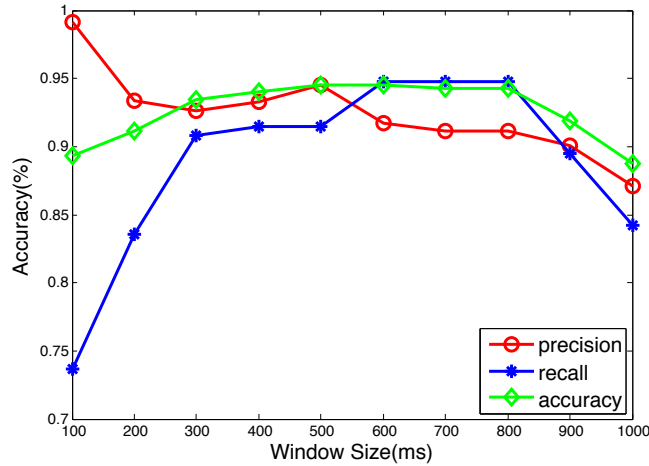
ric. The segmentation accuracy is 93.9%. The worst one is the Gesture Symmetry, the accuracy of which is 90.3%.

**Table 4.3:** Comparison between Machine learning Algorithms and Threshold-based Method for Gesture Segmentation

<b>Algorithm</b>	<b>Precision</b>	<b>Recall</b>	<b>F-Measure</b>	<b>Accuracy</b>
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%

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 [56] 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 [57]), 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 4.3. 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 4.2, 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. 4.7. Three evaluation metrics are considered: precision, recall, and accuracy. When the time window size is



**Figure 4.7:** Precision, recall and accuracy under different window size

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.

#### 4.4.3 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 re-

maining participant. Second, we use all the non-continuous hand gesture samples to train the HMM models, and evaluate the proposed algorithm on the continuous hand gesture samples.

**Table 4.4:** 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 4.5:** 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

**Table 4.6:** 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 4.4 shows the confusion matrix for gesture classification on the Moto dataset with just acceleration features. The average accuracy is 95.4%. Table 4.5 shows the confusion matrix for gesture classification on the Moto dataset with just gyroscope features. The average accuracy is 93.7%. Table 4.6 shows the confusion matrix for gesture classification on the Moto dataset with both acceleration and gyroscope features. The

**Table 4.7:** 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 4.8:** Confusion matrix for gesture classification on the UG dataset with just gyroscope features

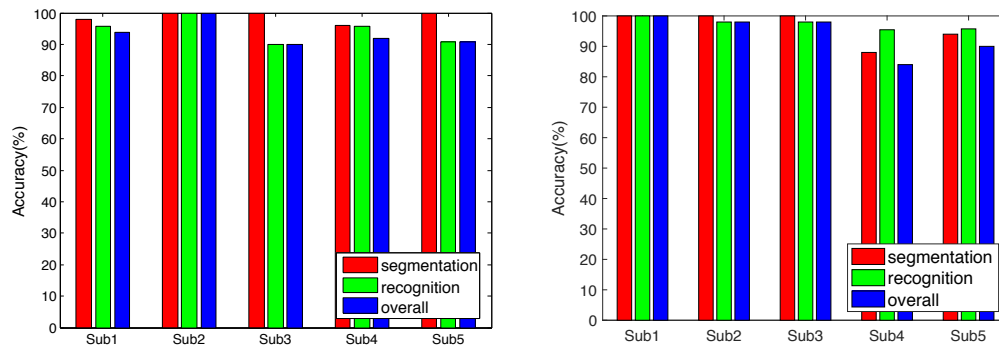
	Left	Right	Up	Down	Back&Forth	Clockwise	Counterclockwise
Left	50	0	0	0	0	0	0
Right	0	50	0	0	0	0	0
Up	0	0	50	0	0	0	0
Down	0	0	0	50	0	0	0
Back&Forth	0	0	8	0	42	0	0
Clockwise	0	0	0	0	0	50	0
Counterclockwise	0	0	0	0	0	0	50

average accuracy is 99.4%. Table 4.7 shows the confusion matrix for gesture classification on the UG dataset with just acceleration features. The average accuracy is 94.9%. Table 4.8 shows the confusion matrix for gesture classification on the UG dataset with just gyroscope features. The average accuracy is 97.7%. Table 4.9 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

**Table 4.9:** 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

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.



**Figure 4.8:** Segmentation and recognition accuracy of the continuous hand gesture recognition algorithm on the Moto dataset. Sub1 means human subject one

**Figure 4.9:** Segmentation and recognition accuracy of the continuous hand gesture recognition algorithm on the UG dataset. Sub1 means human subject one

Fig. 4.8 demonstrates the segmentation accuracy after removing noise segments, recognition accuracy with both 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. 4.9 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 re-

sults demonstrate that the proposed segmentation algorithm and recognition algorithm are very promising.

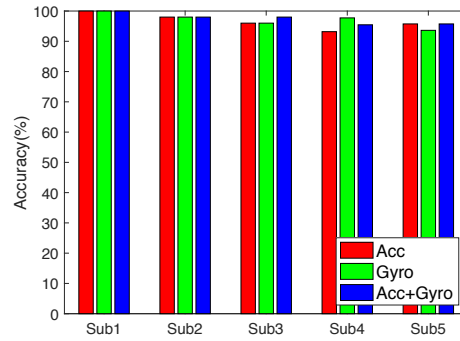
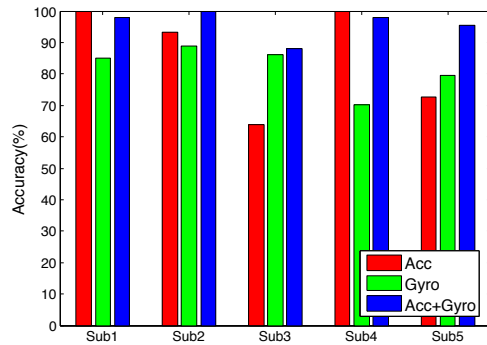
**Table 4.10:** Confusion matrix for classification of continuous hand gestures on the Moto dataset

	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 4.11:** 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

Table 4.10 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 4.11 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.



**Figure 4.10:** Classification accuracy for continuous hand gesture recognition with different HMM models on the Moto dataset. Sub1 means human subject one

**Figure 4.11:** Classification accuracy for continuous hand gesture recognition with different HMM models on the UG dataset. Sub1 means human subject one

Fig. 4.10 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. 4.11 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).

## 4.5 Conclusion

In this chapter, 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 Com-



pleteness 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.

## Chapter 5

# MobiGesture: Mobility-aware Hand Gesture Recognition

### 5.1 Introduction

Regular mobility, such as walking or jogging, is one of the most effective ways to promote health and well-being. It helps improve overall health and reduces the risk of many health problems, such as diabetes, cardiovascular disease, and osteoarthritis [58][59][60]. In addition, regular mobility can also improve depression, cognitive function, vision problems, and lower-body function [61][62]. According to the evidence-based Physical Activity Guidelines released by World Health Organization [63] and the U.S. government [64], adults aged 18-64 should do at least 150 minutes of moderate-intensity or 75 minutes of vigorous-intensity aerobic activity per week, or an equivalent combination of both to keep healthy.

Nowadays, lots of people like to listen to music on their smartphones while they do aerobic activity, such as walking or jogging. Many smartphone apps track users' workouts, play music, and even match the tempo of the songs to users' paces, such as Nike+ Run Club [65], RunKeeper [66], MapMyRun [67]. However, it is inconvenient for the users to interact with these apps while walking or jogging. To change the music, users need to slow down, take out the smartphone, and then change the music. This

is troublesome. Instead, it is more convenient for users to use gestures to control the music. Unlike traditional touchscreen-based interaction, hand gestures can simplify the interaction with a smartphone by reducing the need to take out the smartphone and slow down the pace.

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

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

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

In addition, we integrate the gesture segmentation and classification algorithms into a system called, MobiGesture. For the leave-one-subject-out cross-validation test, exper-

iments with human subjects show that the proposed segmentation algorithm accurately segments the hand gestures with 94.% precision and 91.2% recall when a user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when a user is standing, walking and jogging, respectively.

As far as we know, two efforts have been made to study the gesture recognition problem when a user is moving. Park et al. [5] propose a Multi-situation HMM architecture. They train a HMM model for each pair of hand gesture and mobility situation. As the authors define 8 hand gestures and 4 mobility situations, 32 HMM models are trained in total. Given a hand gesture, they apply the Viterbi algorithm [68] to calculate the likelihood of each HMM model. The HMM model with the highest likelihood is selected as the classified gesture. As the number of the hand gestures and/or the number of the mobility situations increases, their computational cost increases dramatically. Different from their work, we only train one CNN model, which consumes much less computational power and time. In addition, evaluation results show that our CNN model performs better than Multi-situation HMM on gesture classification under leave-one-subject-out cross-validation test. The second effort comes from Murao et al. [30]. They propose a combined-activity recognition system. This system first classifies user movement into one of three categories: postures (e.g., sitting), behaviors (e.g., walking), and gestures (e.g., a punch). Then, Dynamic Time Warping (DTW) is applied to recognize hand gestures for the specific category. However, their system requires five sensors to be attached to the human body for gesture recognition. Instead, we only use one sensor, and hence are less intrusive.

We summarize our contributions as follows:

1. We propose a novel mobility-aware gesture segmentation algorithm to detect and segment hand gestures.
2. We design a CNN model to classify hand gestures. This CNN model conquers mobility noises and avoids overfitting.

3. We integrate the gesture segmentation and classification algorithms into a system, MobiGesture. Our experiments results show that the proposed segmentation algorithm achieves 94.0% precision and 91.2% recall when the user is moving. The proposed hand gesture classification algorithm is 16.1%, 15.3%, and 14.4% more accurate than state-of-the-art work when the user is standing, walking and jogging, respectively.

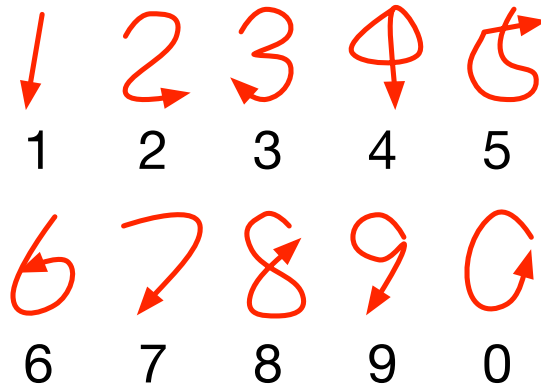
The remainder of this chapter is organized as follows. First, we present the motivation in Section 5.2. Then, we introduce the system architecture in Section 6.2.1. We present our mobility-aware segmentation algorithm in Section 5.4, and CNN model in Section 5.5. In Section 6.3, we evaluate the system performance. Finally, we draw our conclusion in Section 6.5.

## 5.2 Motivation

Hand gestures can help users interact with various mobile applications on smartphones in mobile situations. One common scenario is to control a music app while walking or jogging. We define our hand gestures to be suitable for music control in Section 5.2.1. Based on these defined gestures, we introduce the challenge of gesture recognition when the user is walking or jogging in Section 5.2.2. Finally, we present our data collection and the data set in Section 5.2.3. This data set is used for performance evaluation during the rest of the chapter.

### 5.2.1 Gesture Definition

There has been substantial research on gesture recognition. Some works define gestures according to application scenarios, such as gestures in daily life [19], or repetitive motions in very specific activities [9], while others define gestures casually [5]. In our system, we carefully define the hand gestures that are suitable for controlling a music app. Typically, a music app provides the following functions to control the music: next



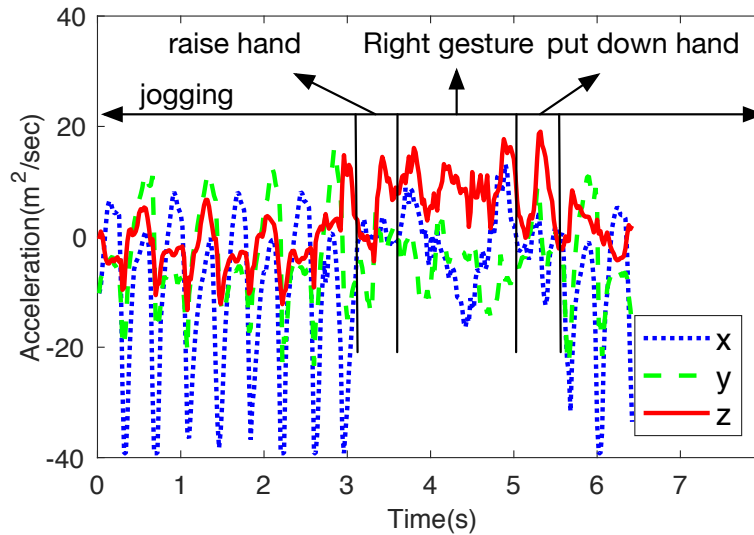
**Figure 5.1:** Ten number gestures for remote control

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

These seven gestures are illustrated in Fig. 4.2 and are defined as follows: (1) Left gesture: move left and then move back to the starting point; (2) Right gesture: move right and then move back to the starting point; (3) Up gesture: move up and then move back to the starting point; (4) Down gesture: move down and then move back to the starting point; (5) Back&Forth gesture: move to shoulder and then extend again to the starting point; (6) Clockwise gesture: draw a clockwise circle; (7) Counterclockwise gesture: draw an counterclockwise circle. In addition, we define 10 number gestures as shown in Fig. 5.1 to select a specific song in the music app.

### 5.2.2 Challenge of Gesture Recognition under Mobility

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



**Figure 5.2:** Accelerometer readings of a Right gesture when a user is jogging

First, it is hard to **detect** gestures while a user is moving. While the user is standing without performing any gesture, the accelerometer readings keep stable. When the user performs a gesture such as the Right gesture, the accelerometer readings change dramatically. Therefore, it is easy to detect a gesture by measuring the amplitude or deviation of the sensor readings. However, when it comes to a jogging scenario, a Right gesture and hand swinging motions are mixed together as shown in Fig. 5.2. Therefore, it is hard to tell whether a hand movement comes from the hand swinging motions or a hand gesture.

The second challenge is that it is hard to **segment** hand gestures while the user is moving. To perform a hand gesture while walking or jogging, the user needs to raise his/her hand, perform a gesture, and then put down his/her hand. To segment hand gestures while the user is moving, we need to not only filter out hand swinging motions caused by body movements, but also accurately exclude the hand-raising and hand-lowering movements. If the starting point and end point of a hand gesture is not precisely determined, it is hard to classify hand gestures accurately. As shown in Fig. 5.2, it is difficult to find the starting point and end point of a Right gesture while jogging.

The third challenge is that it is hard to **classify** hand gestures when a user is mov-

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

### 5.2.3 Dataset

**Table 5.1:** Characteristics of five participants

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

We used a UG wristband [69] to collect 17 hand gestures from 5 human subjects, which is shown in Fig. 5.3. The UG wristband sampled the accelerometer and gyroscope readings at 50 Hz. The data collection experiment contained three independent steps. (1) Each participant performed each gesture 10 times while standing. (2) Each participant performed each gesture 10 times while walking on a treadmill. (3) Each participant performed each gesture 10 times while jogging on a treadmill. In total, 2550 hand gestures were collected. While walking or jogging on a treadmill, different participants tended to walk or jog at different speeds. In our experiment, the speed of walking ranged from 2 miles/hour to 3 miles/hour, and the speed of jogging ranged from 4 miles/hour to 6 miles/hour. We took video of each participant as they completed these tasks to serve as ground truth. The characteristics of our participants are shown in Table 5.1.



### 5.3 System Architecture

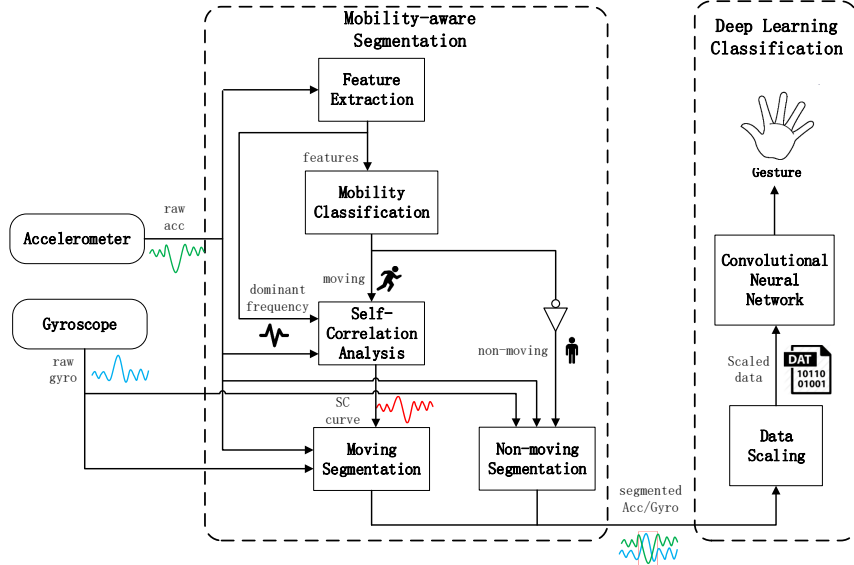


(a) Placement of a UG wristband



(b) Coordinate System of a UG wristband

**Figure 5.3:** UG Wristband



**Figure 5.4:** System Architecture

The system architecture of the MobiGesture is shown in Fig. 6.1. We apply a novel mobility-aware segmentation module to partition the raw accelerometer and gyroscope readings into segments so that each segment contains one complete hand gesture. In the mobility-aware segmentation module, we first detect whether or not the user is moving. We extract a series of time-domain and frequency-domain features from accelerometer readings and apply an AdaBoost Classifier to classify the current body movement into moving or non-moving. If the user is not moving, sensor readings are clean and do not contain any mobility noise. In this case, we apply a simple threshold-based segmentation algorithm to segment the hand gestures.

On the other hand, if the user is walking or jogging, the sensor readings are periodic and self-correlated. We perform Fast Fourier Transform (FFT) on accelerometer readings and compute the dominant frequency, which is the frequency of walking or jogging. Based on the dominant frequency, we propose a novel self-correlation metric,

*SC*. This metric represents the self-correlation characteristics of accelerometer readings at the given frequency. When the user is walking or jogging, the sensor readings are self-correlated at the dominant frequency. Once the sensor readings are no longer self-correlated at the dominant frequency, we regard it as a potential gesture sample. Then, a moving segmentation algorithm is applied to partition the accelerometer and gyroscope readings into segments based on *SC* metric.

As the 17 predefined gestures are different from each other and users tend to perform the gestures at different speeds, the duration of each gesture is different. Therefore, the size of each segment is different. We apply a Cubic Spline Interpolation algorithm [70] to rescale the size of each segment so that each segment contains the same data points. Finally, we design a 9-layer Convolutional Neural Network to recognize hand gestures. The Convolutional Neural Network is designed to be anti-overfitting and robust to mobility noises.

## 5.4 Mobility-aware Segmentation

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

these challenges, we propose a mobility-aware segmentation algorithm. we first classify the body movement into non-moving or moving. Then, we propose a non-moving segmentation algorithm and a moving segmentation algorithm to segment hand gestures for two different moving scenarios.

#### **5.4.1 Feature Extraction**

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

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

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

For the time-domain features, we compute the mean of the accelerometer readings of the X-axis, Y-axis, and Z-axis accordingly, the mean of the pitch, and the mean of the

roll to represent the orientation of the hand for each time window. The pitch and roll are computed as

$$Pitch = \arctan \left( \frac{Acc_y}{\sqrt{(Acc_x)^2 + (Acc_z)^2}} \right), \quad (5.1)$$

$$Roll = -\arctan \left( \frac{Acc_x}{Acc_z} \right), \quad (5.2)$$

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

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

$$Acc = \sqrt{(Acc_x)^2 + (Acc_y)^2 + (Acc_z)^2}, \quad (5.3)$$

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

#### 5.4.2 Mobility Classification

We apply the WEKA machine-learning suite [56] to train five commonly used classifiers. The classifiers include AdaBoost (run for 100 iterations), Naive Bayes, SVM (with polynomial kernels), J48 (equivalent to C4.5 [57]), and Random Forests (100 trees, 4 random features each). To evaluate the performance of the proposed algorithms, we apply two tests: 5-fold cross-validation and leave-one-subject-out (LOSO) cross-validation. The 5-fold cross-validation test uses all the gesture data to form the dataset. It partitions the dataset into 5 randomly chosen subsets of equal size. Four subsets are used to train the model. The remaining one is used to validate the model. This process is repeated 5 times such that each subset is used exactly once for validation. The leave-one-subject-out cross-validation test uses the gesture data from four subjects to train the classifica-

tion model, and then applies this model to test the gesture samples from the remaining subject. Precision, recall, and F-measure are considered as the evaluation metrics.

**Table 5.2:** Comparison of Machine learning Algorithms for Mobility Classification

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

The classification results for these five algorithms are shown in Table 5.2. Under the 5-fold cross-validation test, RandomForest performs the best. The precision, recall, and F-measure are 96.9%, 96.9%, and 96.9%, respectively. Under the leave-one-subject-out cross-validation test, AdaBoost performs the best. The precision, recall, and F-measure are 94.9%, 94.6%, and 94.4%, respectively. We favor the leave-one-subject-out cross-validation test over the 5-fold cross-validation test to avoid overfitting. Therefore, we choose AdaBoost classifier to classify the body movement.

### 5.4.3 Non-Moving Segmentation

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

$$HM = \sqrt{Gyro_x^2 + Gyro_y^2 + Gyro_z^2}, \quad (5.4)$$

where  $Gyro_x, Gyro_y, Gyro_z$  are the gyroscope readings of the X-axis, Y-axis, and Z-axis. When the user's hand is stationary, the  $HM$  is very close to zero. The faster a hand moves, the larger the  $HM$  is. When the  $HM$  is larger than a threshold, i.e. 50

degree/second, we regard it as the starting point of a hand movement. Once the  $HM$  is smaller than this threshold for a certain period of time, i.e.  $200\text{ ms}$ , we regard it as the end point of the hand movement. The time threshold is necessary. Without it, the  $HM$  may fall below this threshold occasionally, leading to unexpected splitting of this gesture [52][53]. As a gesture does not last shorter than  $260\text{ ms}$  or longer than  $2.7$  seconds in our dataset, we drop a segment if the length of this segment is shorter than  $260\text{ ms}$  or longer than  $2.7$  seconds.

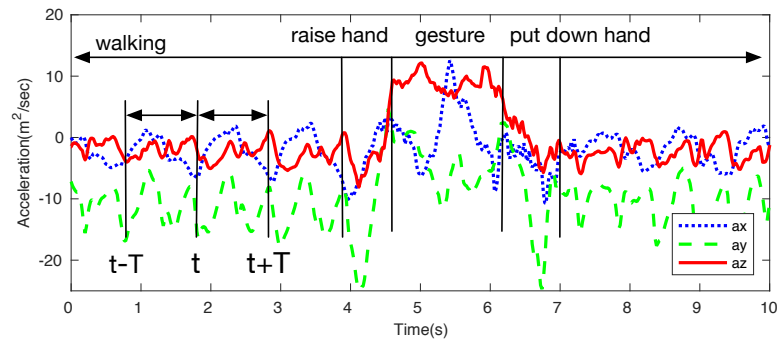
#### 5.4.4 Self-Correlation Analysis

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

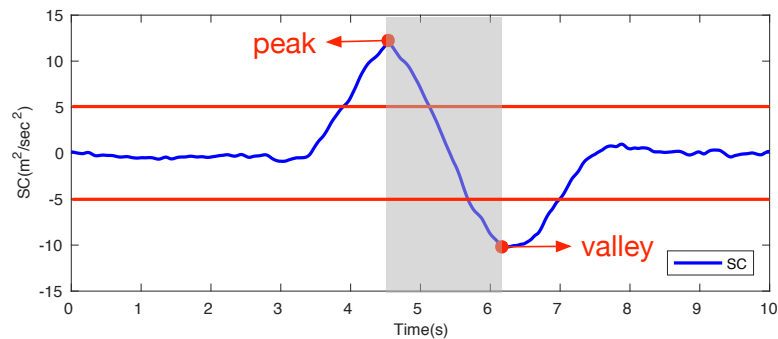
$$SC(t) = \sum_{i \in \{x,y,z\}} \sum_{j=1}^T [Acc_i(t+j) - Acc_i(t+j-T-1)] / T, \quad (5.5)$$

where  $Acc_i$  ( $i \in \{x, y, z\}$ ) are the accelerometer readings of the X-axis, Y-axis, and Z-axis.  $T$  is the cycle of the walking or jogging, which is computed as the inverse of the dominant frequency.  $t$  is the current time. If the accelerometer readings are self-correlated at the dominant frequency, the  $SC$  is very close to zero. If the accelerometer readings are not self-correlated at the dominant frequency, the  $SC$  is either a large positive value or a large negative value. Fig. 5.5(a) shows the accelerometer readings of a Left gesture when a user is walking. The computed  $SC$  curve is in Fig. 5.5(b).

From Fig. 5.5(a) and (b), we find that the  $SC$  is very close to zero when the user swings his/her hand during walking. The  $SC$  begins to increase when the user raises his/her hand. The peak of the  $SC$  occurs when the user finishes raising his/her hand and begins to perform a Left gesture. The valley of the  $SC$  occurs when the user finishes



(a) Acceleration data of a Left gesture while walking



(b) SC curve

**Figure 5.5:** Segmentation when a user is moving

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

When the user is walking, the hand is pointing to the ground. Impacted by the force of gravity, the accelerometer readings of the Y-axis are always negative. When the user is jogging, the hand is pointing forward with the palm facing towards the user. In this case, the accelerometer readings of the X-axis are impacted by gravity and always have negative values. However, when the user raises his/her hand and performs the gesture, the palm faces the ground. The accelerometer readings of the Z axis are impacted by gravity and have positive values. Therefore, when the user is walking or jogging, the sum of

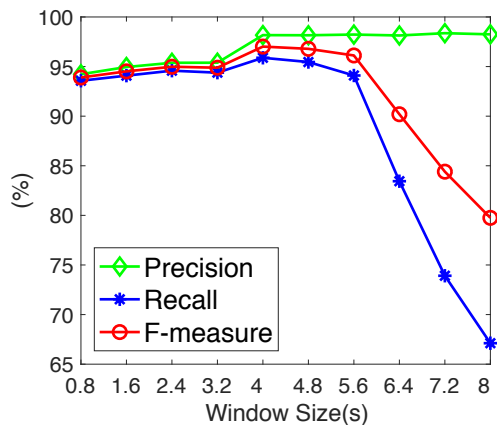
the accelerometer readings in 3 axes are always negative. When the user raises his/her hand, the sum of the accelerometer readings increases and reaches the maximum right after raising hand. As the  $SC$  is computed by the difference of the accelerometer readings in two adjacent time window (window size is the cycle of walking or jogging),  $SC$  reaches the peak right after raising hand. Similarly,  $SC$  reaches the valley right after putting down hand. Therefore, we use the peak and the valley of the  $SC$  curve as the starting point and the end point of the Left gesture.

#### 5.4.5 Moving Segmentation

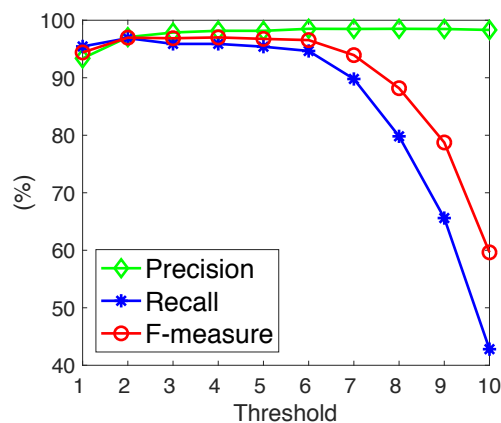
We segment the hand gestures when the user is moving by searching for the peak and valley of the  $SC$ . We compute the  $SC$  metric from the accelerometer readings. If the  $SC$  value is larger than  $5 \text{ m}^2/\text{sec}^2$ , we start to search for the peak of the  $SC$  within a 4 second time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

Fig. 5.6 and Fig. 5.7 show the performance of the moving segmentation under different window sizes and different  $SC$  thresholds accordingly. Three evaluation metrics are considered: precision, recall, and F-measure. As the window size or the  $SC$  threshold increases, the segmentation precision increases and the recall decreases. When the window size is 4 s and the  $SC$  threshold is  $5 \text{ m}^2/\text{sec}^2$ , the F-measure is at its highest value: 93.7%. Therefore, we choose 4 s as the time window size and  $5 \text{ m}^2/\text{sec}^2$  as the  $SC$  threshold to segment the hand gestures when the user is moving.





**Figure 5.6:** Segmentation precision, recall, and F-measure under different window sizes



**Figure 5.7:** Segmentation precision, recall, and F-measure under different *SC* thresholds

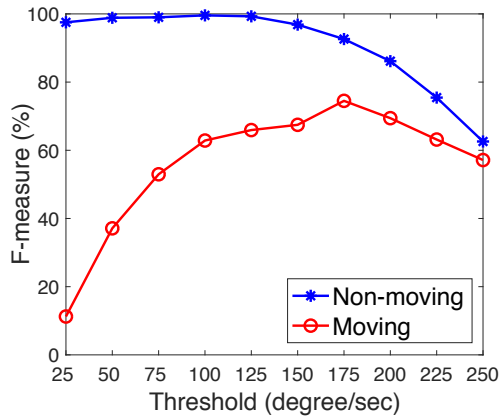
#### 5.4.6 Performance

We compare our gesture segmentation algorithm with E-gesture [5], which is the state-of-the-art. E-gesture segments the hand gestures based on the amplitude of the gyroscope readings. A hand gesture is triggered if the amplitude of the gyroscope readings is higher than 25 degree/sec. The triggered gesture is assumed to have ended if the amplitude of the gyroscope readings is lower than 25 degree/sec for 400 ms. Different from E-gesture, we first apply the AdaBoost classifier to classify the body movement into moving or non-moving. Then, we apply two different segmentation algorithms to segment the hand gestures accordingly.

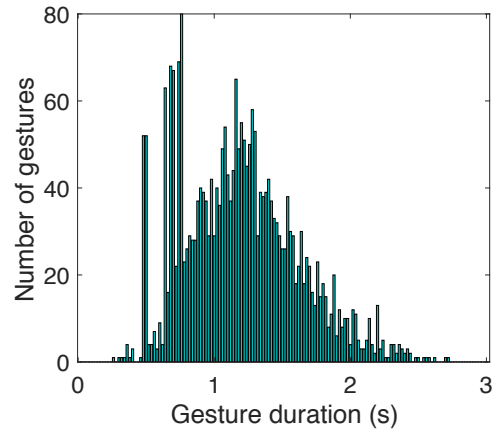
**Table 5.3:** Comparison of the Gesture Segmentation Performance

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

We evaluate the segmentation accuracy by checking the overlap between a segment and a hand gesture. If the middle of a hand gesture lies in a segment, this gesture is



**Figure 5.8:** F-measure of E-gesture under different thresholds



**Figure 5.9:** The distribution of the gesture duration in our dataset

correctly segmented by that segment. The performance of MobiGesture and E-Gesture are shown in Table 5.3. From the table, we find that MobiGesture performs stably in both moving scenarios. The F-measure in two moving scenarios are around 92%. E-gesture performs well when the user is not moving. However, it performs poorly when the user is moving. When the user is moving, the F-measure of E-gesture is only 11.3%. The possible reasons are: (1) the sensors in their wristband are different from ours; (2) their predefined hand gestures are different from ours.

We change the threshold of E-gesture from 25 degree/sec to 250 degree/sec with a step of 25 degree/sec and evaluate the performance of E-gesture. Fig. 5.8 shows the F-measure of E-gesture under different thresholds. When the threshold is 175 degree/sec, the F-measure of E-gesture in moving scenario is at its highest value: 74.5%. This is still much lower than the accuracy of our moving segmentation algorithm: 93.7% under 5-folder cross-validation test, and 92.2% under leave-one-subject-out cross-validation test. Therefore, their segmentation algorithm can not accurately segment the hand gestures when the user is moving. Their solution is not general enough to be extended to the new gestures and hardware platform.

When a user is standing without performing any gesture, the gyroscope readings are close to zero. The amplitude of the gyroscope readings is a good measurement to

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

## 5.5 Deep Learning Classification

Two approaches are popular for classifying hand gestures. One is to use conventional machine learning classifiers, such as Naive Bayes [24], Random Forest [9], and Support Vector Machines [23]. The other is to use sequential analysis algorithms, such as Hidden Markov Model (HMM) [5] and Dynamic Time Warping (DTW) [30].

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

### 5.5.1 Data Scaling

As 17 predefined hand gestures are different from each other and different users perform hand gestures at different speeds, the duration of each hand gesture is different. Fig. 5.9 shows the distribution of the gesture duration in our dataset. The maximum gesture duration is 2.7 seconds. The minimum gesture duration is 260 ms. The average gesture

duration is 1.2 seconds. As the sampling rate is 50 Hz, each gesture contains 60 sample points on average.

As a CNN model requires input data with the same size, we format the segment data so that each segment has the same size. We apply the Cubic Spline Interpolation [71] to rescale the number of sample points for each segment to 60. As 3-axis accelerometer readings and 3-axis gyroscope readings are collected by each sampling,  $60 \times 6$  data points are generated after interpolation. This  $60 \times 6$  data matrix is used for classification.

## 5.5.2 Convolutional Neural Network

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

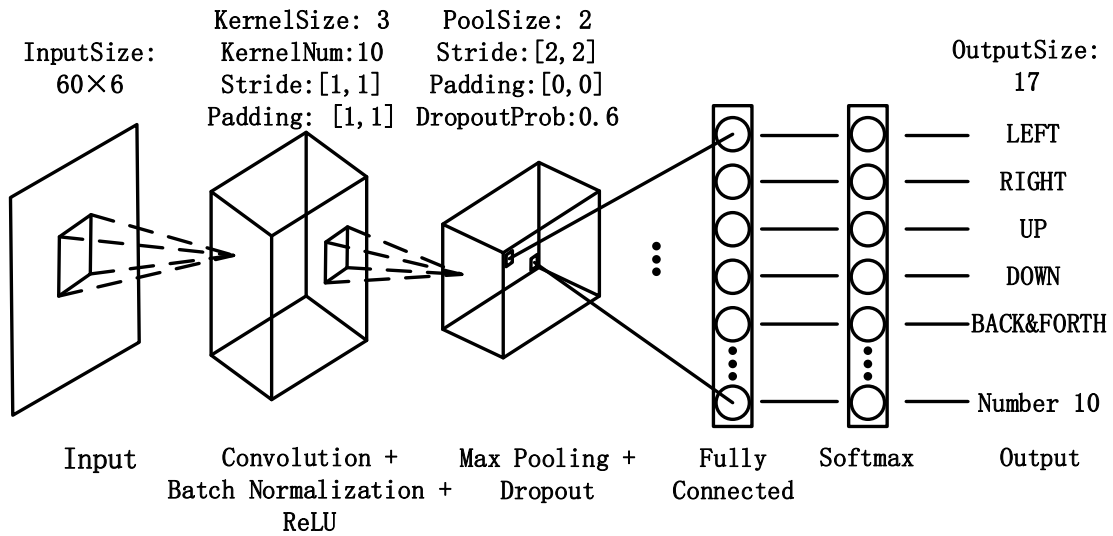
$$\mathbf{y}^{(i)} = \sigma^{(i)} \left( \mathbf{W}^{(i)} \mathbf{x}^{(i)} + \mathbf{b}^{(i)} \right), \quad (5.6)$$

where  $\mathbf{y}^{(i)}$  is the output,  $\mathbf{x}^{(i)}$  is the input,  $\sigma^{(i)}$  is the activation function,  $\mathbf{W}^{(i)}$  is the weight matrix, and  $\mathbf{b}^{(i)}$  is the bias vector [72].  $\mathbf{x}^{(0)}$  is the original input, which is a matrix of the accelerometer and gyroscope sensor data.  $\mathbf{y}^{(n)}$  is the final output, which is one of 17 predefined hand gestures. The output of the  $(i - 1)$ -th layer is the input of the  $i$ -th layer, i.e.,  $\mathbf{x}^{(i)} = \mathbf{y}^{(i-1)}$ .

Fig. 5.10 shows the architecture and parameter settings of the CNN. It includes the following 9 layers:

### 5.5.2.1 Input Layer

The input layer is the entrance to the CNN. It provides data for the following layers. After data scaling, we get a  $60 \times 6$  data matrix. This matrix is supplied to the input layer.



**Figure 5.10:** Architecture and parameter settings of the 9-layer CNN

### 5.5.2.2 Convolutional Layer

The convolutional layer divides the input data into multiple regions. For each region, it computes a dot product of the weights and the input, and then adds a bias term. A set of weights that are applied to a region is called a kernel. The kernel moves along the input data vertically and horizontally, repeating the same computation for each region. The step size with which it moves is called a stride. We use ten  $3 \times 3$  kernels and stride of 1 in both vertical and horizontal directions. To preserve the output size of the convolutional layer, we use a padding of 1 in both vertical and horizontal directions. It adds rows or columns of zeros to the borders of the original input.

### 5.5.2.3 Batch Normalization Layer

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

#### **5.5.2.4 ReLU Layer**

Convolutional and batch normalization layers are usually followed by a nonlinear activation function. We choose a Rectified Linear Unit (ReLU) as the activation function. It performs a threshold operation on each input, where any input value less than zero is set to zero. ReLU is easy to compute and optimize. It provides fast and effective training for deep neural networks. It has been shown more effective than traditional activations, such as logistic sigmoid and hyperbolic tangent, and is widely used in CNN [72].

#### **5.5.2.5 Max-pooling Layer**

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

#### **5.5.2.6 Dropout Layer**

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

#### **5.5.2.7 Fully-connected Layer**

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

### 5.5.2.8 Softmax Layer

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

### 5.5.2.9 Classification Layer

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

As in many other learning systems, the parameters of a CNN model are optimized to minimize the loss function. We apply the Stochastic Gradient Descent with Momentum [73] to learn the CNN parameters (weights  $\mathbf{W}$  and biases  $b$ ). It updates the parameters of the CNN by taking small steps in the direction of the negative gradient of the loss function:

$$\theta_{l+1} = \theta_l - \alpha \nabla E(\theta_l) + \gamma(\theta_{l+1} - \theta_l), \quad (5.7)$$

where  $\theta$  is the parameter vector,  $l$  is the iteration index,  $\alpha$  is the learning rate,  $E(\theta)$  is the loss function, and  $\gamma$  is the momentum term [72]. The momentum term  $\gamma$  controls the contribution of the previous gradient step to the current iteration. We use a momentum term of 0.9 and a learning rate of 0.03.

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

$$E_R(\theta) = E(\theta) + \lambda \Omega(\mathbf{W}), \quad (5.8)$$

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

### 5.5.3 Performance

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

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

We compare our gesture classification algorithm with E-gesture [5]. E-gesture proposes a Multi-situation HMM model for gesture classification. During training, a HMM model is built and trained for each pair of gesture and mobility situation. During testing, E-gesture computes the Viterbi scores [68] for each of the HMM models, and the best candidate is selected to be the classification result. We call this method Multi-situation HMM.



**Table 5.4:** Comparison of the Gesture Classification Performance

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

We apply the 5-fold cross-validation test and leave-one-subject-out cross-validation test to evaluate the gesture classification performance. Table 5.4 shows the gesture classification accuracy of these two algorithms under three different moving scenarios. Under the leave-one-subject-out cross-validation test, CNN is 16.1%, 15.3%, and 14.4% more accurate than Multi-situation HMM when the user is standing, walking and jogging, respectively. Under the 5-fold cross-validation test, Multi-situation HMM is roughly 7% more accurate than CNN. For Multi-situation HMM, the average accuracy is 96.8% under the 5-fold cross-validation test, and 66.8% under the leave-one-subject-out cross-validation test. There is 30% accuracy difference between these two tests. It shows that Multi-situation HMM model is overfitted. For CNN, the accuracy difference between these two tests is 8.2%. The reasonably small difference shows that overfitting is significantly reduced.

## 5.6 Performance Evaluation

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

and compare it with state-of-the-art work.

### 5.6.1 Accuracy

**Table 5.5:** Comparison of the Overall Performance

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

The overall performance of MobiGesture and E-gesture are shown in Table 5.5. Under the 5-fold cross-validation test, E-gesture performs well when the user is standing. However, when the user is walking or jogging, the accuracy of E-gesture is very low. The reason is that E-gesture can not differentiate the hand gestures from the hand swinging motions. MobiGesture performs stably under different moving scenarios. The recognition accuracy when the user is jogging is only 2.2% lower than that when the user is standing. Under the leave-one-subject-out cross-validation test, when the user is standing, the accuracy of E-gesture is only 69.1%. It is much lower than the accuracy under 5-fold cross validation: 95.8%. It shows that the gesture classification model in E-gesture is overfitted. When the user is walking or running, E-gesture performs poorly again due to the low segmentation accuracy. The accuracy of MobiGesture under the leave-one-subject-out cross-validation test is 3.6% ~ 12.2% lower than that under the 5-fold cross-validation test. The reasonably small difference shows the effectiveness of MobiGesture’s anti-overfitting design.

## 5.6.2 Time Delay

**Table 5.6:** Comparison of the Time consumption

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

Table 5.6 shows the time consumption of training and testing of Multi-situation HMM and CNN. For the training time, Multi-situation consumes 58% more time than CNN. For the testing time, Multi-situation HMM consumes roughly three times as much as CNN. Multi-situation HMM trains a HMM model for each pair of the hand gestures and the moving scenarios, while MobiGesture only trains one CNN model for all the hand gestures and moving scenarios. As the number of moving scenarios increases, Multi-situation HMM consumes more time for testing, while our CNN keeps the same. Therefore, CNN is more practical than Multi-situation HMM to be implemented for real-time classification.

## 5.7 Conclusion

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

## Chapter 6

# TennisEye: Tennis Ball Speed Estimation using a Racket-mounted Motion Sensor

### 6.1 Introduction

With the advance of ubiquitous computing, cyber physical systems, and human computer interaction, wearable devices are becoming more and more popular nowadays [74] [75]. Applications of wearable devices have been widely extended to the field of sports. For example, Hao et al. propose a running rhythm monitoring system based on the sound of breathing through smartphone embedded sensors [76]. Kranz et al. propose an automated assessment system for balance board training called Gymskill, which provides feedback on training quality to the user based on smartphone integrated sensors [77]. In addition, there are several studies in other sports like running [78], skiing [79], climbing [80], cricket [81], football [82], and table tennis [83]. In addition to these research publications, the industrial wearable devices market is also evolving at a rapid pace. The global industrial wearable devices market has reached a value of 1.5 billion dollars in 2017, and is expected to have a compound annual growth rate (CAGR) of 9.6% [84]. The wearable devices market in sports is expected to register a CAGR of 9.8%, during

the forecast period (2018 - 2023) [85].

There are also applications of wearable devices in tennis training. Several commercial tennis assistant systems are available on the market that aim to improve players' performance [13] [14] [15]. These products either integrate the motion sensors inside the racket [15], or require users to attach the motion sensors onto the racket [13] [14]. They analyze the motion sensor data and compute the key performance metrics for each swing, such as stroke type, ball speed, ball spin, and ball impact location. In addition to these commercial products, there are also several existing research works on analyzing the performance of tennis shots. For instance, Srivastava et al. analyze the consistency of the tennis shots [43]. The authors provide recommendation on wrist rotation based on the shots from professional players. Sharma et al. analyze the tennis serve [45]. By comparing the serve phases of a user to those of professionals, the system provide the user with corrective feedback and insights into their playing styles.

Tennis ball speed is an important metric in assessing the skill level of a tennis player. There are two main ways to calculate the tennis ball speed. One way is to use multiple high-speed cameras to capture ball movement and calculate speed, such as Hawk-eye technology [10] [11] and PlaySight [12]. These systems use advanced image processing and analytical algorithms to capture ball movement and calculate speed. However, high-speed cameras are very expensive and hard to set up. Therefore, most players cannot get access to these systems, which limits their popularity. Another way is to use motion sensors. Compared with camera-based method, the motion sensors-based method is lower cost, more energy efficient, not influenced by lighting environment, and easier to set up. There are some commercial products on the market that assess the performance of the players and estimate the ball speed [13] [14] [15]. However, none of these commercial products open their algorithms to the public. In addition, to our knowledge, no previous publication has used motion sensors to calculate the tennis ball speed. Therefore, we are motivated to explore how to use a motion sensor to calculate the tennis ball speed.

There are several research publications and commercial products that use motion sensors to analyze tennis shots. However, none of them open their source codes and datasets to the public. Source codes and tennis dataset sharing are valuable as they allow researchers to build their works upon others rather than repeat already existing research work. In addition, they encourage more connection and collaboration between researchers, which promotes the tennis research. Therefore, we are motivated to open our source codes and tennis dataset to the public<sup>1</sup>.

We present TennisEye, a tennis ball speed calculation system using a racket mounted sensor. We apply a simple threshold-based method to detect if there is a tennis stroke. Once a tennis stroke is detected, we use a time window to extract stroke data and interpolate the sensor readings if the true value exceeds the measurement limit. We apply the Random Forest classifier to classify each tennis stroke into one of three stroke types: serve, groundstroke, and volley. A serve is a shot to start a point. A groundstroke is a shot that is executed after the ball bounces once on the ground, while a volley is a shot that is executed before the ball bounces on the ground. If the tennis stroke is a serve, we apply a regression model to calculate serve speed. If the tennis stroke is a groundstroke or volley, we propose two models: a physical model and a regression model. For advanced players, they have consistent and correct stroke gestures. We use the physical model to calculate the ball speed for them. For beginning players, they have inconsistent and even incorrect stroke gestures. We use the regression model to calculate the ball speed.

We summarize our contributions as follows:

1. We propose a tennis ball speed calculation system, TennisEye. It is the first research publication to calculate the serve, groundstroke and volley speed of a tennis ball using a racket-mounted motion sensor.
2. We propose two models to calculate the groundstroke speed: a physical model and a regression model. We apply the physical model to calculate the ball speed

---

<sup>1</sup><https://hongyang-zhao.github.io/TennisEye/>

for advanced players, and the regression model to calculate the ball speed for beginning players.

3. We evaluate the proposed system using the tennis shot data from players of different levels. Our experiment results show that TennisEye is 10.8% more accurate than the state-of-the-art work.
4. We collect an accurate and high-quality tennis dataset. We open our source codes and tennis dataset to the public.

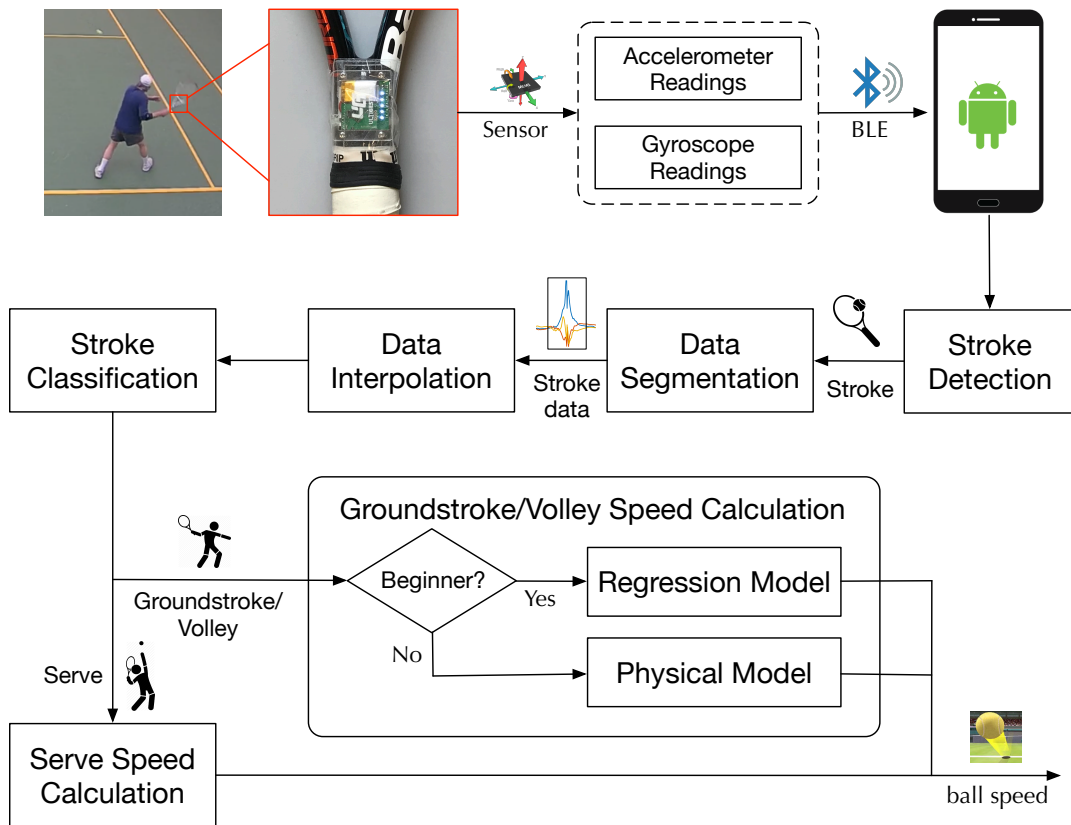
The remainder of this chapter is organized as follows. First, we introduce the system design in Section 6.2. In Section 6.3, we evaluate the system performance. We discuss the system and future work in Section 6.4. Finally, we draw our conclusion in Section 6.5.

## 6.2 TennisEye Design

We introduce the system design of TennisEye in this section. First, we introduce the data processing in Section 6.2.1. Then, we introduce the sensor deployment and data collection in Section 6.2.2. Following that, modules in the data processing are introduced one by one: stroke detection in Section 6.2.3, data segmentation in Section 6.2.4, data interpolation in Section 6.2.5, stroke classification in Section 6.2.6, serve speed calculation in Section 6.2.7, and groundstroke/volley speed calculation in Section 6.2.8.

### 6.2.1 Overview of TennisEye

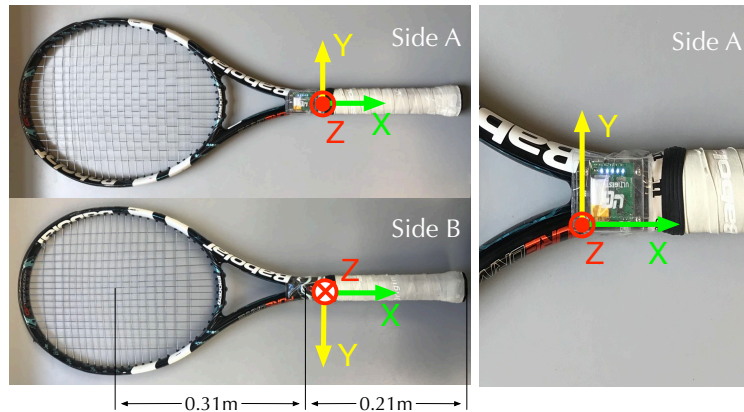
The data processing of TennisEye is shown in Fig. 6.1. A motion sensor is deployed on a racket handle to collect the accelerometer and gyroscope readings, which are real-time transmitted to a smart phone through Bluetooth Low Energy (BLE). Based on the collected motion data, a threshold-based method is proposed to detect tennis strokes. Once a tennis stroke is detected, we apply a peak detection algorithm to find the impact time when a ball hits the racket. Once the impact time is found, we use a sliding window



**Figure 6.1:** Data Processing of TennisEye

with a length of 2 seconds to extract the stroke data. For the stroke data, we apply a cubic spline interpolation to compensate the sensor readings if the sensor saturates. Then, we use the Random Forest classifier to classify each tennis stroke into one of three stroke types: serve, groundstroke, and volley. If the tennis stroke is a serve, we propose a serve speed estimation method to calculate serve speed. If the tennis stroke is a groundstroke or volley, we propose two models, a regression model and a physical model, to calculate the ball speed for the beginning player and the advanced player, respectively. We train the models and evaluate the performance of the system using a laptop.





**Figure 6.2:** Sensor placement and coordinate system

### 6.2.2 Sensor Deployment

The motion sensor we used is a UG sensor [69]. It includes a triaxial acceleration sensor and a triaxial gyroscope. The measure range of the accelerometer and the gyroscope were set to be  $\pm 16g$  and  $\pm 2000^\circ/sec$ , respectively. Both sensors were sampled with 100 Hz. The UG sensor was fixed at the handle of the racket. Fig. 6.2 shows the sensor position and the coordinate system. We call the side with the UG sensor side A, and the side without the UG sensor side B.



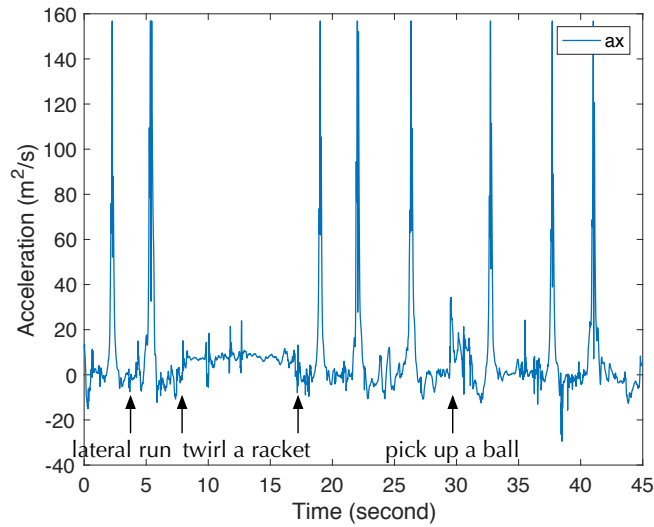
**Figure 6.3:** Birdview of the tennis court from a PlaySight camera

We collected data in a tennis court that was equipped with a PlaySight system [12], which included six high definition (HD) cameras. The PlaySight system uses image processing algorithm to recognize stroke type, ball speed, and more. We use the stroke

type and the ball speed calculated by the PlaySight system as the ground truth. The birdview of the tennis court from a PlaySight camera is shown in Fig. 6.3.

### 6.2.3 Stroke Detection

When a player plays tennis, he/she not only swings a racket, but also performs some non-stroke actions during a match. For example, he/she may run on the court, use the racket to pick up a ball from the court surface, or twirl the racket in his/her hands while waiting for an opponent to serve. The aim of the stroke detection is to detect stroke behaviors, rather than those non-stroke actions.



**Figure 6.4:** Accelerometer readings in the X-axis when a player plays tennis

We find that when a player hits a ball, the player swings the racket in a circular motion. In this circular motion, the racket is affected by a centripetal acceleration that points to the human torso. This centripetal acceleration generates a spike in the accelerometer readings in X-axis,  $a_x$ . Fig. 6.4 shows  $a_x$  when a player plays tennis. Each spike in the figure is a tennis stroke. Other non-stroke actions, such as lateral run, twirling a racket, or picking up a ball, do not generate large values in  $a_x$ . Therefore, we use  $a_x$  to detect stroke behaviors. If  $a_x$  is larger than a threshold, i.e.  $9g$ , it is regarded as a stroke.

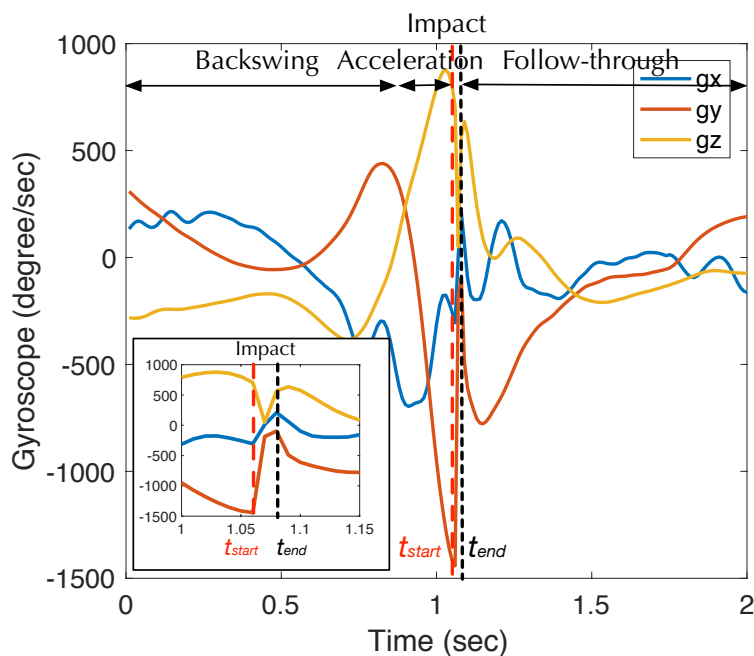
## 6.2.4 Data Segmentation

After a stroke is detected, the next step is to find the start and end time of a tennis stroke and extract this interval of sensor data as stroke data. The extracted stroke data will be used for ball speed calculation.

There are mainly four phases for a tennis stroke: backswing, acceleration, impact, and follow-through. In a backswing phase, a player moves a racket behind his/her body and prepares to swing forward. In an acceleration phase, the player moves the racket forward. The speed of the racket increases in this phase. At the end of the acceleration phase and the start of the impact phase, the speed of the racket reaches maximum. In an impact phase, the racket hits a tennis ball. Due to impact, the speed of the racket decreases rapidly. Finally, in a follow-through phase, the player slows down the racket after hitting a ball.

Based on our study, we find that the impact phase occurs in the middle of these four phases. In addition, the speed of the racket reaches maximum at the start of the impact phase. Therefore, by looking for the time when the speed of the racket reaches maximum, we find the start of the impact phase. In our coordinate system, no matter which stroke (serve, forehand, backhand) a player plays, the racket always rotates around Y-axis. Therefore, the position with the max absolute value of the gyroscope readings in the Y-axis is the start of the impact phase.

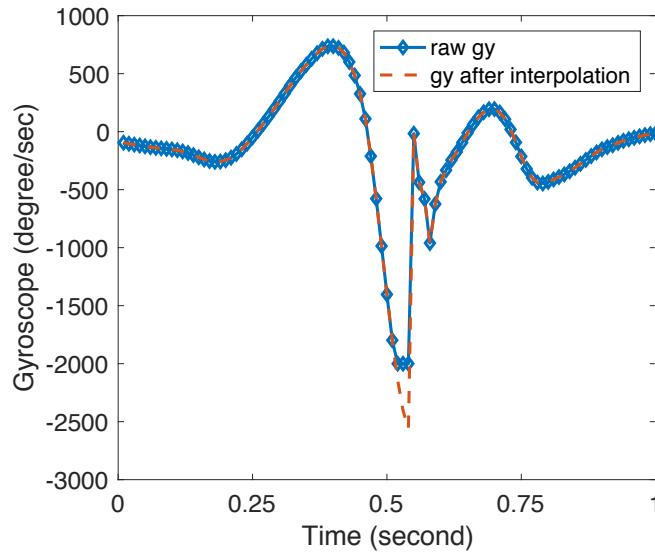
Fig. 6.5 shows the gyroscope readings of a tennis serve. From the figure, we find that the valley of  $g_y$  is the start of the impact phase. We apply a peak detection algorithm with a sliding window on the absolute values of the gyroscope readings in the Y-axis,  $|g_y|$ , to find the impact start time. As the minimum interval between two consecutive strokes is 2 seconds, the time window size is set to be 2 seconds. Specifically,  $|g_y(t)|$  is a peak if it is larger than all of the samples in the time window of  $[t - 1s, t + 1s]$ . Since the minimum of  $|g_y|$  for a stroke in our dataset is 814.5 degree/sec, the detected peak of  $|g_y|$  should be larger than 814.5 degree/sec. After the impact start time  $t_{start}$  is detected, we use a time window of  $[t_{start} - 1s, t_{start} + 1s]$  to extract the stroke data.



**Figure 6.5:** Gyroscope readings of a serve

### 6.2.5 Data Interpolation

In our dataset, we find that the gyroscope readings in the Y-axis sometimes saturate when a player serves with a high speed. This saturation happens when a sensor measures a value that is larger than its measurement range  $\pm 2000^\circ/sec$ . The blue line in Fig. 6.6 shows an example of the sensor saturation. In the figure, we find that the raw gyroscope readings in the Y-axis saturate from 0.52 seconds to 0.54 seconds. To deal with this problem, we apply the cubic spline interpolation [71] to interpolate the saturated gyroscope readings in the Y-axis. Specifically, we use the gyroscope readings in the Y-axis within the range of  $[t_{sat\_start} - tw, t_{sat\_start})$  and  $(t_{sat\_end}, t_{sat\_end} + tw]$  to construct new data points within the range of  $[t_{sat\_start}, t_{sat\_end}]$ .  $t_{sat\_start}$  and  $t_{sat\_end}$  are the start and end times of the saturation, which are 0.52 seconds and 0.54 seconds in this case.  $tw$  is empirically set to be 100ms. The gyroscope readings in the Y-axis after interpolation are shown as the red dashed line in the figure.



**Figure 6.6:** Interpolation of the gyroscope readings in the Y-axis

### 6.2.6 Stroke Classification

After data interpolation, the next step is to classify the stroke type for each stroke data segment. First, we extract a series of features from each data segment. The features include mean, standard deviation, skewness, kurtosis, minimum, and maximum of each axis of accelerometer and gyroscope readings. The amplitude of accelerometer and gyroscope readings are also computed. In total, 38 features for each data segment are calculated and normalized between  $[0, 1]$ . Second, we apply a machine learning classifier to classify each data segment into one of three stroke types: serve, groundstroke, and volley. We compare the performance of five classifiers: Naive Bayes, AdaBoost, Support Vector Machine (SVM), Decision Tree, Random Forest. Since the Random Forest performs best as shown in Section [6.3.4](#), we choose the Random Forest classifier for stroke classification.

### 6.2.7 Serve Speed Calculation

For a tennis serve, the initial speed of a ball is quite small and mainly depends on the racket speed. The larger the racket speed, the higher the ball speed. This motivates us

to use the racket speed to calculate the serve speed. When a player serves a tennis ball, the racket swings in a circular motion with the human shoulder as the center. Because the racket rotates around the Y-axis in our coordinate system,  $g_y$  measures the angular speed of the racket. We use  $g_y$  at the impact start time to model the ball serve speed  $v_{b\_serve}$  by a linear regression model as:

$$v_{b\_serve} = k \cdot |g_y(t_{start})| + b, \quad (6.1)$$

where  $k$  and  $b$  are the parameters of the linear regression model. All of the tennis serve data are used to train the serve model.  $k$  and  $b$  are calculated by using the method of least squares [86].  $k$  and  $b$  are 0.025 and 20.06, respectively.

### 6.2.8 Groundstroke/Volley Speed Calculation

We propose two models to calculate the ball speed: a physical model and a regression model. The physical model is built based on the physical impact between a racket and a ball, while the regression model applies a linear regression model to estimate the ball speed. For advanced players, they have correct stroke gestures, and similar gestures for the same type of the stroke. We propose a physical model to calculate the ball speed for them. For beginning players, it is hard to build the physical model due to two reasons. First, their stroke gestures are incorrect. With incorrect stroke gestures and terrible tennis skills, they often swing the racket in an awkward way, which is hard to build the physical model. Second, for the same type of the stroke, they may perform different gestures and swing the racket in different ways. The big variance in their stroke gestures reduces the accuracy of the physical model. Therefore, instead of building a physical model, we propose a simple regression model to calculate the ball speed for the beginning players.

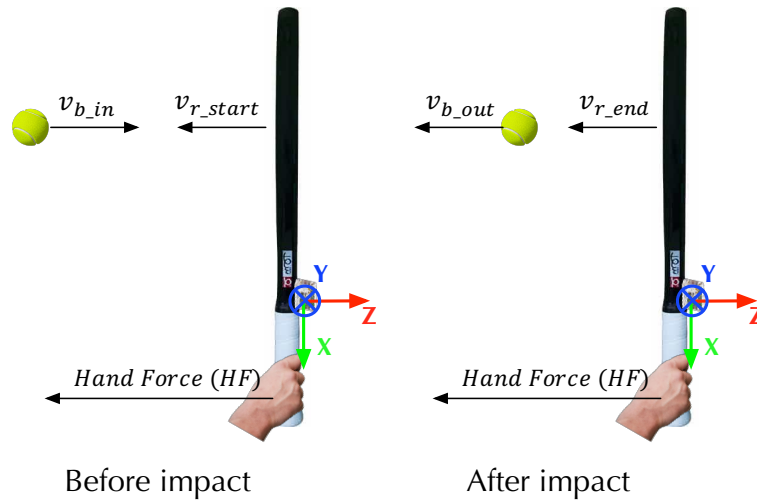
### 6.2.8.1 Physical Model

The impact between a racket and a tennis ball is governed by some physical laws and mechanical principles. By applying these laws and principles, we propose a physical model to estimate ball speed. The outgoing ball speed is mainly influenced by two factors: the racket speed and the incoming ball speed. The racket speed can be calculated using motion sensors, while the incoming ball speed is unknown. Without the incoming ball speed, the outgoing ball speed cannot be calculated by a physical model. To solve this problem, we apply a physical law (conservation of linear momentum [87]) and a mechanical principle (Coefficient of Restitution [88]). Both the physical law and the mechanical principle contain the incoming ball speed parameter. We combine these into one by eliminating the incoming ball speed parameter to get the outgoing ball speed.

The physical process of the impact between the ball and the racket is very complicated. With only one single racket-mounted sensor, it is hard to accurately describe the impact process. Therefore, we make several assumptions to simplify the physical model:

1. The tennis ball horizontally hits and bounces off the racket.
2. The ball impacts at the center of the racket face.
3. During impact, there is a constant hand force on the racket.

There are four reasons to these three assumptions. (1) The physical impact process between a racket and a tennis ball is very complicated. Without these assumptions, it is extremely difficult to build a physical model only using motion sensors. (2) Even if we build a model without these simplified assumptions, this model will be very complicated. A complicated model may require much more computation and energy cost than the simplified model. This is not feasible for the small-size device embedded into the racket. (3) As shown in Section 6.3, the evaluation results demonstrate that our proposed models perform quite well already. A complicated model may only increase the



**Figure 6.7:** Physical impact process between a tennis racket and a tennis ball

accuracy a little. (4) These assumptions are made based on our observations on tennis matches. We find that the angle between a ball and the normal to the racket string plane is typically quite small. Otherwise, the ball will either move towards the ground or move towards the sky. Accordingly, we assume that the ball horizontally hits and bounces off the racket. In addition, we find that tennis balls usually hit at the center of the rackets for advanced players. Therefore, we assume that the ball impacts at the center of the racket face. Finally, we find that the duration of the impact is roughly 20ms, as shown in Fig. 6.5. Thus, it is reasonable to assume a constant hand force during this short period of time.

Based on these assumptions, the physical impact process is shown in Fig. 6.7. Before impact, there is a hand force  $HF$  exerted on a racket. Under the influence of the hand force, the racket of mass  $M$  is moving at velocity  $v_{r\_start}$  towards a tennis ball. At the same time, this tennis ball of mass  $m$  is moving at velocity  $v_{b\_in}$  towards the racket. After impact, the tennis ball bounces off the racket at velocity  $v_{b\_out}$ , reducing the velocity of the racket to  $v_{r\_end}$ .

By conservation of linear momentum, we get:

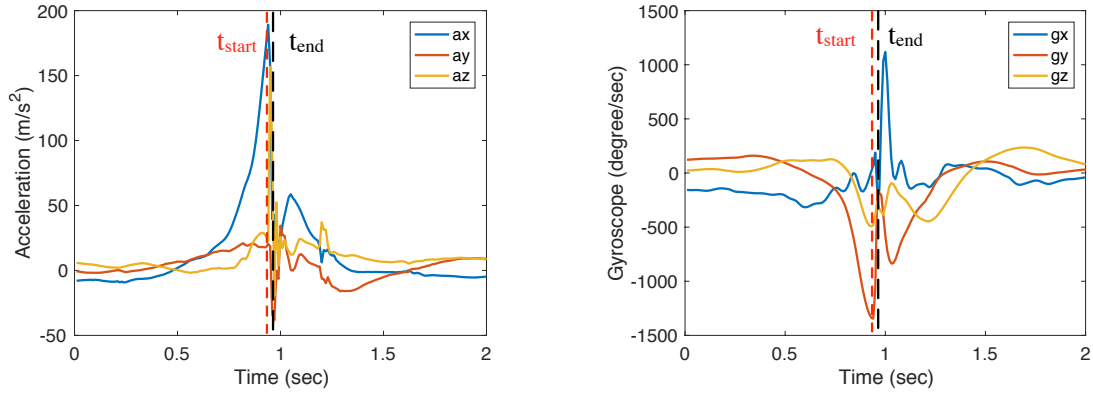


$$\int_{t_{start}}^{t_{end}} HF dt = M \cdot (v_{r\_end} - v_{r\_start}) + m \cdot (v_{b\_out} + v_{b\_in}), \quad (6.2)$$

where  $t_{start}$  and  $t_{end}$  are the start and end times of impact. In this equation,  $t_{start}$ ,  $M$ , and  $m$  are known.  $t_{start}$  is calculated in data segmentation module, as shown in Section 6.2.4.  $M$  and  $m$  are 300g and 50g. To calculate  $v_{b\_out}$ , we need to calculate the other unknown parameters first. They are  $t_{end}$ ,  $v_{r\_end} - v_{r\_start}$ ,  $HF$ , and  $v_{b\_in}$ .

**Calculating  $t_{end}$ .** The impact starts when a ball hits a racket, and ends when the ball leaves the racket. During the impact, due to the momentum lost by the ball, the speed of the racket decreases rapidly. After the ball leaves the racket, the speed of the racket increases again. By searching for the change of the racket speed, we find the end time of impact. More specifically, by searching from  $t_{start}$ ,  $t_{end}$  is calculated as:

$$t_{end} = t \quad \text{if } |g_y(t)| < |g_y(t+1)| \quad (6.3)$$



**Figure 6.8:** Acceleration and gyroscope data of a forehand shot

Fig. 6.8 shows the acceleration data (left figure) and gyroscope data (right figure) of a forehand shot performed by a left-hand player. The start time of impact  $t_{start}$  is marked as the red dashed line. The end time of impact  $t_{end}$  is marked as the black dashed line.

**Calculating  $v_{r\_end} - v_{r\_start}$ .**  $v_{r\_end} - v_{r\_start}$  can be calculated as the integration of the accelerometer readings in the Z-axis. However, as there are two sides of the rackets, the accelerometer readings in the Z-axis will have opposite values when the

tennis ball hits on the different sides. The motion sensor is attached on the side A of the racket, as shown in Fig. 6.2. When the side A of the racket hits the tennis ball, the movement direction of the racket is the same as the positive direction of the Z-axis of the sensor.  $v_{r\_end} - v_{r\_start}$  is calculated as the integration of the accelerometer readings in the Z-axis. However, when the side B of the racket hits the tennis ball, the movement direction of the racket is opposite to the positive direction of the Z-axis of the sensor, as shown in Fig. 6.7.  $v_{r\_end} - v_{r\_start}$  is calculated as the integration of the inverse of the accelerometer readings in the Z-axis. Therefore, to calculate the ball speed, we need to reorientate the motion sensor so that the positive direction of the Z-axis of the sensor is the same as the movement direction of the racket.

**Table 6.1:** Eight impact types between a tennis ball and a racket

<b>Dominant Hand</b>	<b>Forehand/ Backhand</b>	<b>Racket side that hits a ball</b>	$g_y(t_{start})$	$Direction_{Z-axis}$ and $Direction_{racket}$
Left	Forehand	Side A	$> 0$	Same
	Forehand	Side B	$< 0$	Opposite
	Backhand	Side A	$> 0$	Same
	Backhand	Side B	$< 0$	Opposite
Right	Forehand	Side A	$> 0$	Same
	Forehand	Side B	$< 0$	Opposite
	Backhand	Side A	$> 0$	Same
	Backhand	Side B	$< 0$	Opposite

There are eight possible impact types between a tennis ball and a racket, which are summarized in Table 6.1. From the table, we find that when the  $g_y(t_{start})$  is larger than 0, the positive direction of the Z-axis of the sensor is the same as the movement direction of the racket. When the  $g_y(t_{start})$  is smaller than 0, the positive direction of the Z-axis of the sensor is opposite to the movement direction of the racket. This motivates us to use  $g_y(t_{start})$  to reorientate the sensor readings in Z-axis. The reoriented accelerometer readings in the Z-axis  $a'_z$  is calculated as:

$$a'_z = \begin{cases} a_z & g_y(t_{start}) > 0 \\ -a_z & g_y(t_{start}) < 0 \end{cases} \quad (6.4)$$

The change of the racket speed  $v_{r\_end} - v_{r\_start}$  in Eq. 6.2 is calculated as the inte-

gration of the reoriented acceleration readings in the Z-axis during impact as:

$$v_{r\_end} - v_{r\_start} = \int_{t_{start}}^{t_{end}} a'_z(t) dt \quad (6.5)$$

**Calculating HF.** This hand force causes the racket to move in an accelerated motion, which is measured by the Z-axis of the accelerometer. As we assume that the hand force is constant during impact, we use the adjusted accelerometer readings in the Z-axis  $a'_z$  at the start time of impact  $t_{start}$  to model the hand force. The larger the hand force, the larger the acceleration. Thus, we choose a linear function to model this relationship as shown below:

$$HF = k_{HF} \cdot M \cdot a'_z(t_{start}) + b_{HF}, \quad (6.6)$$

where  $k_{HF}$  and  $b_{HF}$  are the model parameters. After substituting Eq. 6.3 through Eq. 6.6 into Eq. 6.2, Eq. 6.2 has four unknown parameters:  $k_{HF}, b_{HF}, v_{b\_in}, v_{b\_out}$ . We use the incoming ball speed  $v_{b\_in}$  and outgoing ball speed  $v_{b\_out}$  calculated from the PlaySight system to train this model. First, we get the incoming and outgoing ball speeds for each tennis shot from the PlaySight system. Then, we feed the incoming and outgoing ball speeds of all the tennis shots into Eq. 6.2. Finally, we use the method of least squares [36] to calculate  $k_{HF}$  and  $b_{HF}$ .  $k_{HF}$  and  $b_{HF}$  are 0.236 and 65.83, respectively.

**Calculating  $v_{b\_in}$ .** After  $k_{HF}$  and  $b_{HF}$  are determined, there are two unknown parameters in Eq. 6.2:  $v_{b\_in}$  and  $v_{b\_out}$ . To calculate  $v_{b\_out}$ , we need to calculate  $v_{b\_in}$  first. However,  $v_{b\_in}$  is hard to be calculated using a motion sensor. To solve this problem, we apply the coefficient of restitution (COR) to build another equation between  $v_{b\_in}$  and  $v_{b\_out}$ . We combine these two equations into one by eliminating  $v_{b\_in}$  parameter to calculate  $v_{b\_out}$ .

The COR is defined as the ratio of the final velocity to the initial velocity between two objects after their collision [38]. The COR is a measure of how much kinetic energy remains after the collision of two bodies, with the value that ranges from 0 to 1. If the COR is close to 1, it suggests that very little kinetic energy is lost during the collision;

on the other hand, if it is close to 0, it indicates that a large amount of kinetic energy is converted into heat or otherwise absorbed through deformation. In our case, the COR is calculated as:

$$e = \frac{v_{b\_out} - v_{r\_end}}{v_{r\_start} + v_{b\_in}} \quad (6.7)$$

In this equation, the racket speed before impact  $v_{r\_start}$  is calculated by the gyroscope readings in the Y-axis:

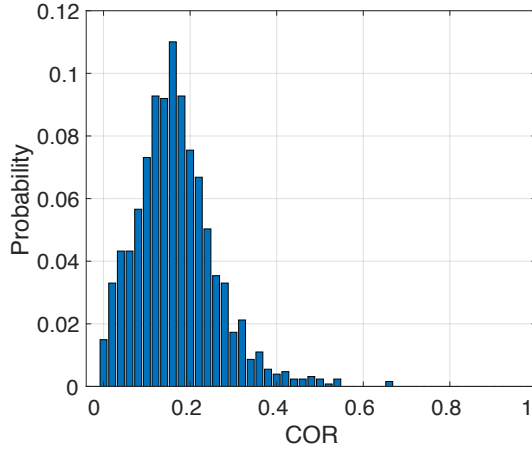
$$v_{r\_start} = |g_y(t_{start})| \cdot R_{swing}, \quad (6.8)$$

where  $R_{swing}$  is the swing radius of the circular motion, which is the distance between the center of the racket face and the rotation center. Howard Brody measured the swing radius among a number of players [89]. He found that the average radius between the butt end of the racket and the rotation center is 0.2m. As the distance between the center of the racket face and the butt end of the racket is 0.52m, as shown in Fig. 6.2. Therefore,  $R_{swing}$  is set to be 0.72m.

By substituting  $v_{r\_start}$  into Eq. 6.5, we calculate the racket speed after impact  $v_{r\_end}$  as:

$$v_{r\_end} = v_{r\_start} + \int_{t_{start}}^{t_{end}} a_z'(t) dt \quad (6.9)$$

After substituting Eq. 6.8 and Eq. 6.9 into Eq. 6.7, Eq. 6.7 has three unknown parameters:  $e$ ,  $v_{b\_in}$ ,  $v_{b\_out}$ . We feed the ball speeds calculated from the PlaySight system into Eq. 6.7 to calculate  $e$ . We feed the incoming ball speeds and outgoing ball speeds calculated from the PlaySight system into Eq. 6.7 to calculate  $e$ . The distribution of all calculated COR values is shown in Fig. 6.9. We find these COR values obey a normal distribution with a mean of 0.16. Because there are many impact factors for COR estimation (such as ball speed, stroke force, string tension and impact position) and the correlations between these impact factors and the COR are unknown, it is extremely difficult to explicitly define the COR model. In addition, most COR values are close to



**Figure 6.9:** Distribution of COR values

the mean of this normal distribution. Therefore, we assume COR to be a constant value, 0.16, for all the tennis shots to simplify the model.

We then combine the Eq. 6.9 and the Eq. 6.2 by eliminating the incoming ball speed to get the outgoing ball speed as:

$$v_{b\_out} = \frac{e}{1+e} \cdot \frac{1}{m} \left[ m \cdot v_{r\_start} - M \cdot (v_{r\_end} - v_{r\_start}) + \frac{m}{e} \cdot v_{r\_end} + \int_{t_{start}}^{t_{end}} HF dt \right] \quad (6.10)$$

From the above equation, we find that the outgoing ball speed  $v_{b\_out}$  depends on  $v_{r\_start}$ ,  $v_{r\_end}$ ,  $t_{start}$ ,  $t_{end}$ , and  $HF$ . The incoming ball speed  $v_{b\_in}$  does not appear in the equation as it is eliminated when we combine Eq. 6.2 and Eq. 6.7. Though  $v_{b\_in}$  does not appear in Eq. 6.10, it indirectly influences the outgoing ball speed via  $v_{r\_end}$ , which is calculated based on the integration of acceleration data in the Z-axis during impact as shown in Eq. 6.9. The larger incoming ball speed, the larger fluctuation of the acceleration data during impact. Therefore, the influence of the incoming ball speed is considered in our model.

### 6.2.8.2 Regression Model

When a player performs a groundstroke or volley, he/she swings the racket in a circular motion. In our coordinate system, the racket rotates around the Y-axis for both strokes. Similar to the serve speed model, we use  $g_y$  to calculate the outgoing ball speed  $v_{b\_out}$  by a linear polynomial model as:

$$v_{b\_out} = k \cdot |g_y(t_{start})| + b, \quad (6.11)$$

where  $k$  and  $b$  are the parameters of the linear regression model. We use all the tennis groundstroke and volley data to train this model by the method of least squares [86].  $k$  and  $b$  are 0.039 and 3.94, respectively.

## 6.3 Performance Evaluation

We evaluate the performance of TennisEye in this section. We first introduce the data set in Section 6.3.1. Then, we introduce the performance of TennisEye using leave-one-subject-out cross validation, self test, and 5-fold cross validation in Section 6.3.2.1, Section 6.3.2.2, and Section 6.3.2.3, respectively. We evaluate the influence factors on the physical model in Section 6.3.2.4. After that, we evaluate the performance of stroke detection in Section 6.3.3 and stroke classification in Section 6.3.4. Finally, we evaluate the overall performance of TennisEye in Section 6.3.5.

### 6.3.1 Data Set

We collected data from 7 players. Based on their self-report information, we divided the subjects into three categories: coach, regular player, and casual player. The coaches play several times per week, the regular players play one time per week, and the casual players play 0~2 times per month. Table 6.2 shows the summary of tennis data we collected with a UG sensor. In total, we collected 569 serves, 1398 groundstrokes, and 18 volleys.

**Table 6.2:** Dataset collected with a UG sensor

Player	Player Description	Dominant Hand	Data Collection Time	# Serve	# Groundstroke	# Volley
# 1	Male Coach	Right	5/8/2018	36	0	0
# 2	Female Coach	Left	5/15/2018; 5/22/2018	75	214	5
# 3	Female Coach	Right	8/13/2018	50	136	0
# 4	Regular Player	Right	5/23/2018	0	302	3
# 5	Casual Player	Left	7/7/2018; 7/15/2018; 7/22/2018	0	628	10
# 6	Casual Player	Right	5/7/2018; 8/13/2018; 9/7/2018	263	118	0
# 7	Casual Player	Right	9/7/2018	145	0	0

**Table 6.3:** Dataset collected with both a UG and a Zepp sensor

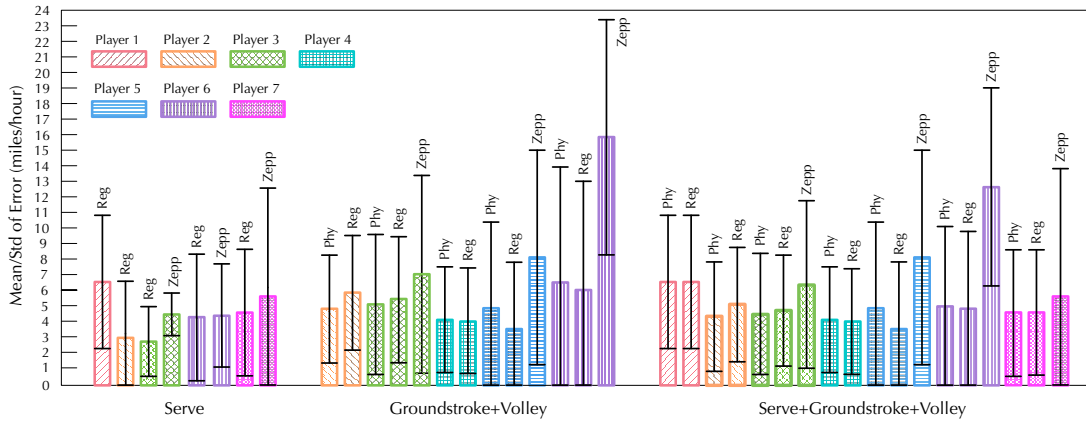
Player	Data Collection Time	# Serve	# Groundstroke	# Volley
# 3	8/13/2018	50	136	0
# 5	7/22/2018	0	209	6
# 6	9/7/2018	46	118	0
# 7	9/7/2018	145	0	0

No previous publication has used motion sensors to estimate tennis ball speed. However, we are aware that there are some tennis sensors on the market, such as Zepp [13], Sony Sensor [90], Usense [14], and Babolat Play [91]. These sensors are attached to the tennis racket. By analyzing the tennis data, they compute the key performance metrics for each swing, such as stroke type and ball speed. Among them, Sony Sensor and Usense are no longer manufactured. Babolat Play does not compute the ball speeds of groundstroke and volley shots, which we focus on. Zepp is a popular tennis sensor that recognizes stroke types and calculates serve, groundstroke and volley speeds. Therefore, we choose Zepp as state-of-the-art work. To compare with the ball speed calculation algorithm in Zepp, we collected tennis data using both a UG and Zepp sensor. Table 6.3 shows the summary of the tennis data collected.

## 6.3.2 Ball Speed Calculation Accuracy

### 6.3.2.1 Leave-one-subject-out Evaluation

To evaluate the performance of the proposed ball speed models. First, we use the video data to locate each tennis stroke in the dataset collected with a UG sensor as shown



**Figure 6.10:** Performance of ball speed estimation models under leave-one-subject-out cross validation

in Table 6.2. Then, we calculate the tennis ball speeds for each stroke based on the proposed ball speed models. Finally, we compare the ball speeds calculated by our model with the ball speeds calculated by other models. Here, we use the leave-one-subject-out cross validation to evaluate the performance of the proposed ball speed models. The leave-one-subject-out cross validation uses the tennis shot data from 6 subjects to train the ball speed estimation model, and then applies this model to estimate the ball speed from the remaining subject. Mean and standard deviation of error are considered as the evaluation metrics. The results are shown in Fig. 6.10. For the serve ball speed estimation, the error of the proposed regression model is  $4.3 \pm 4.0$  miles/hour. The accuracy is 93.4%. For the groundstroke and volley ball speed estimation, the error of the proposed physical model and regression model are  $5.0 \pm 4.9$  miles/hour and  $4.9 \pm 4.4$  miles/hour respectively. The accuracy are 88.4% and 88.8% respectively. For all the tennis shot data, the error of the proposed physical model and regression model are  $4.8 \pm 4.7$  miles/hour and  $4.7 \pm 4.3$  miles/hour, respectively. The accuracy are 90.0% and 90.2%, respectively. From Fig. 6.10, we find that the performance of the physical model is similar to that of the regression model. For the casual players, the regression model is slightly better. For the regular players, two models have similar performances. For the coaches, the physical model performs slightly better than the regression model.



Therefore, we use the regression model to estimate the ball speed for beginner players and the physical model to estimate the ball speed for advanced players.

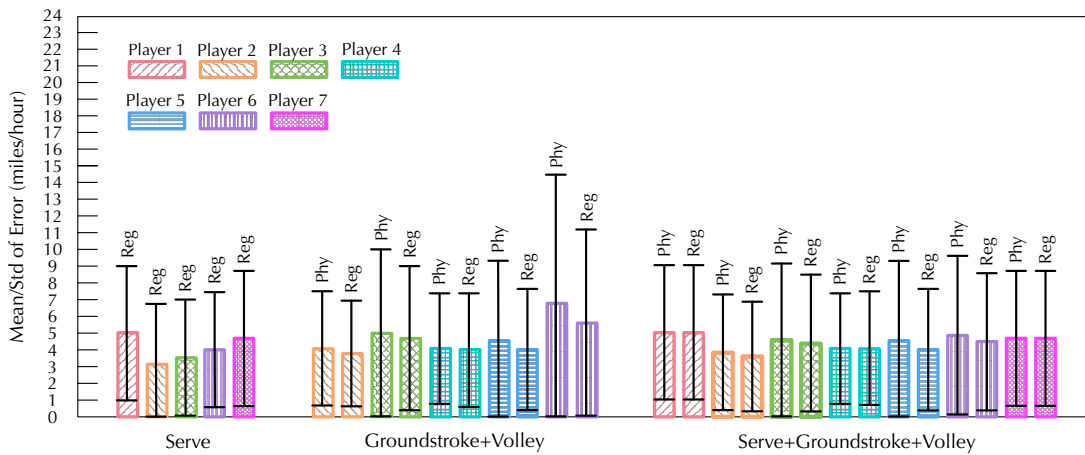
**Table 6.4:** Comparison with State-of-the-art Ball Speed Models

Ball Speed Model	Evaluation Metric	Serve	Groundstroke + Volley	Serve + Groundstroke + Volley
Peter et al. [83]	Mean±Std	11.6±7.1 miles/hour	7.6±6.9 miles/hour	9.1±7.3 miles/hour
	Accuracy	81.4%	81.2%	81.3%
Zepp	Mean±Std	5.2±5.7 miles/hour	10.6±miles/hour	8.9±7.8 miles/hour
	Accuracy	91.8%	70.8%	77.4%
Phy	Mean±Std	N/A	5.3±5.7 miles/hour	4.9±5.1 miles/hour
	Accuracy	N/A	86.7%	89.1%
Reg	Mean±Std	4.3±3.9 miles/hour	5.1±4.9 miles/hour	4.8±4.6 miles/hour
	Accuracy	93.1%	87.4%	89.5%

We compare our ball speed models with two works: the ball speed model in Zepp and a table tennis ball speed model [83]. There are two reasons to choose this table tennis ball speed model for comparison. (1) Both tennis and table tennis belong to racket sports. Compared with other racket sports, table tennis is most similar to tennis. (2) As far as we know, this table tennis model is the only motion sensors-based ball speed model for racket sports. We evaluate the performance of these three models using leave-one-subject-out (LOSO) cross validation. The dataset used for evaluation is collected with both a UG and Zepp sensor, as shown in Table 6.3. The evaluation results are shown in Table 6.4. From the table, we find that the proposed physical model and regression model have similar performance. Both of them perform much better than Zepp and the table tennis model. For all the tennis shot data, the proposed physical model is 7.8% more accurate than the table tennis model and 11.7% more accurate than Zepp. The proposed regression model is 8.2% more accurate than the table tennis model and 12.1% more accurate than Zepp. The table tennis model performs better than Zepp, especially in groundstroke and volley speed estimation. However, it is still outperformed by our models. We think there are two main reasons. (1) Table tennis and tennis are two racket sports that differ a lot in terms of the size, mass, and material of the ball and racket. In addition, the techniques of swinging the rackets are different. In table tennis, a player swings the racket with a lot of wrist action. However, in tennis, a player swings the racket with a lot of body rotation and less wrist action. Therefore, the

table tennis model may not be appropriate for tennis. (2) In this table tennis model, the racket speed is modeled as the sum of the wrist linear speed and wrist rotation speed. However, in tennis, the racket speed mainly depends on the rotation of the arm and body. Therefore, our models are more accurate than the table tennis model in characterizing the tennis racket speed.

### 6.3.2.2 Self Evaluation

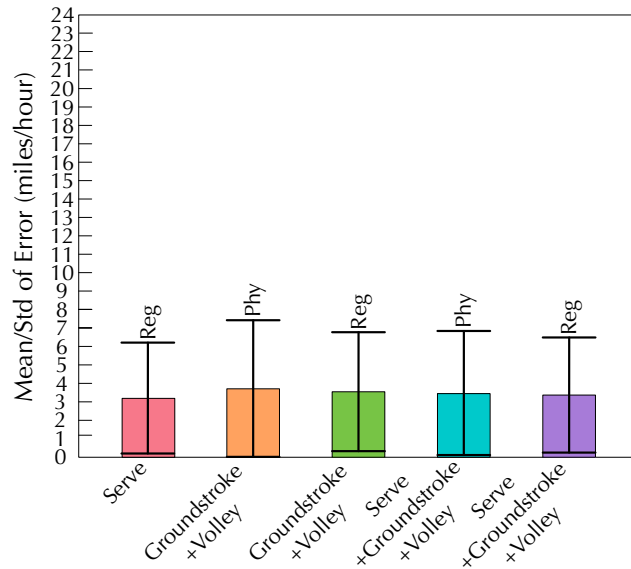


**Figure 6.11:** Performance of ball speed estimation models under self test

The self test partitions the dataset for each player into 5 randomly chosen subsets of equal size. Four subsets are used to train the model, and the remaining one is used to validate the model. This process is repeated 5 times, such that each subset is used exactly once for validation. The results of the self test are shown in Fig. 6.11. Mean and standard deviation of error are considered as the evaluation metrics. For the serve ball speed estimation, the error of the proposed regression model is  $4.0 \pm 3.6$  miles/hour. For the groundstroke and volley speed estimation, the error of the proposed physical model and regression model are  $4.6 \pm 4.6$  miles/hour and  $4.2 \pm 3.9$  miles/hour. For all the tennis shot data, the error of the proposed physical model and regression model are  $4.4 \pm 4.3$  miles/hour and  $4.2 \pm 3.8$  miles/hour. Under the self test, both the regression model and the physical model perform better than that under leave-one-subject-out cross validation.

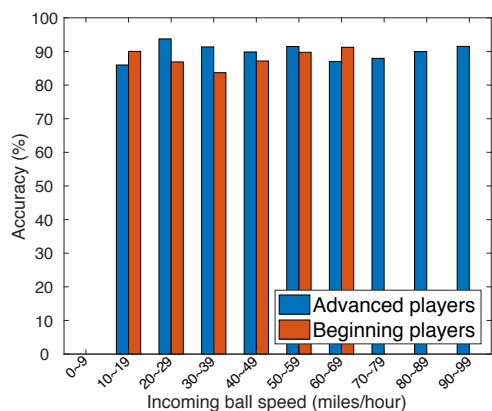
In addition, under the self test, the regression model always performs slightly better than the physical model for all the players.

### 6.3.2.3 Cross Evaluation

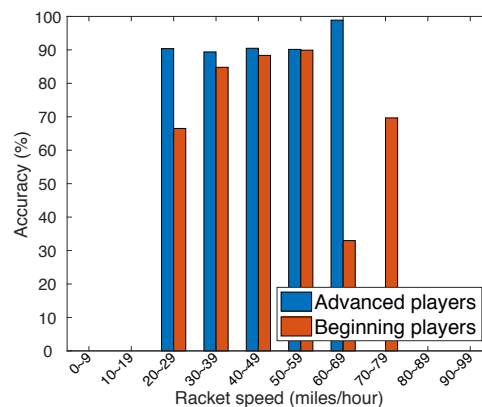


**Figure 6.12:** Performance of ball speed estimation models under 5-fold cross validation

The 5-fold cross-validation test uses all the data to form the dataset. It partitions the dataset into 5 randomly chosen subsets of equal size. Four subsets are used to train the model, and the remaining one is used to validate the model. This process is repeated 5 times such that each subset is used exactly once for validation. The results of the 5-fold cross validation are shown in Fig. 6.12. Mean and standard deviation of error are considered as the evaluation metrics. For the serve ball speed estimation, the error of the proposed regression model is  $4.1 \pm 3.8$  miles/hour. For the groundstroke and volley speed estimation, the error of the proposed physical model and regression model are  $4.8 \pm 4.8$  miles/hour and  $4.6 \pm 4.0$  miles/hour. For all the tennis shot data, the error of the proposed physical model and regression model are  $4.6 \pm 4.5$  miles/hour and  $4.4 \pm 3.9$  miles/hour. Under the 5-fold cross-validation test, both the regression model and the physical model perform better than that under leave-one-subject-out cross validation,



**Figure 6.13:** Performance of the physical model under different incoming ball speeds

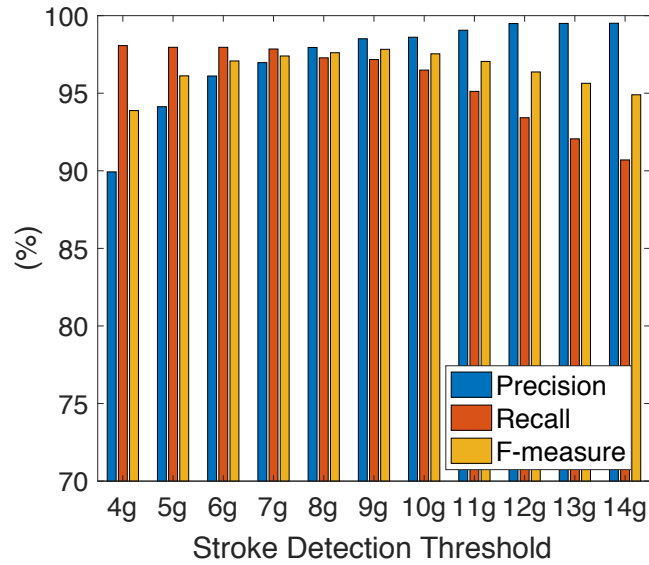


**Figure 6.14:** Performance of the physical model under different racket speeds

and worse than that under self test. In addition, under the 5-fold cross-validation test, the regression model always performs slightly better than the physical model.

#### 6.3.2.4 Influence Factors on the Physical Model

We evaluate the performance of the physical model under different incoming ball speeds and racket speeds using the leave-one-subject-out cross validation. The results are shown in Fig. 6.13 and Fig. 6.14. From Fig. 6.13, we find that the physical model performs well for both advanced and beginner players under various incoming ball speeds. Therefore, the incoming ball speed factor does not have significant influence on the accuracy of the physical model. From Fig. 6.14, we find that the physical model performs well for advanced players under various racket speeds. However, when the racket speed is too low ( $< 30$  miles/hour) or too high ( $\geq 60$  miles/hour), the physical model does not perform well for beginner players. The reduction of the accuracy results from the awkward and incorrect swing gestures performed by beginner players when they swing the racket too slow or too fast.



**Figure 6.15:** Performance of stroke detection under different thresholds

### 6.3.3 Stroke Detection Accuracy

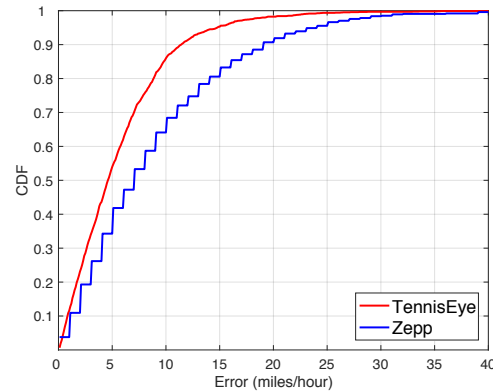
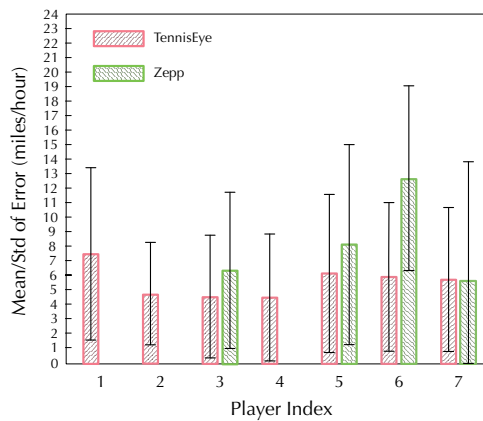
Fig. 6.15 shows the performance of the stroke detection under different thresholds. Three evaluation metrics are considered: precision, recall, and F-measure. F-measure considers the balance between precision and recall as described in Eq. 6.12:

$$F\text{-measure} = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (6.12)$$

From the figure, we find that as the threshold increases, the precision increases and the recall decreases. When the threshold is 9g, the F-measure is at highest value: 97.8%. Therefore, we choose 9g as the threshold for stroke detection. Under this threshold, the accuracy and recall are 98.5% and 97.2%. The promising results show the effectiveness of the proposed stroke detection method.

### 6.3.4 Stroke Classification Accuracy

We apply the WEKA machine-learning suite [56] to train five commonly used classifiers. The classifiers include AdaBoost (run for 100 iterations), Naive Bayes, SVM (with



**Figure 6.16:** Performance of TennisEye under leave-one-subject-out cross validation

**Figure 6.17:** Cumulative distribution functions of TennisEye and Zepp

polynomial kernels), Decision Tree (equivalent to C4.5 [57]), and Random Forests (100 trees, 4 random features each). Table 6.5 shows the performance of these five classifiers under 5-fold cross validation test. Same as stroke detection, precision, recall, and F-measure are considered as the evaluation metrics. From the table, we find that the Random Forest classifier performs the best. The precision, recall, and F-measure are 96.2%, 98.2%, and 97.2%, respectively. Therefore, we choose the Random Forest classifier for stroke classification.

**Table 6.5:** Comparison of Machine learning Algorithms for Stroke Classification

Algorithm	Precision	Recall	F-Measure
AdaBoost	92.1%	92.5%	92.3%
Naive Bayes	79.6%	71.6%	73.9%
SVM	92.1%	94.2%	92.9%
Decision Tree	83.4%	85.3%	84.4%
RandomForest	<b>96.2%</b>	<b>98.2%</b>	<b>97.2%</b>

### 6.3.5 TennisEye Performance

Fig. 6.16 shows the overall performance of TennisEye system and Zepp using the leave-one-subject-out cross validation. Mean and standard deviation of error are considered as the evaluation metrics. The cumulative distribution functions of TennisEye and Zepp

are shown in Fig. 6.17. The error of TennisEye is  $5.6 \pm 4.9$  miles/hour, while that of Zepp is  $8.9 \pm 7.8$  miles/hour. The accuracy of TennisEye is 88.2%, while that of Zepp is 77.4%. Therefore, TennisEye is 10.8% more accurate than Zepp in terms of the accuracy.

## 6.4 Discussion and Future Work

In our physical model, we make several assumptions. For example, we assume that the tennis ball horizontally hits and bounces off the racket. In addition, we assume that the ball impacts at the center of the racket face. However, these are not always the case. The tennis ball may hit the racket with different angles and different positions. In our study, we find that coaches always swing the racket horizontally and hit the tennis ball at the sweet spot of the racket, while the casual players often swing the racket at different angles and hit the tennis ball at bad positions, i.e., racket frame. For the casual players, the large variance of their tennis swings and bad hit positions significantly reduce the accuracy of the proposed physical model. Thus, the physical model does not perform well for the casual players. We plan to further improve the accuracy of the physical model by releasing these two assumptions.

In our study, we only consider the horizontal impact between a tennis racket and a tennis ball. However, the racket not only moves horizontally, but also vertically. During impact, the vertical momentum of the racket converts to the vertical speed and spin speed of the tennis ball. It would be interesting to explore how much momentum are converted to the vertical speed and how much momentum are converted to the spin speed. We plan to take the vertical speed of the racket into consideration to further improve the accuracy of the ball speed estimation model. In addition, we also plan to estimate ball spin speed in the future.

In addition, the impact process between a tennis ball and a tennis racket is very short. To accurately measure the racket vibration during the impact, a high sampling rate of the UG sensor is needed. The sampling rate in our system is set to be 100Hz. It

would be interesting to investigate whether the accuracy will be further improved or not with a higher sampling rate, which will be explored in the future.

## **6.5 Conclusion**

In this chapter, we propose TennisEye, a tennis ball speed calculation system using a racket mounted sensor. It detects tennis strokes, recognizes stroke types, and calculates the ball speed. We propose a regression model to estimate the serve speed. In addition, we propose two models, a regression model and a physical model, to estimate the groundstroke and volley ball speed for the beginning and the advanced player, respectively. For the leave-one-subject-out cross-validation test, experiments with human subjects show that the TennisEye is 10.8% more accurate than the state-of-the-art work. TennisEye is promising and has commercial potential as it is lightweight and more accurate than the existing commercial product.



## Chapter 7

# Conclusion and Future Work

In this dissertation, we investigate how to utilize motion sensors to study human behaviors. Specifically, we work on the following four topics:

First, we present Ultigesture, a wristband platform for motion sensing and human behavior study. We carefully design the hardware and the firmware of Ultigesture wristband. It is comfortable to wear and affordably priced. We provide a series of open APIs to Android developers. The developers can use our APIs to configure and access the data of the connected UG wristband through BLE. An Android library is carefully designed so that one smartphone can connect to multiple UG wristbands without conflict.

Second, 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 the HMM classification. Evaluation results show that the proposed algorithm can achieve over 94% recognition accuracy when users perform gestures continuously.

Third, we present MobiGesture, a mobility-aware gesture recognition system. We present a novel mobility-aware gesture segmentation algorithm to detect and segment hand gestures. In addition, we design a CNN model to classify hand gestures with mobility noises. Evaluation results show that the proposed CNN is 16.1%, 15.3%, and

14.4% more accurate than state-of-the-art work when the user is standing, walking and jogging, respectively. The proposed CNN is also two times faster than state-of-the-art work.

Finally, we propose TennisEye, a tennis ball speed estimation system using a racket-mounted sensor. It detects tennis strokes, recognizes stroke types, and calculates ball speed. We propose a regression model to estimate the serve speed. In addition, we propose two models, a regression model and a physical model, to estimate the ground-stroke and volley ball speed for beginning and advanced players, respectively. Evaluation results show that TennisEye is 10.8% more accurate than the state-of-the-art work.

For future work, we are considering the following four research directions:

- Our UG wristband only integrates accelerometer, gyroscope, and compass sensors for sensing. It would be interesting to add more sensors, such as a microphone and an infrared sensor. With more sensors integrated, a UG wristband will sense the human behaviors from different views.
- Rather than separately recognizing continuous hand gestures and hand gestures with body movement noises, it would be interesting to investigate the continuous hand gesture recognition with body movement noises. In some scenarios, a person tends to perform multiple hand gestures continuously while moving. For example, a person may like to use hand gestures to continuously control a toy car while walking with it. In addition, a person may like to continuously change music while jogging. In these scenarios, continuous hand gesture recognition with body movement noises is needed.
- Ball speed is an important factor in assessing the skill level of a player. In this dissertation, we use a racket-mounted sensor to estimate tennis ball speed. In future work, we plan to use motion sensors to estimate ball speed in other swing sports, such as badminton, golf, and baseball. We hope that our research can benefit players who cannot get access to the expensive Hawk-Eye system.

- Finally, we use motion sensors to study the human behaviors, including hand gestures and sports. In future work, we plan to use motion sensors to study other aspects of human behaviors, such as human activity recognition, person-to-person interaction and human-computer interaction. Our long-term goal is to use motion sensors to detect, recognize, and understand human behaviors.

# Bibliography

- [1] Alessandro Tognetti, Federico Lorusi, Raphael Bartalesi, Silvana Quaglini, Mario Tesconi, Giuseppe Zupone, and Danilo De Rossi. Wearable kinesthetic system for capturing and classifying upper limb gesture in post-stroke rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 2(1):8, 2005.
- [2] Hongyang Zhao, Shuangquan Wang, Gang Zhou, and Daqing Zhang. Gesture-enabled remote control for healthcare. In *Proceedings of the IEEE/ACM CHASE*, pages 392–401. IEEE, 2017.
- [3] Mashfiqui Rabbi, Min Hane Aung, Mi Zhang, and Tanzeem Choudhury. Mybehavior: automatic personalized health feedback from user behaviors and preferences using smartphones. In *Proceedings of the ACM Ubicomp*, pages 707–718. ACM, 2015.
- [4] Uwe Maurer, Anthony Rowe, Asim Smailagic, and Daniel P Siewiorek. ewatch: a wearable sensor and notification platform. In *Proceedings of the IEEE BSN*, pages 4–pp. IEEE, 2006.
- [5] Taiwoo Park, Jinwon Lee, Inseok Hwang, Chungkuk Yoo, Lama Nachman, and Junehwa Song. E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. In *Proceedings of the ACM SenSys*, pages 260–273. ACM, 2011.
- [6] E4 wristband. <https://www.empatica.com/e4-wristband>.

- [7] Simon W Davies, Sally L Jordan, and David 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):1581–1586, 1992.
- [8] Neela Harish and S Poonguzhali. Design and development of hand gesture recognition system for speech impaired people. In *Proceedings of the IEEE ICIC*, pages 1129–1133. IEEE, 2015.
- [9] Abhinav Parate, Meng-Chieh Chiu, Chaniel Chadowitz, Deepak Ganesan, and Evangelos Kalogerakis. Risq: Recognizing smoking gestures with inertial sensors on a wristband. In *Proceedings of the ACM MobiSys*, pages 149–161. ACM, 2014.
- [10] Hawk eye innovations. <http://www.hawkeyeinnovations.co.uk/sports/tennis>, 2018. Accessed: 2018-08-14.
- [11] NEIL Owens, C Harris, and C Stennett. Hawk-eye tennis system. In *Proceedings of the IET VIE*, pages 182–185. IET, 2003.
- [12] PlaySight. <http://playsight.com/>, 2018. Accessed: 2018-08-14.
- [13] Zepp. <http://www.zepp.com/tennis/>.
- [14] Usense. <http://www.ubc-tech.com/en/index.html>, 2018. Accessed: 2018-08-14.
- [15] Babolat. <http://www.babolat.us/>, 2018. Accessed: 2018-08-14.
- [16] Moto 360 2nd gen. <https://www.motorola.com/us/products/moto-360>.
- [17] Narayanan C Krishnan, Colin Juillard, Dirk Colbry, and Sethuraman Panchanathan. Recognition of hand movements using wearable accelerometers. *Journal of Ambient Intelligence and Smart Environments*, 1(2):143–155, 2009.
- [18] Yujie Dong, Adam Hoover, and Eric Muth. A device for detecting and counting bites of food taken by a person during eating. In *Proceedings of the IEEE BIBM*, pages 265–268. IEEE, 2009.

- [19] Holger Junker, Oliver Amft, Paul Lukowicz, and Gerhard Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, 2008.
- [20] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009.
- [21] Invensense mpu6050 datasheet. <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>.
- [22] Yujie Dong, Adam Hoover, Jenna Scisco, and Eric Muth. A new method for measuring meal intake in humans via automated wrist motion tracking. *Applied psychophysiology and biofeedback*, 37(3):205–215, 2012.
- [23] Lorenzo Porzi, Stefano Messelodi, Carla Mara Modena, and Elisa Ricci. A smart watch-based gesture recognition system for assisting people with visual impairments. In *Proceedings of the ACM IMMPD*, pages 19–24. ACM, 2013.
- [24] Chao Xu, Parth H Pathak, and Prasant Mohapatra. Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch. In *Proceedings of the ACM HotMobile*, pages 9–14. ACM, 2015.
- [25] Liwei Chan, Rong-Hao Liang, Ming-Chang Tsai, Kai-Yin Cheng, Chao-Huai Su, Mike Y Chen, Wen-Huang Cheng, and Bing-Yu Chen. Fingerpad: private and subtle interaction using fingertips. In *Proceedings of the ACM UIST*, pages 255–260. ACM, 2013.
- [26] Ke-Yu Chen, Kent Lyons, Sean White, and Shwetak Patel. utrack: 3d input using two magnetic sensors. In *Proceedings of the ACM UIST*, pages 237–244. ACM, 2013.
- [27] Ke-Yu Chen, Shwetak N Patel, and Sean Keller. Finexus: Tracking precise motions

- of multiple fingertips using magnetic sensing. In *Proceedings of the ACM CHI*, pages 1504–1514. ACM, 2016.
- [28] Wii controller. <http://wii.com/>.
- [29] Hyeon-Kyu Lee and Jin-Hyung Kim. An hmm-based threshold model approach for gesture recognition. *IEEE Transactions on pattern analysis and machine intelligence*, 21(10):961–973, 1999.
- [30] Kazuya Murao and Tsutomu Terada. A recognition method for combined activities with accelerometers. In *Proceedings of the ACM UbiComp*, pages 787–796. ACM, 2014.
- [31] Fei Yan, W Christmas, and Josef Kittler. A tennis ball tracking algorithm for automatic annotation of tennis match. In *Proceedings of the BMVC*, volume 2, pages 619–628, 2005.
- [32] Tayeba Qazi, Prerana Mukherjee, Siddharth Srivastava, Brejesh Lall, and Nathi Ram Chauhan. Automated ball tracking in tennis videos. In *Proceedings of the IEEE ICIP*, pages 236–240. IEEE, 2015.
- [33] Qizhi Wang, KangJie Zhang, and Dengdian Wang. The trajectory prediction and analysis of spinning ball for a table tennis robot application. In *Proceedings of the IEEE CYBER*, pages 496–501. IEEE, 2014.
- [34] Gopal Pingali, Agata Opalach, and Yves Jean. Ball tracking and virtual replays for innovative tennis broadcasts. In *Proceedings of the IEEE Pattern Recognition*, volume 4, pages 152–156. IEEE, 2000.
- [35] Xinyu Wei, Patrick Lucey, Stuart Morgan, and Sridha Sridharan. Predicting shot locations in tennis using spatiotemporal data. In *Proceedings of the IEEE DICTA*, pages 1–8. IEEE, 2013.

- [36] Xinyu Wei, Patrick Lucey, Stuart Morgan, and Sridha Sridharan. Sweet-spot: Using spatiotemporal data to discover and predict shots in tennis. In *Proceedings of the Annual MIT Sloan Sports Analytics Conference*, 2013.
- [37] Xinyu Wei, Patrick Lucey, Stephen Vidas, Stuart Morgan, and Sridha Sridharan. Forecasting events using an augmented hidden conditional random field. In *Proceedings of the Springer ACCV*, pages 569–582. Springer, 2014.
- [38] David Whiteside, Olivia Cant, Molly Connolly, and Machar Reid. Monitoring hitting load in tennis using inertial sensors and machine learning. *International journal of sports physiology and performance*, 12(9):1212–1217, 2017.
- [39] Weiping Pei, Jun Wang, Xubin Xu, Zhengwei Wu, and Xiaorong Du. An embedded 6-axis sensor based recognition for tennis stroke. In *Proceedings of the IEEE ICCE*, pages 55–58. IEEE, 2017.
- [40] Lars Bütke, Ulf Blanke, Haralds Capkevics, and Gerhard Tröster. A wearable sensing system for timing analysis in tennis. In *Proceedings of the IEEE BSN*, pages 43–48. IEEE, 2016.
- [41] Miha Mlakar and Mitja Luštrek. Analyzing tennis game through sensor data with machine learning and multi-objective optimization. In *Proceedings of the ACM ISWC*, pages 153–156. ACM, 2017.
- [42] Akash Anand, Manish Sharma, Rupika Srivastava, Lakshmi Kaligounder, and Divya Prakash. Wearable motion sensor based analysis of swing sports. In *Proceedings of the IEEE ICMLA*, pages 261–267. IEEE, 2017.
- [43] Rupika Srivastava, Ayush Patwari, Sunil Kumar, Gaurav Mishra, Lakshmi Kaligounder, and Purnendu Sinha. Efficient characterization of tennis shots and game analysis using wearable sensors data. In *Proceedings of the IEEE SENSORS*, pages 1–4. IEEE, 2015.



- [44] Disheng Yang, Jian Tang, Yang Huang, Chao Xu, Jinyang Li, Liang Hu, Guobin Shen, Chieh-Jan Mike Liang, and Hengchang Liu. Tennismaster: an imu-based on-line serve performance evaluation system. In *Proceedings of the ACM AH*, page 17. ACM, 2017.
- [45] Manish Sharma, Rupika Srivastava, Akash Anand, Divya Prakash, and Lakshmi Kaligounder. Wearable motion sensor based phasic analysis of tennis serve for performance feedback. In *Proceedings of the IEEE ICASSP*, pages 5945–5949. IEEE, 2017.
- [46] UG smart wristband. <http://www.ultigesture.com/>.
- [47] Amit A Levy, James Hong, Laurynas Riliskis, Philip Levis, and Keith Winstein. Beetle: Flexible communication for bluetooth low energy. In *Proceedings of the ACM MobiSys*, pages 111–122. ACM, 2016.
- [48] Marketsandmarkets, 2015. <http://www.marketsandmarkets.com/Market-Reports/touchless-sensing-gesturing-market-369.html>.
- [49] Juan P Wachs, Helman I Stern, Yael Edan, Michael Gillam, Jon Handler, Craig Feied, and Mark Smith. A gesture-based tool for sterile browsing of radiology images. *Journal of the American Medical Informatics Association*, 15(3):321–323, 2008.
- [50] Milena Milenkovic, Emil Jovanov, John Chapman, Dejan Raskovic, and John Price. An accelerometer-based physical rehabilitation system. In *Proceedings of the IEEE SSST*, pages 57–60. IEEE, 2002.
- [51] Christopher Lee and Yangsheng Xu. Online, interactive learning of gestures for human/robot interfaces. In *Proceedings of the IEEE ICRA*, volume 4, pages 2982–2987. IEEE, 1996.
- [52] Won-Chul Bang, Wook Chang, Kyeong-Ho Kang, Eun-Seok Choi, Alexey Potanin,

- and Dong-Yoon Kim. Self-contained spatial input device for wearable computers. In *Proceedings of the IEEE ISWC*, page 26. IEEE Computer Society, 2003.
- [53] Ari Y Benbasat and Joseph A Paradiso. An inertial measurement framework for gesture recognition and applications. In *International Gesture Workshop*, pages 9–20. Springer, 2001.
- [54] Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.
- [55] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.
- [56] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *Proceedings of the ACM SIGKDD*, 11(1):10–18, 2009.
- [57] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.
- [58] Maureen C Ashe, William C Miller, Janice J Eng, and Luc Noreau. Older adults, chronic disease and leisure-time physical activity. *Gerontology*, 55(1):64–72, 2009.
- [59] I-Min Lee and David M Buchner. The importance of walking to public health. *Medicine and science in sports and exercise*, 40(7 Suppl):S512–8, 2008.
- [60] Thomas Prohaska, Elaine Belansky, Basia Belza, David Buchner, Victor Marshall, Kathleen McTigue, William Satariano, and Sara Wilcox. Physical activity, public health, and aging: critical issues and research priorities. *The Journals of Gerontology Series B: Psychological Sciences and Social Sciences*, 61(5):S267–S273, 2006.

- [61] Eleanor M Simonsick, Jack M Guralnik, Stefano Volpato, Jennifer Balfour, and Linda P Fried. Just get out the door! importance of walking outside the home for maintaining mobility: findings from the women's health and aging study. *Journal of the American Geriatrics Society*, 53(2):198–203, 2005.
- [62] Susan E Hardy, Yihuang Kang, Stephanie A Studenski, and Howard B Degenholtz. Ability to walk 1/4 mile predicts subsequent disability, mortality, and health care costs. *Journal of general internal medicine*, 26(2):130–135, 2011.
- [63] Global recommendations on physical activity for health. world health organization. [http://www.who.int/dietphysicalactivity/factsheet\\_recommendations/en](http://www.who.int/dietphysicalactivity/factsheet_recommendations/en).
- [64] Physical activity guidelines for americans. u.s. department of health and human services. <http://health.gov/paguidelines>.
- [65] Nike+ run club app. [https://www.nike.com/us/en\\_us/c/nike-plus/running-app-gps](https://www.nike.com/us/en_us/c/nike-plus/running-app-gps).
- [66] Runkeeper app. <https://runkeeper.com/>.
- [67] Mapmyrun app. <http://www.mapmyrun.com/>.
- [68] Andrew J Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *The Foundations Of The Digital Wireless World: Selected Works of AJ Viterbi*, pages 41–50. World Scientific, 2010.
- [69] Hongyang Zhao, Shuangquan Wang, Gang Zhou, and Daqing Zhang. Ultigesture: A wristband-based platform for continuous gesture control in healthcare. *Smart Health*, 2018.
- [70] Peter Alfeld. A trivariate cloughâtocher scheme for tetrahedral data. *Computer Aided Geometric Design*, 1(2):169–181, 1984.
- [71] Carl De Boor, Carl De Boor, Etats-Unis Mathématicien, Carl De Boor, and Carl De Boor. *A practical guide to splines*, volume 27. Springer, 1978.

- [72] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [73] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the Springer COMPSTAT*, pages 177–186. Springer, 2010.
- [74] Frank Mokaya, Roland Lucas, Hae Young Noh, and Pei Zhang. Burnout: a wearable system for unobtrusive skeletal muscle fatigue estimation. In *Proceedings of the ACM/IEEE IPSN*, pages 1–12. IEEE, 2016.
- [75] Biyi Fang, Nicholas D Lane, Mi Zhang, and Fahim Kawsar. Headscan: A wearable system for radio-based sensing of head and mouth-related activities. In *Proceedings of the ACM/IEEE IPSN*, pages 1–12. IEEE, 2016.
- [76] Tian Hao, Guoliang Xing, and Gang Zhou. Runbuddy: a smartphone system for running rhythm monitoring. In *Proceedings of the ACM Ubicomp*, pages 133–144. ACM, 2015.
- [77] Andreas Möller, Luis Roalter, Stefan Diewald, Johannes Scherr, Matthias Kranz, Nils Hammerla, Patrick Olivier, and Thomas Plötz. Gymskill: A personal trainer for physical exercises. In *Proceedings of the IEEE PerCom*, pages 213–220. IEEE, 2012.
- [78] Amin Ahmadi, Edmond Mitchell, Francois Destelle, Marc Gowing, Noel E O’Connor, Chris Richter, and Kieran Moran. Automatic activity classification and movement assessment during a sports training session using wearable inertial sensors. In *Proceedings of the IEEE BSN*, pages 98–103. IEEE, 2014.
- [79] Benjamin H Groh, Frank Warschun, Martin Deininger, Thomas Kautz, Christine Martindale, and Bjoern M Eskofier. Automated ski velocity and jump length determination in ski jumping based on unobtrusive and wearable sensors. *Proceedings of the ACM IMWUT*, 1(3):53, 2017.

- [80] Cassim Ladha, Nils Y Hammerla, Patrick Olivier, and Thomas Plötz. Climbox: skill assessment for climbing enthusiasts. In *Proceedings of the ACM Ubicomp*, pages 235–244. ACM, 2013.
- [81] Aftab Khan, James Nicholson, and Thomas Plötz. Activity recognition for quality assessment of batting shots in cricket using a hierarchical representation. *Proceedings of the ACM IMMUT*, 1(3):62, 2017.
- [82] Bo Zhou, Harald Koerger, Markus Wirth, Constantin Zwick, Christine Martindale, Heber Cruz, Bjoern Eskofier, and Paul Lukowicz. Smart soccer shoe: monitoring foot-ball interaction with shoe integrated textile pressure sensor matrix. In *Proceedings of the ACM ISWC*, pages 64–71. ACM, 2016.
- [83] Peter Blank, Benjamin H Groh, and Bjoern M Eskofier. Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor. In *Proceedings of the ACM ISWC*, pages 2–9. ACM, 2017.
- [84] Industrial Wearable Devices Market. [https://www.researchandmarkets.com/research/nrtbv9/global\\_industrial?w=4](https://www.researchandmarkets.com/research/nrtbv9/global_industrial?w=4), 2018. Accessed: 2018-08-14.
- [85] Global Wearable Devices in Sports Market. [https://www.researchandmarkets.com/research/k8p2gz/global\\_wearable?w=4](https://www.researchandmarkets.com/research/k8p2gz/global_wearable?w=4), 2018. Accessed: 2018-08-14.
- [86] Least squares. [https://en.wikipedia.org/wiki/Least\\_squares](https://en.wikipedia.org/wiki/Least_squares), 2018. Accessed: 2018-09-23.
- [87] Conservation of Linear Momentum. <https://en.wikipedia.org/wiki/Momentum>, 2018. Accessed: 2018-09-23.
- [88] Graham Weir and Peter McGavin. The coefficient of restitution for the idealized impact of a spherical, nano-scale particle on a rigid plane. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 464, pages 1295–1307. The Royal Society, 2008.

- [89] Howard Brody. The physics of tennis. iii. the ball–racket interaction. *American Journal of Physics*, 65(10):981–987, 1997.
- [90] Sony. <https://www.sony.com.au/microsite/tennis/>, 2019.
- [91] Babolat Play. <http://en.babolatplay.com/play>, 2019.

## VITA

### Hongyang Zhao

Hongyang Zhao has been working on his Ph.D. degree in the Department of Computer Science at the College of William and Mary since Fall 2014. He is working with Dr. Gang Zhou in the fields of gesture recognition and ubiquitous computing. Hongyang Zhao got his M.S. in 2014 from Zhejiang University, China, and B.S. in 2011 from Shanghai Jiaotong University, China.