NOVEL GESTURES FOR WEARABLES

A Dissertation
Presented to
The Academic Faculty

By

Cheng Zhang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

May 2018

NOVEL GESTURES FOR WEARABLES

Approved by:

Dr. Gregory D. Abowd, Co-Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Omer Inan, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Thad E. Starner
School of Interactive Computing
Georgia Institute of Technology

Dr. Chris Harrison
Human Computer Interaction Institute
Carnegie Mellon University

Dr. Thomas Ploetz
School of Interactive Computing
Georgia Institute of Technology

Date Approved: March 12, 2018

A great quote to start the thesis

George P. Burdell

A great dedication goes here.

ACKNOWLEDGEMENTS

First and foremost, I am very grateful to have a great family who is always there for me. They support me when I fail and cheer for me when I make any progress. Their unconditional love spoils me and drives me through the whole Ph.D. journey. I can not come to this point without them. I love them and will love forever.

I would like to thank my advisors, Gregory Abowd and Omer Inan, who have given me tremendous support and trust during the past few years. Their generous sharing of both their professional and personal experience helped me overcome the frustration I encountered and grow up as a Ph.D. student, a researcher, and a person. I also want to thank Rosa Arriaga, who was my co-advisor in the first two years. She helped me to adjust myself to the completely new environment when I first came to GT and prepared me to be an independent researcher.

I am extremely lucky to have my all-star thesis committee, including Gregory Abowd, Omer Inan, Thad Starner, Thomas Ploetz, and Chris Harrison. Each of them is the leader in their fields. Their generous support not only helps me find the right direction for my thesis work and support my career development, but more importantly, they all set the role models for my future career.

I also would like to thank the support from all the fellow labmates in the Ubicomp group, the staff in the school, and the talented students who I felt honored to work with. Without their help and friendship, I can not achieve what I have done.

Finally, I want to thank the dozens of participants in my user studies. These study were not all interesting and enjoyable. Many were boring. Without their generous sharing of time, I can not thoroughly evaluate my projects and published the results.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Wearable computing is an inevitable part of the next generation of computing [1]. Compared with traditional computers (e.g., laptop, smartphones), wearable devices are much smaller, creating new challenges for the design of both hardware and software. Providing appropriate input capabilities for wearables is one such challenge. The input techniques that have been proven efficient on traditional computing devices (e.g., keyboard, touchscreen) are no longer appropriate for wearables due to various reasons. One is the inherently small size of wearables. For instance, it is impossible to place a physical keyboard on a wearable device. Most of the commodity wearable devices such as the smartwatch and Google Glass adopt a touch-based input solution, which suffers from the small operation area as well as the limited richness of input vocabulary. The other reason is the more dynamic working environment the wearables are exposed to. For instance, wearable devices are expected to be functional and efficient even when the user is in motion (e.g., walking). Traditional input devices are no longer appropriate to address these challenges. Compared with input on the physical keyboard or touchscreen, gesture-based input provides the user with much larger freedom of operation, which can potentially improve the interaction experience.

In this thesis, I explore designing and implementing various novel gestures to address the input challenges on wearables: from using the built-in sensors of an off-the-shelf device to building customized hardware; from 2D on-body interaction to 3D input with a larger freedom of operation; from recognizing predefined and discrete hand or finger gestures with machine learning to providing continuous input tracking through a deeper understanding of physics.

I start with exploring the natural and novel input gestures that can be supported by using only the built-in sensors of a smartwatch. I describe WatchOut and TapSkin, which allow user input on the watch case, band, and the skin around the watch. Though using only the built-in sensors is more practical, the richness of input gestures and performance

of recognition are not optimized because of the limited choice of sensors.

To better address the input challenge of wearable, I designed and implemented another sets of wearable input techniques using customized hardware (e.g., a thumb-mounted ring), which provides new input gestures for wearable that are not available on a commodity device, such as, input number digits, Graffiti-style characters, menu selection and quick response with the protection of users' privacy. However, these complementary gestures can only partially improve the interaction experience. To fundamentally address the input challenge on wearables, new interaction paradigms are needed to replace the touch-based on-device interaction.

Such an interaction paradigm is usually comprised of various low-level input events of high resolution. For instance, the press and release of the mouse keys are the low-level input events for the WIMP interface. This leads to the last section of my dissertation work: providing low-level input events for wearables, such as continuously tracking of the position of different body parts of interest in 3D space using wearables. I also demonstrate how such low-level input event can be used to design interaction on wearables.

CHAPTER 1
INTRODUCTION AND MOTIVATION

We are entering a new era of computing [1] , where wearable computing is an inevitable part. Researchers have explored wearable computing as a research topic for decades. However, due to the hardware limitation, wearables as commercial products started appearing on the market in the past few years

Compared with traditional computers, wearable devices require each hardware part to be small and light enough, such that a user can wear it all day long without suffering much discomfort. However, due to the hardware limitation, the early prototypes usually look cumbersome, which keep them away from the major consumer market. In the past five years, because of the technology advancement (e.g., battery, display), the size and weight are no longer a problem for the wearable computers. The industry is able to produce every hardware part in a much smaller size with comparable functionality and performance compared with traditional computers. Thus, we have seen various wearable computing devices appear in the consumer market. Google Glass[1] and smart watches are the most representative ones. However, these first generation of consumer wearable devices do not succeed as the initial high attention indicated. Google Glass Explorer Program was aborted as a public project in 2015, and the growth of smartwatch shipments failed to match the previous expectations in 2016. The current slowdown of growth on wearable devices can be attributed to several to-be-addressed challenges, all of which are related to user experience. The most obvious one is the limited capacity of the battery. Many wearable devices may run out of battery in the middle of a day, and the user has to take off the device to recharge it, which disrupt the always-on experience. The other equally important issue is the lack of satisfying interaction experience. Simply replicating the proven successful interaction

[1] https://www.x.company/glass/

solution (e.g. touchscreen, QWERTY Keyboard) on traditional computing devices (e.g., laptop, smartphone) to wearable devices has failed to provide satisfying user interaction experience. Because most wearable devices such as the smartwatch, by necessity, are relatively small compared to traditional computing devices. The freedom of operation for both input and output are very much limited, which prevent these interaction solution to be fully functional as it is on other traditional computing devices. For instance, a touchscreen can be used for text input on a smartphone but not on a smart watch.

If history can be of any help on predicting the future, providing appropriate interaction interface can be the last development milestone before the wearable devices can truly be widely popularized. We have seen similar story developed on the traditional personal computers. They were born with giant size and requires tedious process to interact with. It did not succeed as personal computers until the appearance of the graphical user interface, the keyboard and mouse. Wearable computers demands their own interaction paradigm.

In this thesis, I presented the design and implementation of various input gestures to improve the wearable interaction experience.

1.1 Thesis Statement and Research Questions

I propose the following thesis statement:

The inherently small size presents a challenge for providing appropriate input capabilities for wearables, which can be partially addressed if not all, by providing various high-level interaction gestures, novel low-level input event, and redesign of the interaction.

1.1.1 Research Questions

In particular, I addressed the following research questions.

RQ1

What novel input gestures can be developed to improve the interaction experience on wearables by only using the sensors on off-the-shelf wearable devices.

To answer this question, I developed two techniques that allows the user to input on the watch case, band, and the skin around the watch.

RQ2

What novel gestures can be developed to address the current input challenges on wearables using customized hardware?

I designed and implemented another sets of wearable input techniques using customized hardware (E.g., a thumb-mounted ring, wrist-mounted device), which provides new input gestures for wearable that are not available on a commodity device, such as, one-handed input for number digits and text, hand poses recognition (E.g., Sign language), menu selection and quick response with the protection of users' privacy.

RQ3

What kinds of low-level input events can be developed using novel sensing modalities? How would such new input events being used for wearable interaction?

To answer this question, I developed two new interaction techniques that can continuously track the position of different body parts in 3D space. I also demonstrate novel interactions that are supported by these low-level input events.

CHAPTER 2
RELATED WORK

I present the related works in three sections according to the proposed research questions. In the first section, I review the existing input technology for wearables, which provide various sets of input gestures and alternative input techniques. However, these techniques usually can only be used as alternative input for wearables, due to the limited richness of input vocabulary. To thoroughly improve the input experience on wearables, more fundamental input technology is needed instead of simply recognizing pre-defined gestures. Allowing the user to input in 3D space on wearable is one potential solution. The first step is to continuously track the finger position on the wearable device. Therefore, I review the techniques on continuously tracking finger position in the second section. In the last section, I discuss works on using the human-body as a sensing medium for interaction(e.g., communication, input).

2.1 Input technique for wearables

The wearable computing community has worked on providing appropriate input technology for tiny portable computers for years. These projects usually instrument different parts of the body with additional hardware to provide discrete input gestures.

2.1.1 Input with an arm/hand device device

Many novel wearable input technologies are based on some form of armbands to facilitate user input. Various sensing modalities have been explored to capture signals on arms, such as acoustic signals[2, 3] and Electromyography(EMG) signals [4]. Even though these armbands provide a rich set of input functions, the user may find it inconvenient and socially awkward to wear these devices during daily activities.

Different from an armband, it is easier to convince a user to wear a wrist-mounted devices, since people have already worn watches for years. Therefore, many projects built wrist-mounted devices for input. These devices were embedded with different sensing modalities to recognize finger or hand gestures, such as force sensing [5], acoustic sensing [6, 7, 3], electrical sensing [8], electromagnetic sensing [9] static electric field sensing [10], proximity sensing [11], camera [12] and motion sensing [13]. The input vocabularies for these are all relatively small, such that text input is not supported. The Twiddler [14] allows the user to input text by wearing a keyboard in hand. However, wearing a device in hand may be cumbersome in daily activities. Moreover, it has a steeper learning curve, so much so that a user needs to spent over 20 hours to learn how to effectively use the device for input [14].

Others designed devices which can be worn on the hand such as a data glove [15].

2.1.2 Input with a ring

Many projects have explored building rings for input, as the rings are relatively small and light. A ring can capture the finger movement by applying appropriate sensing modalities. Cameras have been used extensively to build a ring that tracks user input through hand gestures. For example, the Digits system [16] tracks hand poses using a wrist mounted camera. PinchWatch [12] operates similarly through utilizing a chest worn camera, while CyclopsRing [17] uses a thumb worn camera.

Alternatively, inertial measurement units (IMUs) have been used in ThumbRing [18] for the recognition of gestures whereas magnetic sensing was used in DigitSpace [19] and uTrack [20] for capture the finger movement in a 3D space. Others were built to support finger interaction on surfaces [8, 21] or interaction with other objects [22]. Subtle finger movements can also be detected with a ring made of printed electrodes [23].

2.1.3 Input on Smartwatch

As wrist-mounted devices (e.g., health activity monitors and smartwatches) dominate the wearable device market, more interest has been raised on improving the interactions with them, especially the input on the smartwatch. HCI researchers have explored how to improve the user experience in spite of the inherent limits of the small touchscreen and form factor.

Approaches include increasing the size of the screen area [24], reducing the size of the touch area by using a stylus on the finger nail [25], or by applying carefully designed touchscreen gestures [26, 27]. Another approach is to make other parts of the watch interactive [28, 29], including turning the band to an input surface [30, 31], or using an additional arm band [32]. Others have considered extending the interaction area to a larger space around the device; for instance, recognizing gestures in the space above using IR sensors or acoustic sensing [33, 34], around [35, 36, 37, 38] the watch or expand the perception space by using dynamic peephole[39]. More recent works enrich the input vocabulary by recognizing a set of specific designed finger/hand gestures [40, 41, 42, 26, 13].

Most of these solutions require either some additional sensing on the device or a completely new device to be worn by the user. Some recent work has shown the possibility of detecting the taps on the side and back of a phone with built-in sensors [43, 44, 45] . However, the shape, the worn position, and the size of a watch is different from a smartphone. Therefore, how to design the gesture families and situate them into the watch applications are the new challenges. In chapter three, I will present my works that examines the novel input gestures that can be recognized using only the built-in sensors (e.g., inertial sensing unit, acoustic sensing) of a off-the-shelf smartwatch.

2.2 Tracking fingers in 3D space

Continuously tracking the position of one or more fingers in 3D space allows a much larger interaction space and potentially richer input vocabulary for wearable input. This technical challenge is relatively new as a research topic in HCI community. However, localizing objects in 3D space has been a well-established research topic for years.

2.2.1 Object localization using acoustic signals or radio frequency signals

Acoustics has been widely used for distance calculation for many years, especially for indoor localization. Most of these technologies actively send a chirp or a pulse signal from the sender to the receiver. The location is calculated based on the amount of delay time of the pulse. However, the resolution usually varies from room level to meter level. For instance, Cricket sends a concurrent radio frequency signal and ultrasonic signal to infer distance, and it provides room-level localization by comparing the time delay between radio signal and ultrasonic signal [46]. Swadloon [47] provides meter-level indoor localization of the phone by using both the phase information and the Doppler effect. Some RF systems [48, 49, 50] also provides room level object localization. However, meter-level accuracy is not good enough for continuous tracking of an input gesture for a wearable device. Furthermore, the relatively large size and high power consumption makes these techniques impossible to be fit into an wearable device.

In order to recognize wearable input gestures with higher resolution, researchers built input devices with active ultrasonic technology. One of the earliest projects proposing this concept was Sonic Pen [51], which involved placing the speaker in a pen to generate a sonic pulse that was received by an orthogonal pair of microphones. The distance from the sonic pen to the microphones can be measured by comparing the received pulses at different receivers. Unfortunately, the published study on Sonic Pen only presents an estimate of the resolution achievable without providing any experimental results. When the

tracked device becomes small and wireless, it is usually challenging to provide synchronized signals, which are essential for calculating the time of travel between the speaker and receivers. To overcome this limit, infrared [52] or radio frequency (RF) signals are used to synchronize between devices [53]. Since these signals travel at the speed of light, comparing the arrival of the sound signals against the infrared/RF signals allows for a derivation of time travel of the sound. Mao et al. describe a high-precision Acoustic motion Tracker system (CAT) for 3D tracking, which uses a distributed Frequency Modulated Continuous Waveform (FMCW) to derive the change of distance to the speaker when the mobile moves from one position to another [54].

Recent projects also use the echo effect to detect hand or finger gestures [55, 7, 56]. Chirp shows the ability to detect hand gestures in 3D space by measuring the elapsed time between a transmitted ultrasonic signal and the received echo [55]. It is not clear how Chirp would work for finger gesture detection, because the fingers are much smaller than the overall hand and may, subsequently result in a less significant echo effect. Low-Latency Acoustic Phase (LLAP) has been used to detect the sound signal reflected by a moving hand or finger [56]. The phase change of echo signals caused by the hand or finger is used to detect and classify gestures. Though this technology does not require the instrumentation of the finger, LLAP can only track 1D and 2D movement and it is not clear whether it is capable of tracking hand movement in a relatively large area.

2.2.2 Continuous finger tracking for wearable input

Popular commercial finger tracking solutions (e.g., Leap Motion and Kinect) use computer-vision and depth sensors. While there are relatively small depth sensors available now, it remains a challenge to embed those cameras/sensors into the current wearable devices on the market. Other sensing modalities have also been utilized to develop continuous finger tracking technology. SkinTrack is able to track the finger position on the skin by passing electricity through the body [8]. However, it is hard to extend this technique to the 3D

space above the skin because air has very low electrical conductivity. FingerIO extends the input space for a smartwatch around the watch by detecting the echo pattern using the OFDM technique, but technique only works for a 2D space [7]. UTrack can track the finger movement in the space volume of 38400 mm^3 with a mean euclidean error of 24.54 mm for one-handed gestures using magnetic sensing [57]. However, because the intensity of the magnetic field attenuates cubically with distance, it is very challenging to track the finger position in a larger area.

My research proposes a novel 3D input solution for wearables using active acoustics. This technique only requires a relatively straightforward microphone array that we formed into a casing which fits around an existing wearable.

2.3 Human body mediated sensing

One key difference between wearables and other computing devices is that these wearable device are in contact with the skin of the body, which allows them to use the human body as a sensing medium for various purposes, such as communication and interaction.

2.3.1 Body-Area network

As wearable computers became a research topic in the 90s, the risk eavesdropping of wireless communication has been raised [58]. Zimmerman [59, 60] first proposed the idea of using the human body as a medium to construct personal area networks through capacitive coupling. Then Fukumoto [61] applied 'direct coupling' on the human body to communicate between devices on the finger and the wrist. After these pioneering efforts in the 90s, later research discovered the signal transmission mechanism of the body channel communication [62, 63, 64] as well as the electric field distribution of propagating electric signals through the human body[65]. A frequency of 10MHz was shown to be optimal for propagating electrical signals across the body [66]. Recently, Park et al. [67] demonstrated the possibility of using magnetic resonance for data transfer in body-area networks. Also,

applications such as biomedical monitoring systems [68], MP3 players [69], and real-time authentication [58], were built to demonstrate the enormous potential of constructing personal area networks via transmitting signals across the body.

All the work described above focuses on propagating electrical or magnetic signals (MHz frequencies) through the human body. The human body is also an active medium for transmitting acoustic signals. Researchers have already investigated how sound can be transmitted in the cadaver [70], and used sound to transmit information between smartphones [71, 72] or recognize gestures [**Harrison2010**, 73, 74]. OsteoConduct [75] used sound to transmit information within the same body with a very low bit rate (5 bits/sec). Recently, ViBand [13] demonstrated data transmission with a data rate up to 165 bits/sec between the watch on the wrist to a vibration motor. How the acoustic signal would transmit across a longer distance or from the body to other objects with a relatively higher data rates still remains unknown.

2.3.2 Approaching the skin as the input surface

Obviously, defining the skin as the input surface has great potential in the space of gesture design. Harrison 's demonstration of OmniTouch [76] showed how depth sensing can be used to enhance interaction around the hand and wrist, specifically using the skin as the interaction surface. Indeed, a person's skin is a good alternative input surface for wearable devices[77], because of its proximity to the device.

To recognize gestures on the skin, a variety of on-body sensing modalities have been explored, including capacitance [78], vision [76, 3], muscle activation via electromyography [79], infrared proximity sensors [35], motion/inertial and force [5], electricity[37] as well as sound [80, 2, 81, 34, 3].

Different from the previous works, my research demonstrates the possibility of intra-body signal transmission using acoustic signals under 20kHz from the wrist to different locations on the body, between two bodies, and between the body and other objects via

contact [82]. The preliminary characterization of the propagation of acoustical signals on different parts of the body also lays the foundation for other novel sensing techniques . A subsequent research project demonstrates how to recognize a rich set of one-handed input gestures by modeling the acoustic spectrogram on body using active acoustics sensing.

CHAPTER 3

NOVEL INPUT FOR COMMODITY SMARTWATCHES

Modern smartwatches usually adopt a touchscreen for interaction, whose functionality is limited by the inherent small size. New input is demanded for better interaction experience on smartwatches. One obvious solution is to add new input hardware to support richer set of input gestures or more freedom of operation. For instance, by replacing the band with a touchscreen, the input area is enlarged. Adding new hardware requires a longer cycle before it can be widely applied in real-world applications. However, most smartwatches are equipped with various sensors which can be potentially re-purposed to support a more extensive set of input gestures. By using the built-in hardware of the commodity smartwatches, these input techniques can be quickly added to the commercial devices without further change of the hardware.

In this chapter, I present WatchOut and TapSkin, two novel input techniques for smartwatches that are designed and implemented with only the built-in hardware of commodity smartwatches.

3.1 WatchOut: Extending Interactions on a Smartwatch with Inertial Sensing

3.1.1 Overview

Most modern portable devices, such as the smartphone and smartwatch, come equipped with an inertial measurement unit (IMU). In my previous work[44], I demonstrated a set of slide and tap gestures on the edge and back of a commodity smartphone can be supported to improve interaction experience by using the IMU and microphone. In a similar spirit, I present here WatchOut, which is a suite of interaction techniques using inertial sensors to provide a variety of tapping and sliding gestures on the side, bezel, and band of

a smartwatch as shown in figure 3.1.

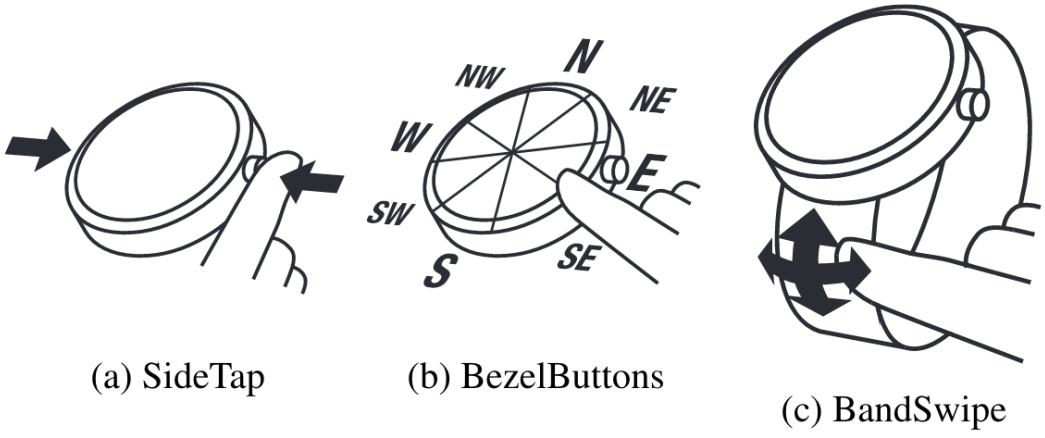


Figure 3.1: WatchOut Interaction Families

3.1.2 Gesture Design

Based on our observations of the raw sensor data from various tap and swipe gestures around the case and band of a watch, we designed three families of gestures: SideTap; BezelButtons; and BandSwipe (see Figure 3.1).

SideTap

The user can tap on either the left or the right side of the watch case. These interactions can be performed eyes-free, and are appropriate for performing simple acknowledge or dismiss actions, such as rejecting a phone call.

BezelButtons

A user can tap the bezel area around the outside of the screen, as Figure 3.1b shows. We can recognize up to eight tap locations, whose positions are equally distributed around the watch case. Intuitively, these eight tap gestures can be used for navigating directionally, as

with a D-pad. Potentially, BezelButtons can also help facilitate richer menus on the watch. For instance, most watch applications can only display a limited number of menu choices (usually around three items) due to the limited screen real estate.

BandSwipe

We can turn the band of a watch into an interactive surface by recognizing four different directions (up, down, left, right) of a sliding gesture, as Figure 3.1c shows. These four gestures can be naturally used in applications that require directional controls. The BandSwipe gestures can also be used in combination with the touchscreen to enhance multitouch interaction.

3.1.3 Theory of Operation

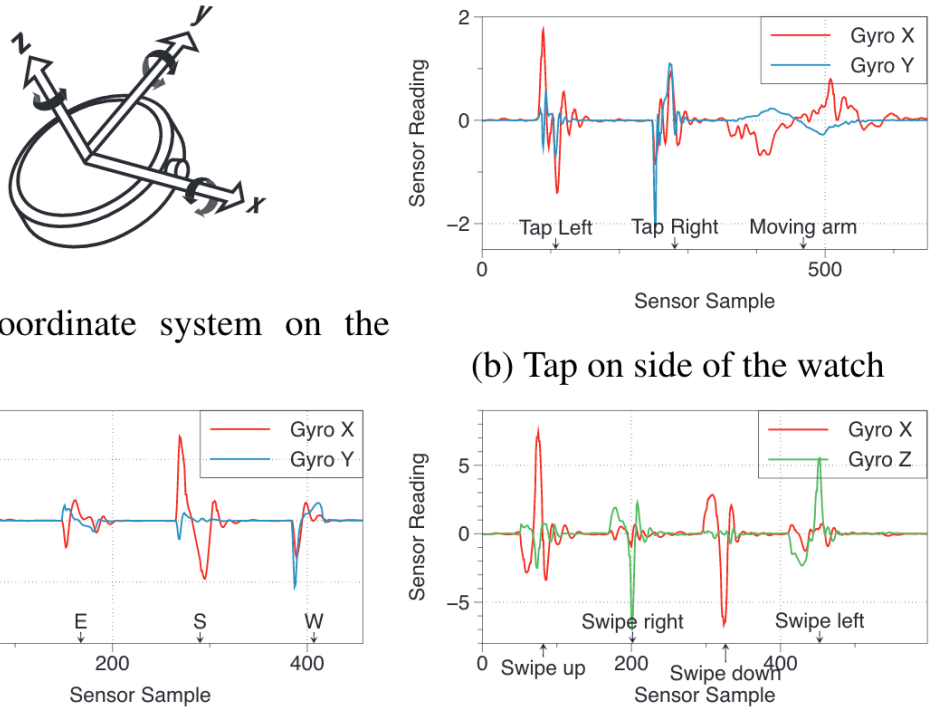
WatchOut is developed based on the observation that the tap and slide gestures on the watch case can be well captured by the IMU. Figure 3.2 shows the raw IMU sensor data of tapping or sliding on different positions on a watch. These gestures are visually distinct from each other, which can potentially be recognized with appropriate algorithms.

3.1.4 Gesture Recognition

To recognize a gesture, we first segment the sensor data for event detection, and then classify the gesture with pre-built machine learning models. We use the sequential minimal optimization (SMO) implementation of a support vector machine (SVM) provided by Weka[83] to build the models, which are also used for real-time event detection and classification in our interactive prototype.

Event Detection

We collect gyroscope and accelerometer (linear acceleration) data on the watch at their highest sample rates. To process the data, we apply a one-second sliding window with 50%



(a) The coordinate system on the watch

(b) Tap on side of the watch

(c) Tap on watch bezel

(d) Swipe on watchband

Figure 3.2: Sensor data collected when various gestures are performed

overlap. Within the window, we compare the maximum absolute value for each axis of the gyroscope and the linear acceleration, respectively, against a pre-determined threshold value. If one of the threshold criteria is satisfied, we then identify the peak absolute value in the entire data buffer for each sensor (gyroscope and accelerometer), and the axis on which that peak falls. We use 0.25 seconds data before the peak position and 0.25 seconds data after the located peak to form an instance containing data of 0.5 seconds, which will be used for feature extraction.

Classification

To extract features, we first derive virtual sensors (Deri-gyro, Deri-linaccl) by calculating the derivative for each axis of the gyroscope and the linear accelerometer. For each sensor and virtual sensor axis, we calculate a set of statistical features—the minimum,

maximum, mean, standard deviation, median, root-mean-square (RMS), variance, zero-crossing-counts, the values and difference of the first and second peaks. To capture the relative relationship between three axes of each sensor, we also calculate the ratio of the energy, the first and second absolute peak values, the difference between the first and second absolute peak values, and the correlation for the data for each pair of axes on each sensor. In total, 192 features are extracted for each half-second event frame.

The features are used to train two support vector machines (SVM) for gesture classification. The first classifier distinguishes the noise instances from the gesture instances. If a gesture is detected, the second classifier then identifies which gesture is being performed and we move to the next window without overlap.

3.1.5 Evaluation

We implemented our interaction techniques on two Android Wear smartwatches: 1) the LG G Watch Urbane with a round screen and a leather watchband. and 2) the Sony Smartwatch 3 with a square screen and a rubber watchband. We offload most computation work to a paired Google Nexus 6 smartphone, All the inertial sensors are set to sample at the highest rate (200-250 Hz) during the data collection process.

We collected training data from seven people (3 researchers) to build two classification models (event detection and gesture classification) for each of the three gesture families and for each watch ($2 * 3 * 2 = 12 models$). Each trainer provided 20-40 samples for each gesture, in order to train the gesture classifier. Each trainer also provided 20-40 samples of simulated noise (non-gestures e.g., moving and rotating arm and wrist), in order to train the event detection classifier. The noise samples included actions such as raising the arm and rotating the wrist. After collecting these samples, we built the event detection model by labeling all the gestures as gesture, and the non-gestures as noise.

We evaluated WatchOut in a laboratory environment with 12 participants (4 females, average age 25.7). We had each participant perform gestures from the three gesture fam-

ilies (SideTap, BezelButtons, and BandSwipe) on each of the two smartwatches (LG and Sony). Three of the participants are daily smartwatch wearers, four had experience using a smartwatch, eight had never interacted with a smartwatch. We asked participants to wear each watch on the left arm and use the right hand to perform the gestures. Each participant performed two sessions for each gesture family on each watch. Each session consisted of ten stimuli for each gesture in the family, presented in random order. In addition, participants also recorded two sessions of ten "noise" events by lifting their arms, as described in training above. We saved the sensor data for the gesture in files with a length of 3 seconds for later analysis.

Results

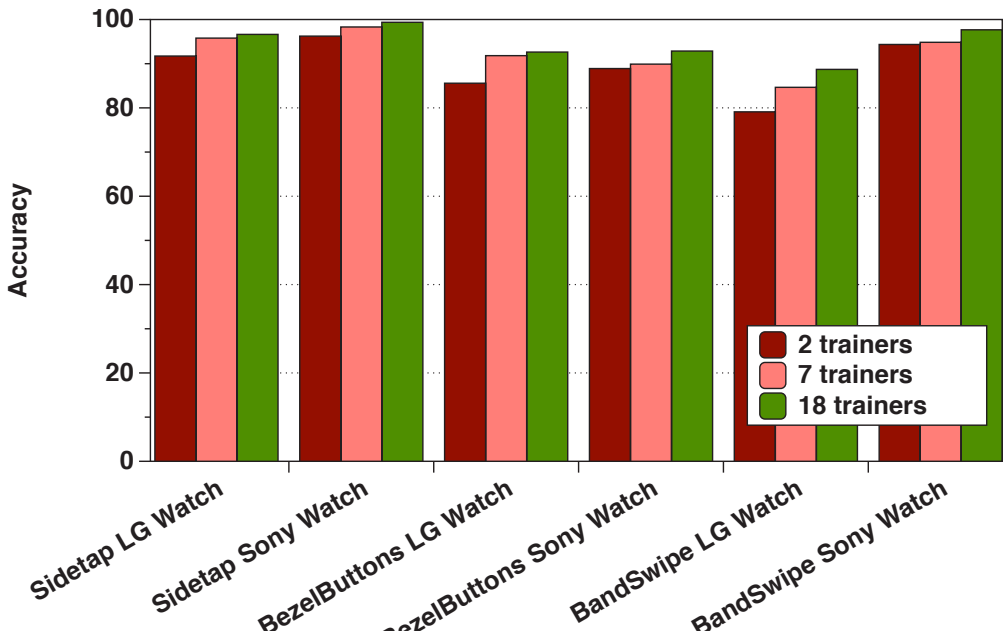


Figure 3.3: Overall accuracy of gesture classification for 12 participants, based on 2, 7, and 18 people in training set.

In total, we collected 6698 gesture samples from 12 participants. We ran several analyses of the gesture recognition performance offline, with data collected from the 12 participants and the training sets.

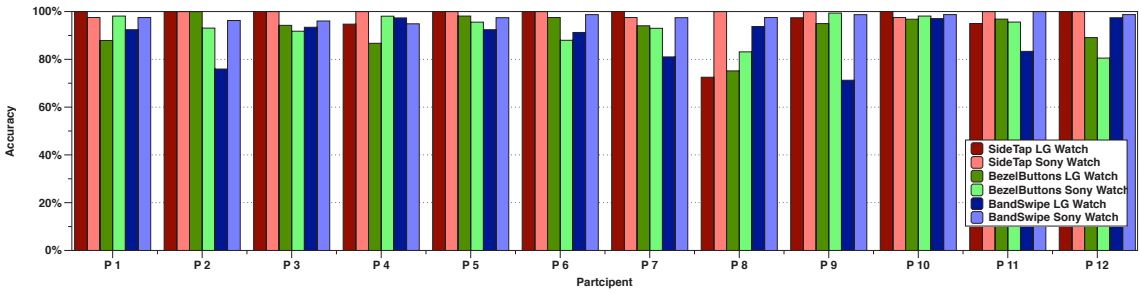


Figure 3.4: Gesture recognition accuracy for each participant

In the first analysis, we used "leave-one-participant-out" methods. Both the noise-classifiers and gesture-classifiers were trained with all the data from the seven trainers, in addition to data from the remaining eleven participants.

The first stage of our gesture recognition pipeline is event detection. If we did not detect any gesture event in the sensor data from the files representing one gesture instance, we counted that instance as a false negative error. If we detect more than one gesture instance from the sensor data, we counted that instance as a false positive error. The false negative and positive instances were not passed into the gesture recognition classifier. Out of 6698 total gestures recorded, the total number of false positives was 37 and the total number of false negatives was 56. The false positive rates ranged from 0.00% (BezelButtons on the square watch) to 2.40% (BandSwipe on the round watch). The false negative rates ranged from 0.00% (SideTap on the square watch) to 1.30% (BezelButtons on the square watch).

The second stage of the pipeline is the gesture recognition classifier. Most of the gestures were classified with accuracies of above 90%, based on the user-independent model trained with data from 18 people. The best accuracy was observed for the SideTap gesture family, which is not surprising because it only included two gestures (left and right taps). The worst accuracy was for BandTap on the round watch, which had a leather band with stitching.

To investigate the influence of the size of training data to the classification performance, we compare the recognition results when only two trainers are used (leftmost/red bar for each condition), when 7 trainers are used (middle/pink bar), and when 18 trainers are used

(rightmost/green bar). Figure 3.3 reveals that for all cases, the accuracy improves with more training data, but the increase is much greater when moving from 2 to 7 trainers than when moving from 7 to 18 trainers.

3.1.6 Discussion

Generalizability

In order to determine if the gesture recognition models were *device*-independent, as well as *user*-independent, we tested participants' data from the round watch with the models generated for the square watch, and vice versa (see Figure 3.5).

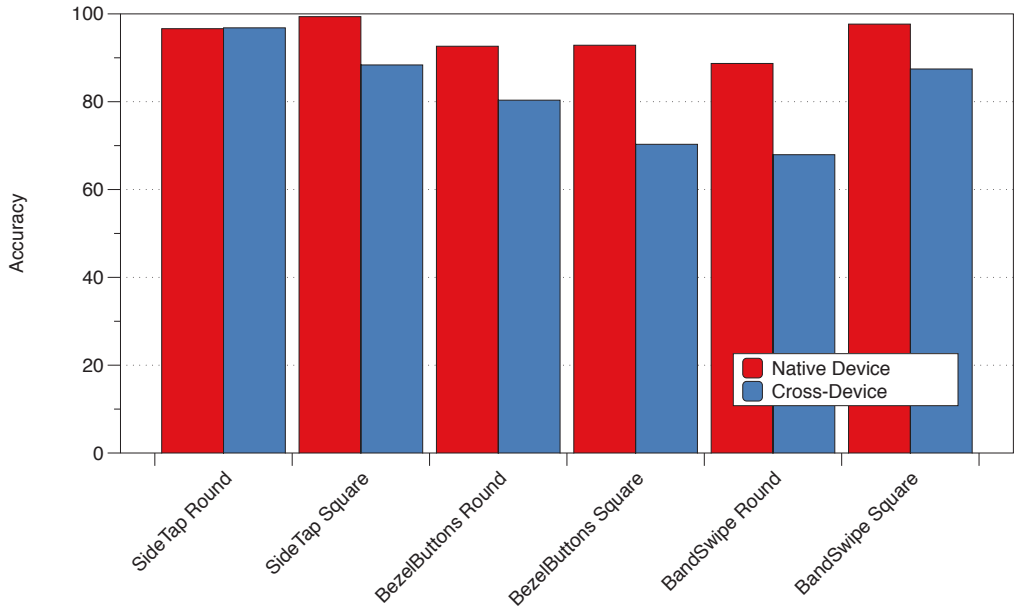


Figure 3.5: Device-independent gesture recognition accuracy.

The cross-device accuracy results for SideTap were virtually the same for the round watch (96.8% vs. 96.6% for the native model), and reasonably good for the square watch (88.4% vs. 99.4% for the native model). The BezelButtons gesture family does not appear promising for building device-independent models, at least between round and square watches. The cross-device accuracy results for BezelButtons were worse for the round

watch (80.4% vs. 92.6% for the native model), and considerably worse for the square watch (70.3% vs. 92.9% for the native model). We suspect that the distinct corners of the square watch provide a more specific and consistent target for the NE, SE, SW, and NW gestures as compared with the round watch. The different shapes of the watch cases and location of the IMU may also produce different physical responses in the inertial sensors to taps on the bezel.

We observed the highest discrepancy of accuracy between watches for BandSwipe, likely based on the differences in each watch band. The square Sony watch had a smooth rubber band, while the round LG watch had a more textured, leather band with raised stitching. Overall, the BandSwipe recognition accuracy was considerably lower (88.7%) on the leather band as compared to the accuracy of 97.7% on the rubber band. These difference may be due to how the friction from swipes in the two materials is reflected in the inertial sensors, and also due to the difference described above on how some users performed swipes on the leather band.

BezelButtons Accuracy with Four or Eight Gestures

The BezelButtons family includes eight distinct gestures that our classifier can distinguish, as compared to four for BandSwipe and two for SideTap. The recognition accuracy was sufficiently high, at 92.6% for the round watch and 92.9% for the square watch. Upon examination of the confusion matrices, we discovered that the vast majority of classification errors were for adjacent gestures (e.g., NE or NW when N was intended) as shown in figure 3.6.

However, many applications does not require up to 8 gestures. A subset is probably enough to provide full functionality for some applications such as music player. To explore BezelButtons with four gestures (BezelButtons-4), we ran four additional analyses. We built new models that included only the N, E, S, and W gestures (BezelButtons-4N) for each of the two watches. We also built new models that included only the diagonal ges-

	N	NE	E	SE	S	SW	W	NW
N	0.90	0.01	0.00	0.00	0.00	0.00	0.00	0.09
NE	0.00	0.96	0.03	0.00	0.00	0.00	0.00	0.01
E	0.00	0.01	0.91	0.08	0.00	0.00	0.00	0.00
SE	0.00	0.00	0.03	0.93	0.04	0.00	0.00	0.00
S	0.00	0.00	0.00	0.04	0.92	0.04	0.00	0.00
SW	0.00	0.00	0.00	0.01	0.05	0.92	0.02	0.00
W	0.00	0.00	0.00	0.00	0.00	0.03	0.96	0.00
NW	0.04	0.01	0.00	0.00	0.00	0.00	0.03	0.92

(a) On the round watch.

	N	NE	E	SE	S	SW	W	NW
N	0.91	0.07	0.00	0.00	0.00	0.00	0.00	0.02
NE	0.00	0.91	0.08	0.00	0.00	0.00	0.00	0.00
E	0.00	0.03	0.94	0.03	0.00	0.00	0.00	0.00
SE	0.00	0.00	0.01	0.97	0.02	0.00	0.00	0.00
S	0.00	0.00	0.00	0.04	0.95	0.00	0.00	0.00
SW	0.00	0.00	0.00	0.00	0.01	0.98	0.00	0.00
W	0.00	0.00	0.00	0.00	0.00	0.04	0.87	0.08
NW	0.01	0.00	0.00	0.00	0.00	0.00	0.10	0.88

(b) On the square watch.

Figure 3.6: Confusion Matrix for BezelButtons

tures (NE, SE, SW, SW) for both watches (BezelButtons-4NE). The overall gesture recognition accuracy for BezelButtons-4 improved further on both watches, as compared with the accuracy for BezelButtons-8 . On the round watch, the gesture recognition accuracy jumped to 98.3% and 99.3% , respectively for BezelButtons-4NE and BezelButtons-4N, as compared with 92.6% accuracy for BezelButtons-8. On the square watch, the gesture recognition accuracy jumped to 99.4% and 99.2%, respectively for BezelButtons-4NE and BezelButtons-4N, as compared with 92.9% accuracy for BezelButtons-8.

Applications

We have designed and implemented applications that demonstrate how WatchOut can improve smartwatch interactions.SideTap offers two gestures, which we demonstrate acknowledging or dismissing an alert, such as a phone call. BezelButtons offers up to eight distinct gestures, which we demonstrate in an always-available interface for shortcuts to launch applications. BandSwipe offers an additional modality for navigation, which we demonstrate in a map navigation application.We imagine extending these gesture families to other square, round, and band-type wearable surfaces: such as belt buckles, straps, and jewelry, in future work.

3.2 TapSkin: Recognizing On-Skin Input for Smartwatches

3.2.1 Overview

In addition to the watch case and band, I also extended input area to the skin around watch using built-in IMU and microphone of a commodity smartwatch. I present TapSkin, a novel interaction technique that provides a family of tap gestures (up to 11 tap locations) on the skin around the wrist area without additional instrumentation, by only using the inertial measurement unit (IMU) and the microphone on a commodity smartwatch. By extending the input outward to the skin, the user can view the screen while interacting, similar to the indirect interaction provided by a trackpad on a laptop. Although approaching the skin as the input surface has been previously explored [2, 77], those approaches have exploited additional instrumentation of the body to achieve the interaction. We demonstrate providing a rich set of tapping gestures that work with the current sensing capabilities of a smartwatch, specifically its microphone and inertial sensors.

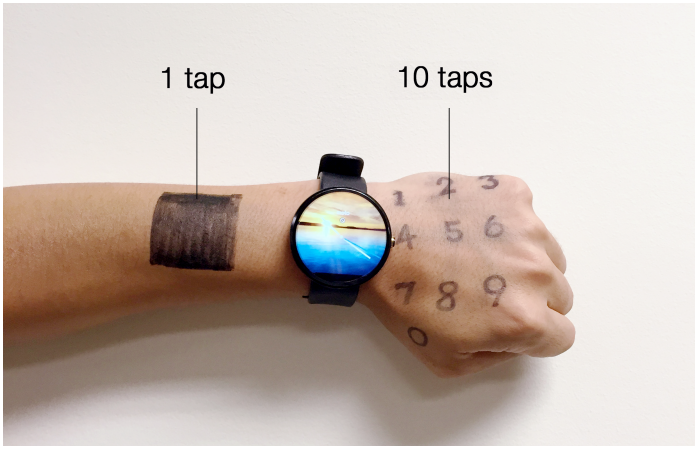


Figure 3.7: The TapSkin interaction technique. Drawings on the skin are only for clarity to indicate where input events can be performed.

3.2.2 Gesture Design

TapSkin provides three sets of tapping gestures that can map into useful input: the NumPad, the DPad, and the CornerPad. Figure 3.8 depicts these three gesture sets. For each set,

there is also a single tap on the top of the forearm beyond the smartwatch (depicted as a gray rectangular patch), which can be used to activation gesture or redo function.

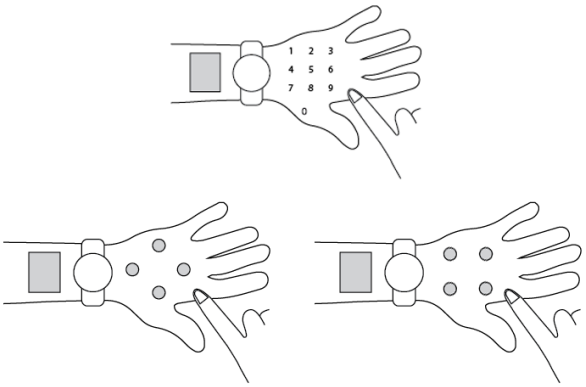


Figure 3.8: The TapSkin gesture sets: NumPad (top), DPad (bottom left), and CornerPad (bottom right).

NumPad

The NumPad gesture family simulates the layout of a number pad, which provides 10 distinct tap locations on the back of the hand and one tap location to the left side of the watch (as seen in Figure 3.8). It can potentially be used for entering numbers or even text (through multi-tap or T9), which is currently difficult on a small touch screen.

DPad

The DPad contains the TapLeft gesture as well as four tap gestures located at the top, bottom, left, and right parts of the back of the hand. It can be used for directional controls or scrolling through a list of items on the smartwatch. For instance, the most common operation on a smartwatch touchscreen is to scroll down the menu vertically and horizontally. This operation can be naturally mapped to the four tap gestures using DPad.

CornerPad

The CornerPad includes the four corners on the back of the hand. The two gestures on the left are close to the wrist joint, and the other two gestures on the right are close to the knuckles at the base of two fingers. These four distinct tap gestures can be used as shortcuts to certain applications on a smartwatch. The TapLeft gesture again serves as the activation gesture into CornerPad.

3.2.3 Implementation of TapSkin

The recognition of TapSkin gestures are developed based on the observation that , the sounds of tapping on different locations of the hand are distinguishable from each other. In addition, the act of tapping will induce a movement of the arm that the smartwatch is mounted on. These signatures can be captured by the IMU and microphone embedded in the smartwatch. Therefore, we developed TapSkin on commodity smarwatches (Sony SmartWatch3, Moto 360) using available sensor data from the gyroscope, accelerometer, and microphone. A Google Nexus 6 phone is used to running the recognition algorithm. .

We apply a traditional machine learning pipeline to recognize the gestures from these sensor data, which consists of two steps: gesture event detection and gesture classification.

Event Detection

To detect whether there is a gesture event in each 1.2 seconds sliding window, we inspect the window to determine if either the gyroscope or accelerometry data passes an empirically-determined threshold value based on the maximum energy in the window. If one of the thresholds is satisfied, we continue to the next step in the pipeline to define an event window. Within the original sliding window used for event detection, we locate the position of the maximum absolute value of each sensor signal. Note that the max points may be at different times for each sensor. We then define an event window of data values for each sensor stream balanced around that maximum point. The size of event window is

0.5 seconds both IMU data (accelerometer and gyroscope) and acoustic data.

Gesture recognition

For each event window, we calculate a vector with 286 features. For each axis of the linear acceleration and the gyroscope data, we first derived virtual sensors by taking the derivative of each axis. Then for each axis of the raw sensor and virtual sensors, we extracted a set of statistical features including the maximum, minimum, mean, median, standard deviation, root-mean-square(RMS), variance, zero-crossing rate, and the values of peaks and their differences. To capture the relationship between the different axes of each IMU sensor, we also calculate the ratio and differences of the energy, first and second absolute peak values of each two axes on each sensor. We also perform a Fast Fourier Transform (FFT) on the gyroscope data (50 points), which results in 24 features (abandoned the first bin) for each axis. Since the frequency of our gesture is under 6kHz, we only add the values of first three bins of the FFT results and the entropy across the values of different bins into a feature vector.

For the acoustic data, we extracted features in the frequency domain. We divided the window into 30ms frames with 50% overlap. We first extract 26 Mel-frequency cepstral coefficients (MFCC) features, based on the observation that the sounds generated by these on-skin gestures are mostly under 4kHz, which is similar to the range of the human voice. For each frame of data, we also calculate the FFT and average the results across different frames. We only keep the first 30 values of the FFT results, which represent the magnitude of frequencies between 0Hz to 500Hz, which we determined empirically as the most informative frequency band for the sound of our gestures. The center of mass across the frequencies is also added to the feature vector.

The feature vector is used to train a support vector machine (SVM) to recognize which gesture has been performed. We use the sequential minimal optimization (SMO) of SVM provided by Weka for building our training models and for real-time classification in our

prototype system [83].

3.2.4 Evaluation

We conducted a user study with 12 participants (6 male) with an average age of 26 on a Sony SmartWatch 3 (Android Wear) for data collection and a Nexus 6 (Android 5.1) for running our machine learning algorithm. The gyroscope,linear acceleration and the microphone are sampled at 200Hz, 250Hz, and 8000Hz respectively.

The study was conducted in an open space at a university building, where the sound from the air conditioner as well as people's talking and movement can be heard. Participants was instructed to wear the watch on the left wrist and use the right hand to perform the gesture. For consistency and helping participants remember the locations of each tap gesture, a number pad as seen in figure 3.7 was drawn on the back of each participant's hand.A researcher first demonstrated the entire gesture set and asked the participant to practice. The formal study consisted of 8 sessions. During each session, the participant performed all the gestures while sitting in front of a table, where they could rest their elbow during the study. The participants were advised to take breaks between sessions.

During each session, they were asked to follow the stimuli displayed on the screen of the watch to perform the NumPad gestures. The sequence of the stimuli was randomized. Five instances of each gesture were collected per session, and 11*5 = 55 total gesture events were collected per session. Each gesture event was saved into 2-second raw data files for further analysis. In total, we collected 5*11*8 = 440 gesture instances from each of the 12 participants. We recorded video of the sessions for later analysis.

"Leave-one-session-out" evaluation results

During the study, each participant provided gesture instances through 8 separate sessions. To understand how the system works when the training data came from different sessions, we applied a "leave-one-session-out" method to evaluate the system. For each participant,

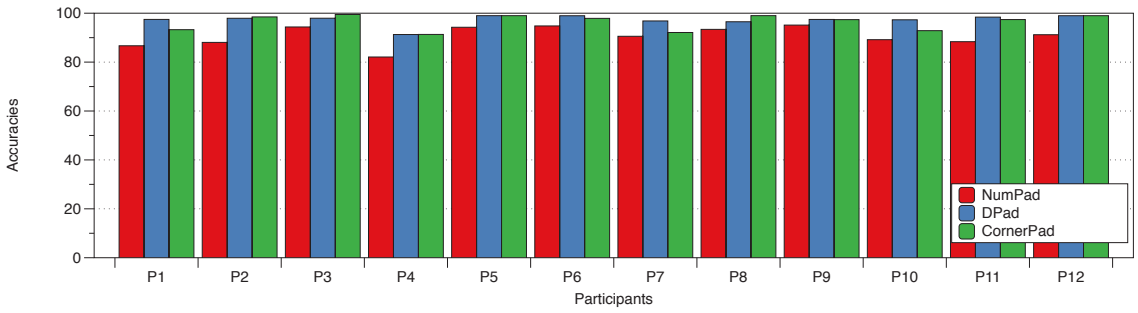


Figure 3.9: The classification accuracy for each participant

we iteratively used the instances from one session as the testing data, while using the instances from the rest seven sessions as the training data to build the model.This step was repeated eight times for each participant and we reported the overall results.

In our data processing pipeline, the first step is to detect the gesture event. If we failed to detect any gesture event in any sliding window in the offline files representing a gesture instance, we counted this instance as a false-negative error. If we detected more than one gesture event representing a gesture instance in the files' sliding windows, we counted these instances as false-positive errors. We did not run these instances with errors through gesture classification. As a result, 9 false-positive errors and 162 false-negative errors were detected out of 5264 gesture instances (NumPad). The false positive rate are 0.17%, 0.17%, 0.21%, and the false negative rate are 3.08%, 2.68%, 2.84% respectively for NumPad, DPad and CornerPad.

The average classification accuracy across 12 users for NumPad in the leave-one-session-out evaluation is 90.69%. The accuracy for each participant is shown in Figure 3.9. Participant 9 (P9) provided the gesture instances with the highest accuracy (95.14%) and P4 provided the instances resulting in the lowest accuracy at 82.09%. We reviewed the video from P4's data collection session and found that the way the participant performed the tap gesture was apparently lighter than others. We present the confusion matrix of NumPad in Figure 3.10.The lowest precision of 85.62% was provided by N8. Most of the confusion exists between adjacent tap locations.

	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	TapLeft
N0	97.15%	0.22%	0.44%	0.44%	0.00%	0.00%	0.00%	0.66%	0.88%	0.22%	0.00%
N1	0.22%	85.81%	6.88%	1.29%	2.80%	0.43%	0.22%	1.08%	0.00%	0.43%	0.86%
N2	0.00%	4.70%	86.75%	5.56%	0.43%	1.71%	0.85%	0.00%	0.00%	0.00%	0.00%
N3	0.43%	0.22%	3.70%	93.26%	0.00%	0.87%	1.30%	0.00%	0.22%	0.00%	0.00%
N4	0.22%	0.43%	0.43%	0.00%	90.91%	5.19%	0.43%	0.87%	0.00%	0.65%	0.87%
N5	0.00%	0.44%	1.54%	0.44%	3.96%	89.43%	2.86%	0.44%	0.88%	0.00%	0.00%
N6	0.00%	0.21%	0.21%	1.27%	0.21%	3.82%	93.84%	0.00%	0.21%	0.21%	0.00%
N7	1.08%	0.87%	0.43%	0.43%	1.52%	0.22%	0.00%	89.59%	3.69%	1.74%	0.43%
N8	1.74%	0.00%	0.00%	0.22%	0.65%	1.31%	0.22%	3.05%	85.62%	7.19%	0.00%
N9	0.85%	0.21%	0.00%	0.21%	0.63%	0.42%	1.27%	0.21%	7.40%	88.79%	0.00%
TapLeft	0.00%	0.43%	0.43%	0.22%	0.86%	0.22%	0.00%	0.86%	0.22%	0.22%	96.55%

Figure 3.10: Confusion Matrix for NumPad

The DPad gesture family is a subset of NumPad gestures. The four tap gesture is mapped to the N2, N4, N6, and N8 in the NumPad gesture set. The layout of the these four tap locations is similar to a control pad, which can be used to navigate in the up, down, left and right directions. The average accuracy for DPad across 12 participants is 97.33%. TapLeft has the highest accuracy 97.84% and N4 offered the lowest at 96.75%.

The layout of the CornerPad gestures is mapped to the four corners on the back of the hand. By locations, they are mapped to the N1, N3, N7, and N9 gesture of the NumPad set. The average accuracy across 12 participants is 96.64%. The most accurate gesture is N9 with an accuracy of 98.52% and the least accurate gesture is N7 with an accuracy of 94.36%.

3.2.5 Discussion

Evaluation with more practical models

The results of leave-one-session-out validation provide a good estimate of how well the classifier performs on recognizing different gestures across different sessions. However, in practice, building a real-time gesture recognition model would face more challenges, such as how many training instances are needed to build a model for a person, whether the model can be built for user-independent operation, and how many calibration instances would be necessary to improve the classification accuracy. To answer these questions, we conducted another set of experiments by building user-dependent, user-independent(leave-

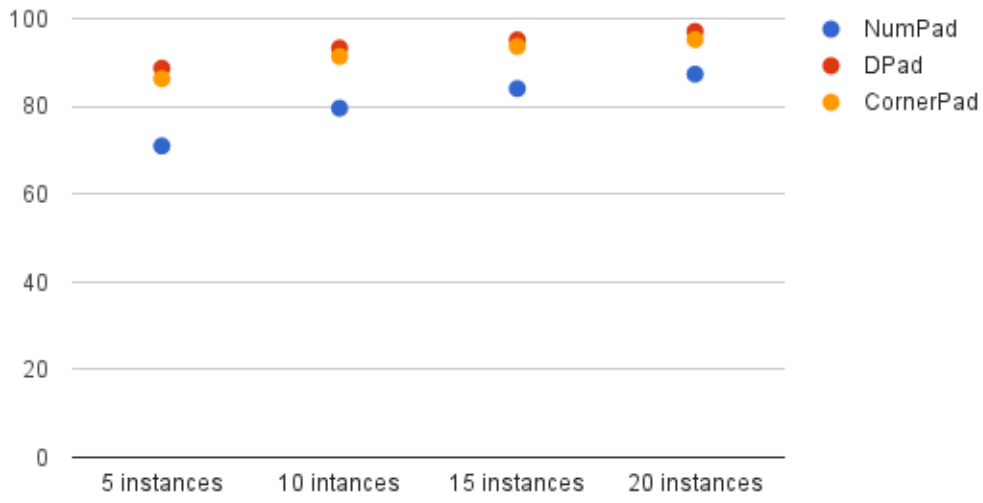


Figure 3.11: Accuracy of different number of training instances for user-dependent models

one-participant-out), and user-adaptive models. Similarly, we simulate the real-time classification pipeline by applying a sliding window of 1.2 seconds with 75% overlap for all the files we collected in all the models we built in this section.

User-dependent models only use the data from the same participant to train a model which will be then used for classifying his/her own gestures. Usually, a model trained with more training instances provide higher recognition accuracies, but also require longer time to collect the data from the participant, which can be annoying. Therefore, to investigate how many instances are needed to train a user-dependent model, we built user-dependent models for each participant by using the first 5, 10, 15, and 20 collected instances per gesture from that same participant to train an SVM model. We used the remaining instances of that participant to test the model. The results are shown in Figure 3.11. When only using 5 instances to build a model, the overall accuracy for NumPad, DPad and CornerPad across 12 participants were 70.97%, 88.69% and 86.3%. The highest accuracies were 87.33%, 97.09%, and 95.23% respectively for NumPad, DPad and CornerPad, when we trained the

user-dependent models with 20 instances per gesture for each participant.

Compared with user-dependent models, user-independent models(Leave-one-user-out) do not require the user to provide any training data before using the technique. It would improve the user experience by reducing the workload on the side of the user, but it places a higher standard for the reliability of the gesture models.

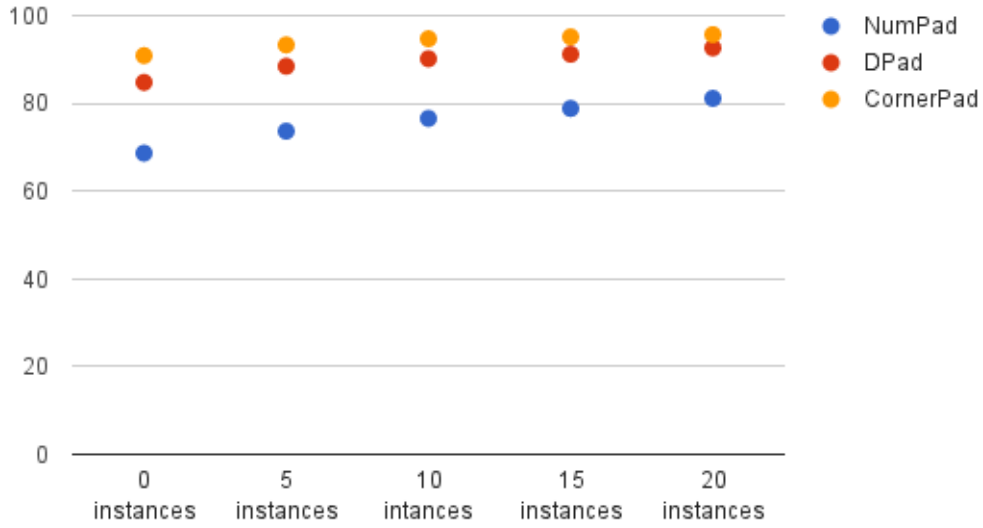


Figure 3.12: Accuracies of User-independent and user-adaptive Models. Along the x-axis, a value of 0 instances is the same as a user-independent model.

We used the data of 11 participants from the total 12 participants to train a support vector machine, which is then tested with data from the remaining participant. The average accuracy across 12 participants for the NumPad, DPad and CornerPad are 68.65%, 84.78% and 90.35% as shown in Figure 3.12.

The user-adaptive model falls in between the user-independent and user-dependent models, considering the amount of effort required for each user to provide training instances. It strengthens the user-independent models by adding a few number of instances per gesture from an individual user. We evaluated our technique on user-adaptive models

by incrementally adding 5, 10, 15, and 20 instances per gesture from each participant to the user-independent training set. As Figure 3.12 shows, the more personalization data is added to the models, the higher the average accuracies are. The highest accuracies are reached when 20 instances are added to the models, which are 81.13%, 92.64%, and 95.66% for NumPad, DPad and CornerPad.

We notice that the accuracies of user-adaptive models with 20 instances per gesture are lower than the accuracies of user-dependent models built with similar number of training instances. We believe this is due to tap locations being different from person to person based on different hand sizes. For example, tapping at the same location of the hand (e.g., knuckle) can generate a different sound from person to person. Therefore, adding more training data from others may not necessarily help improve the recognition accuracy. However, the user-independent model may work for a smaller set of gestures. CornerPad achieved 90.35% accuracy with user-independent models. With 5 and 10 instances added, the accuracy further increased to 93.32% and 94.72%.

Others

There are several important limitations of our results and system to consider before applying TapSkin to real applications.

First, in our validation experiment, gesture instances were collected while the participants were holding their arms in a static position. In practical scenarios, the users may use TapSkin while they are in motion (e.g., while walking), which may cause more false-positive errors in event detection. We think an activation gesture can be used to avoid this issue in practice.

Second, to help the experimental participants learn the tap locations, we drew the layout on the skin, a solution that may not be desirable in practice. One challenge is to design a gesture set that is both memorable as well as machine recognizable.One way to enhance recognition is to provide a small set of tap gestures, which are located at greater distances

between each other (e.g., DPad, CornerPad). Another method to prevent drawing icons on the skin is to utilize the natural layout of the hand and map gestures to different parts of the body. For instance, each hand consists of 4 knuckles.

CHAPTER 4

GESTURE INPUT FOR WEARABLES USING CUSTOMIZED HARDWARE

In the previous section, I designed interactions on the off-the-shelf smartwatches , using only built-in hardware. Though this approach is more practical, the richness of the input vocabulary and the freedom of gesture design are relatively limited because I do not have the choice of the hardware, which are not designed for detecting input gestures. For instance, the IMU in a watch is widely used for activity tracking with a relatively low sample rate(under 100 Hz). However, to capture subtle difference between input gestures, higher sample rate is preferred. Also, the microphone is designed to capture the air-borne sound (e.g., human speech) rather than the on-body communication. Therefore, using it to capture on-body acoustic signals are not ideal as what I have done in TapSkin.

In this chapter, I presented a set of projects using customized ring to provide various sets of input gestures that can be potentially applied to address input challenges such as, one-handed number input, text input, hand pose recognition, and menu selection.

4.1 FingerSound: Recognizing Unistroke Thumb Gestures using a Ring

4.1.1 Overview

Input character on wearables is challenging, which is missing on most commodity wearables. Twiddler[84] provides a complete one-handed input solution, but requires a relatively long period of learning.

I developed FingerSound, a system for unistroke thumb gesture recognition that enables character-based input for wearable computing devices. FingerSound uses a ring on the thumb to detect unistroke gestures made against the hand. Input can be started virtually at any time and in any position without requiring the visual attention of the user to select each

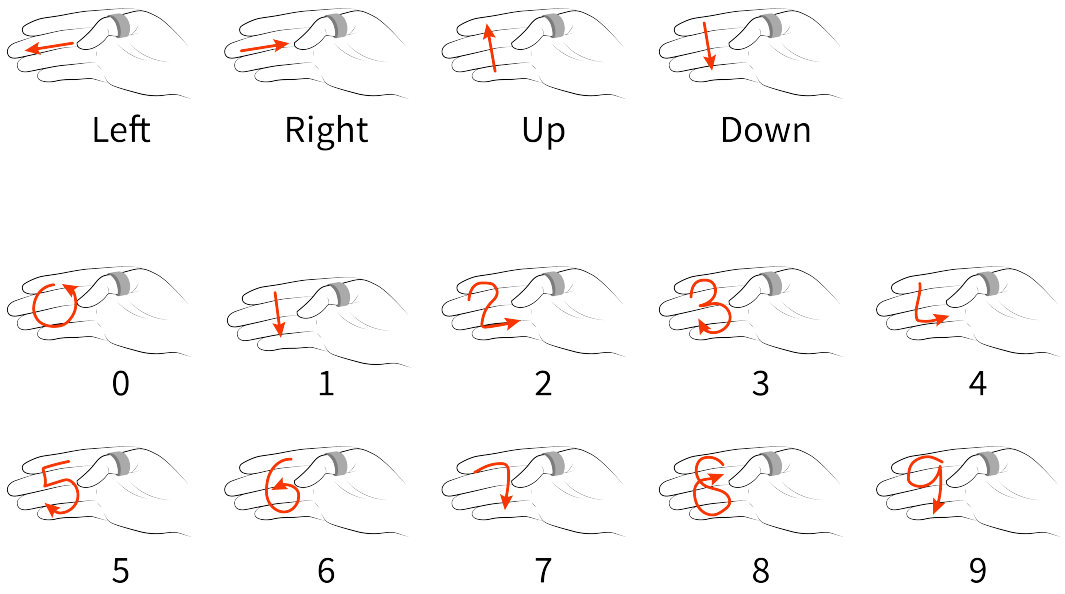


Figure 4.1: The top shows the FingerSound prototype and typical placement on the user's thumb. The bottom shows the two unistroke gesture families (directional and digits) we designed, implemented and evaluated.

letter. Similarly, command gestures can be made without requiring the visual attention of the user or causing the user to feel around the physical environment blindly searching for an interface device.

4.1.2 Gesture Design

FingerSound can recognize two sets of Graffiti gestures: the 10 digits number input(figure 4.1) and the 26 English character input (figure 4.2). Graffiti is a gesture set that was created by Palm, Inc.[1] to provide text input on PDA. Each Graffiti gesture resembles the uppercase form of the English alphabet, such that it is easy to learn and use. A previous study already has shown that the participants can achieve an accuracy of 97% after five minutes of practice [85]. Showing that our system is able to recognize the Graffiti gestures not only demonstrates the power of our technology to recognize a rich set of unistroke gestures, but also examines the possibility of using this technique as an alternative text

[1] http://www.palm.com

Figure 4.2: Additional 28 unistroke gestures used in the followup study

input method in the future.

To perform a gesture, the users are required to scrape the ring-bearing thumb along the palm or fingers, so as to make a preset pattern. This scraping action of rubbing the thumb against the skin creates a small, but easily perceived sound as well as subtle motion movement of the thumb. This sound is picked up by a contact microphone on the FingerSound ring which is used to activate the recognition system. But before the activation happens, this sound captured by the microphone along with the motion data captured by the gyroscope sensor passes through multiple filtering mechanisms and gets analyzed to determine whether a gesture was performed or whether it was simply noise from other finger-related activity. Given the location of the device on the finger, it is very easy to get input sounds and motion which are produced not while performing a gesture pattern but are generated by any kind of touching of the fingers to another surface. This is like the Midas touch problem [86] that our system needs to handle. We designed an activation gesture with a dedicated machine learning pipeline to recognize the gestures while rejecting the noise. The detailed technical implementation is explained in the following sections.

4.1.3 Implementation

Hardware

To capture the thumb gestures, we designed a ring with a built-in contact microphone and gyroscope to capture the relevant acoustic and motion data as shown in figure 4.3.

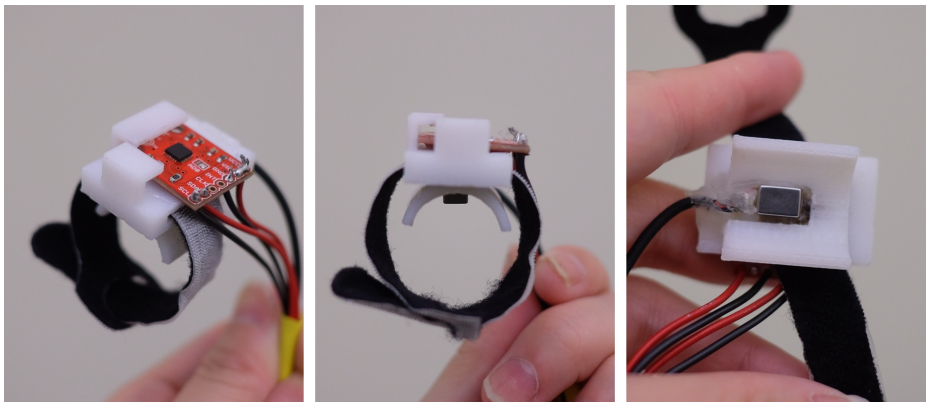


Figure 4.3: The ring with a contact microphone and a gyroscope

The contact microphone we used is a Knowles BU-21771, which is connected to a pre-amplifier board which amplifies the signal by over 100 times. The output of the preamp is passed on to a laptop via a USB sound card and sampled at $44,100$Hz. The gyroscope sensor (InvenSense ITG-3200) is connected to Teensy 3.2 microcontroller board, which sends the data to the same laptop via USB. To achieve higher perforamnce on recognition, we sample the gyroscope at $3,800$Hz by optimizing the I^2C[2] communication.

Data processing pipeline

Our data processing pipeline allows the FingerSound system to capture and analyze data in real-time. The hardware components—a microphone and gyroscope—send data to a MacBook Pro laptop over its USB ports separately. A java program reads both inputs simultaneously and stores them in a readily accessible data structure. On another parallel thread, the input sound stream is continuously analyzed to detect an input gesture activity.

[2]https://en.wikipedia.org/wiki/I%C2%B2C

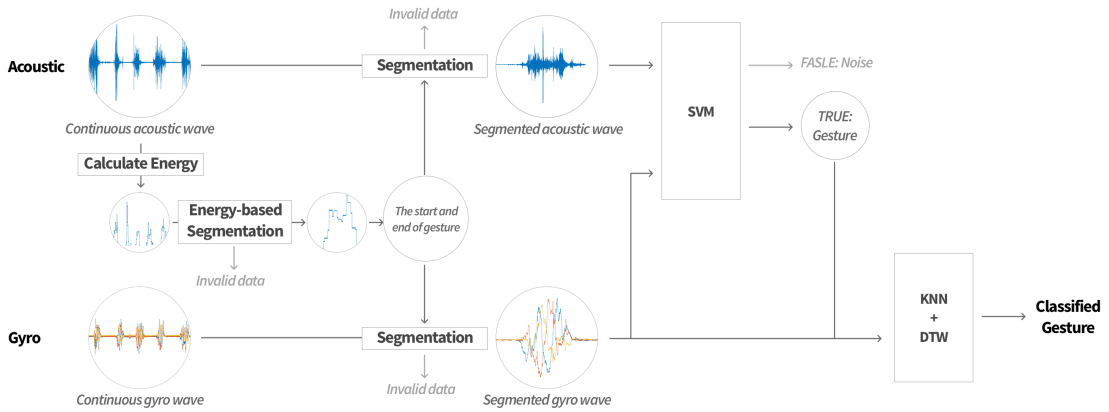


Figure 4.4: Data processing pipeline for FingerSound

To detect the start and end of a gesture, we analyze the sound produced by the grazing of the thumb on the fingers or palm. Our analysis is based on the short-term energy representation of the processed microphone signal. If the energy of the processed window is above the empirical set threshold, we set it as the start of the gesture, until the energy falls below , which we market it as the end of the gesture. If the length of the gesture is larger than preset duration, we segment that part of sound data and the corresponding gyroscope data and saves it for further processing. This segmented sound and gyroscope data is then passed through a support-vector machine (SVM) classifier to detect if the data represents a genuine gesture or noise. We extract the similar features as demonstrated in TapSkin. If the data is recognized as a gesture by the SVM classifier, then the gyroscope data is sent through a low-pass-filter and finally to our classifier which recognizes the input gesture pattern. Figure 4.4 highlights the major components of the data processing pipeline.

We employ Dynamic Time Warping based classification, which has been widely used for the analysis of time-series data in general and for gesture recognition in particular[87]. We combine DTW based sequence matching with a standard k-NN classifier (k=3) for classification. Effectively this procedure translates into very effective and efficient template matching. Our template database consists of representative examples of all relevant thumb gestures.

4.1.4 Evaluation of Digits and directional Input

We evaluated the digits and swipes input with FingerSound with 9 participants with an average age of 25.67 (3 male) on two sets of unistroke gestures—the digits 0-9 and directional swipes (see Figure 4.1)—under two settings. Before the study, two researchers provided about 100 gestures and 100 noise samples as the basic training data for building the SVM noise classifier. The actual study consisted of 2 training sessions and 6 testing sessions.

In the first two training sessions, the participants were asked to put hands and arms on the table. Each unistroke gesture was performed 3 times in a random sequence during each session. Visual stimuli on the screen and an audio cue were used to remind the participant of the gesture to be performed.

We treated the first testing session as a practice session, which helped the participants get familiar with the unistroke gesture sets as well as our experimental real-time system. The second session was used as the training data collection session for building machine learning models of gesture segmentation (SVM) and gesture classification (KNN with DTW distance function). In total, 30 (3 * 10 gestures) and 15 (3 * 5 gestures) gesture samples were collected as the training data set for unistroke digit gesture and directional swipe gesture, respectively, for each participant. The collected gesture data was combined with pre-collected data from researchers to train the SVM-based noise classifier for each participant.

After the first two sessions, each participant was required to test the system in the following 6 sessions(5 instance per gesture for each session). To investigate whether the user can perform gestures in an eyes-free fashion and in a different hands posture, we divided these 6 test sessions into two groups. In the first 4 testing sessions, the participants placed hands on a table, similar to the training session. In the last two sessions, the participants were required to hold the hands under the table to perform the gestures. These two sessions were designed to simulate the real-world scenarios where the user would likely perform gestures in an eyes-free fashion with various hand postures. The gesture recognition results

were presented to the participants in real-time on the screen. If the classification result matched the stimuli gesture, the background was marked in green. Otherwise, it turned to red. All real-time recognition results and the raw sensor data were saved for later analysis.

Result

We report the real-time classification results. The average accuracies for the first four sessions and the last two sessions are 92% and 89% respectively for the 10 unistroke digits. On average, 2.58 false-negative errors were captured in each session. The confusion matrix is presented in figure 4.5. The most accurate gestures are the digits '1','7','8', and the least accurate digits are '0', '6' and '4'.

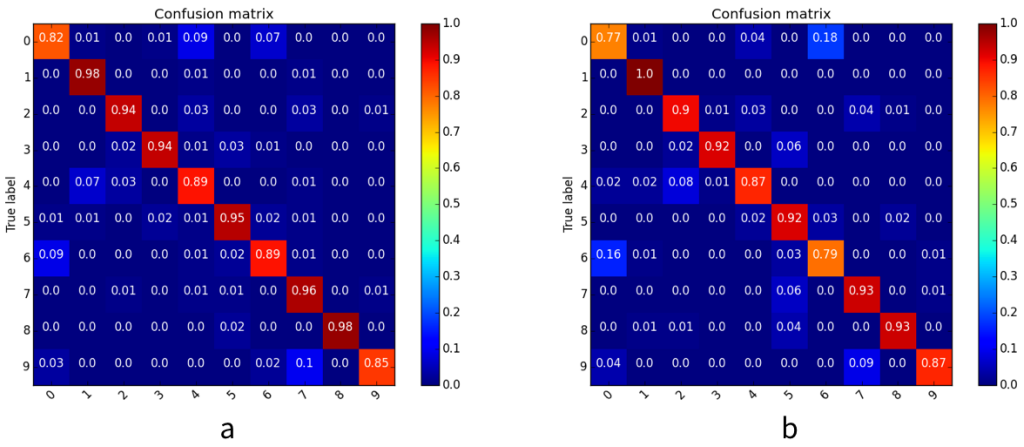


Figure 4.5: Confusion matrix for 10 digits: a. first 4 sessions where participants' the hands were on the table. b.last 2 sessions where the gestures were performed undertable in an eyes-free fashion

The average accuracies for the four directional swipes are high in general, 98.19% and 96.94% in the first four sessions and the last two sessions (eyes-free), respectively. Only 'down' and 'left' caused a few confusions when the hands were held below the table. On average, 2.74 false-negative errors were observed in each session.

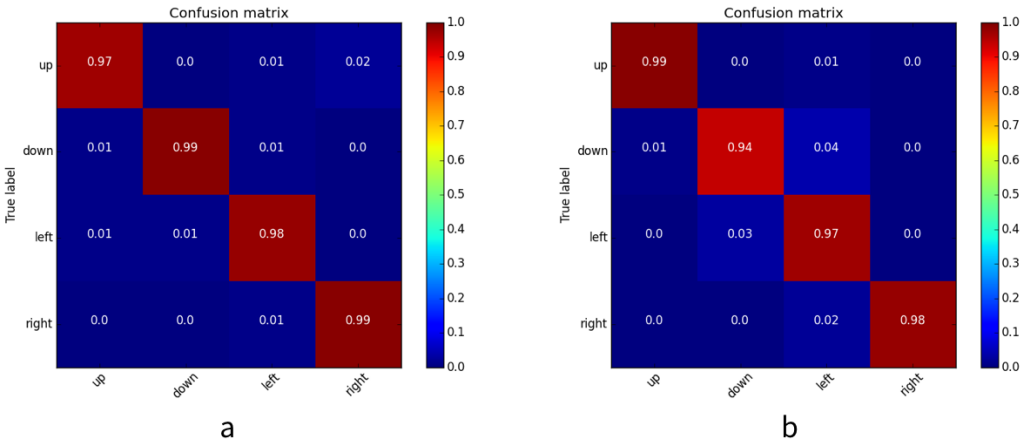


Figure 4.6: Confusion matrix for 4 directional slides: a. first 4 sessions where participants' the hands were on the table. b.last 2 sessions where the gestures were performed under table in an eyes-free fashion

4.1.5 Evaluation of Graffiti Character Input

To evaluate how user can user FingerSound to input 28 Graffiti-style letters, I conducted another study with 10 participants (including four researchers, 2 female) with an average age of 27 participated in this study. We reduced the number of testing sessions (given a large number of samples) but added one more practice session compared with the previous user study, to give the users more time to learn the larger gesture set. Overall, we had 5 sessions in this study. The first two sessions were practice sessions (3 samples per gesture per session), the third session (3 samples per gesture per session) is the training data collection session and the last two sessions (5 samples per gesture per session, hands on table only) were the testing sessions. As before, the real-time classification result was presented to the participant and recorded.

Result

The real-time classification results for recognizing the 28 unistroke gestures resulted in an average accuracy of 92.46%. There were 5.9 false-negative errors observed on average for

each session. Figure 4.7 shows the confusion matrix for this gesture set. The most accurate gestures are the letter 'X' and 'Z' whose precision are 100%. The least accurate gestures are the letter 'D' and 'P' with the precision of 69% and 74%. The reason for confusion between these two letters is visually apparent on Figure 4.2 as they look very similar. The only difference between them is where the gesture ends. The "D" ends at a lower position than where "P" ends, which appears harder to be distinguishable by intuition compared to other letters.

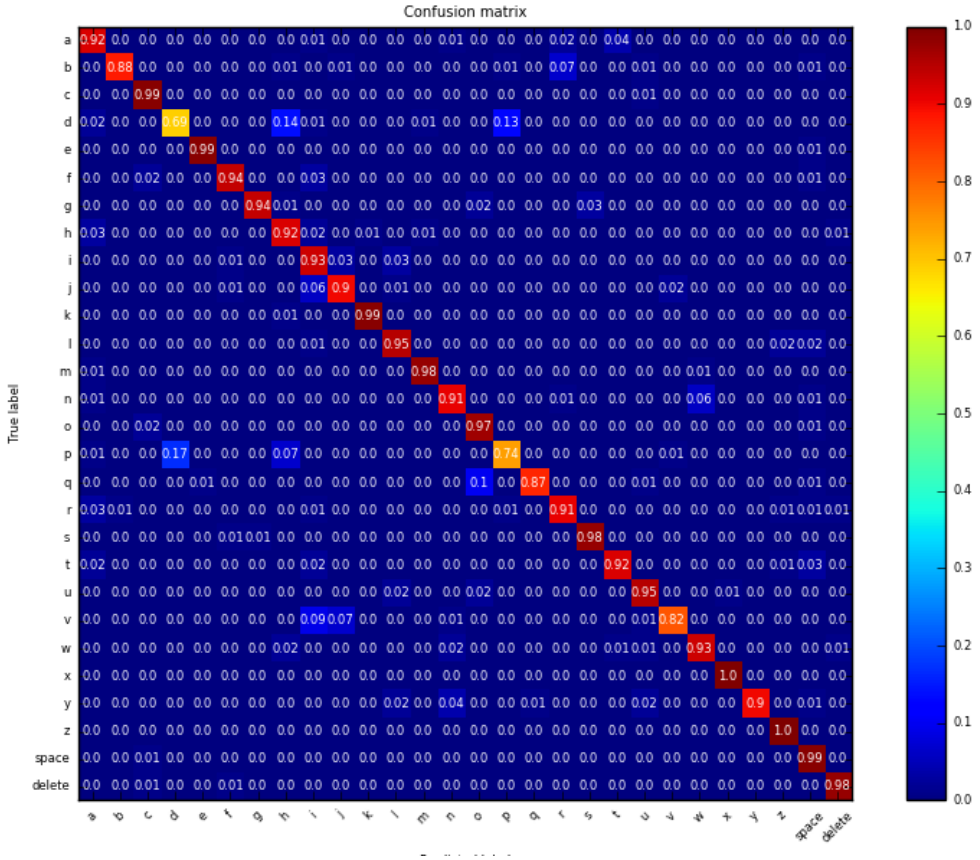


Figure 4.7: Confusion matrix for Graffiti Unistroke Gestures

4.2 FingOrbits: Interaction with Wearables Using Synchronized Thumb Movements

4.2.1 Overview

Using the similar hardware which I adopted in FingerSound, I developed another set of input gestures inspired by the recently presented Orbits system [88], which allows users to interact with displays through synchronizing their eye movements (captured using eye trackers) with moving dots on the screen. We adapt the concept of synchronized movements for one-handed input: *i)* Instead of an eye tracking system only a simple thumb ring with integrated contact microphone is required; *ii)* No exact trajectory match is required but rather only a frequency, which provides for a more flexible interaction.

This technique is called FingOrbits a concept for thumb based interaction with wearables such as heads up displays or smart watches. Through rubbing a thumb against the fingers of the same hand users perform specific input gestures.

The FingOrbit system consists of a ring worn on the thumb (details below) and an accompanying visual interface which guides users to perform gestures. The visual interface is a standard number pad consisting of digits from $0-9$ and the symbols * and # presented in the tabular format of four rows and three columns. Each row starting from row 1 (topmost) to row 4 (bottommost) is mapped to the index, middle, ring, and little finger, respectively. Each column has a horizontally moving cursor which moves within the bounds of its column (Figure 4.8-Left). The moving cursor frequencies are 2.4Hz, 1.0Hz, and 1.7Hz (from left to right). For instance, the leftmost cursor moves with a frequency of 2.4Hz in the first column between points A and B repeatedly as shown in figure 4.8. During testing, the gesture to be performed is marked by a highlighted cell. For instance, (Figure 4.8) depicts cell 1 (row 1, col 1) as highlighted. The user is expected to rub the against index finger (row 1) at a frequency of 2.4Hz matching the cursor frequency in column 1.This technique can recognize up to 12 different one-handed input gestures, which require little to no user training.

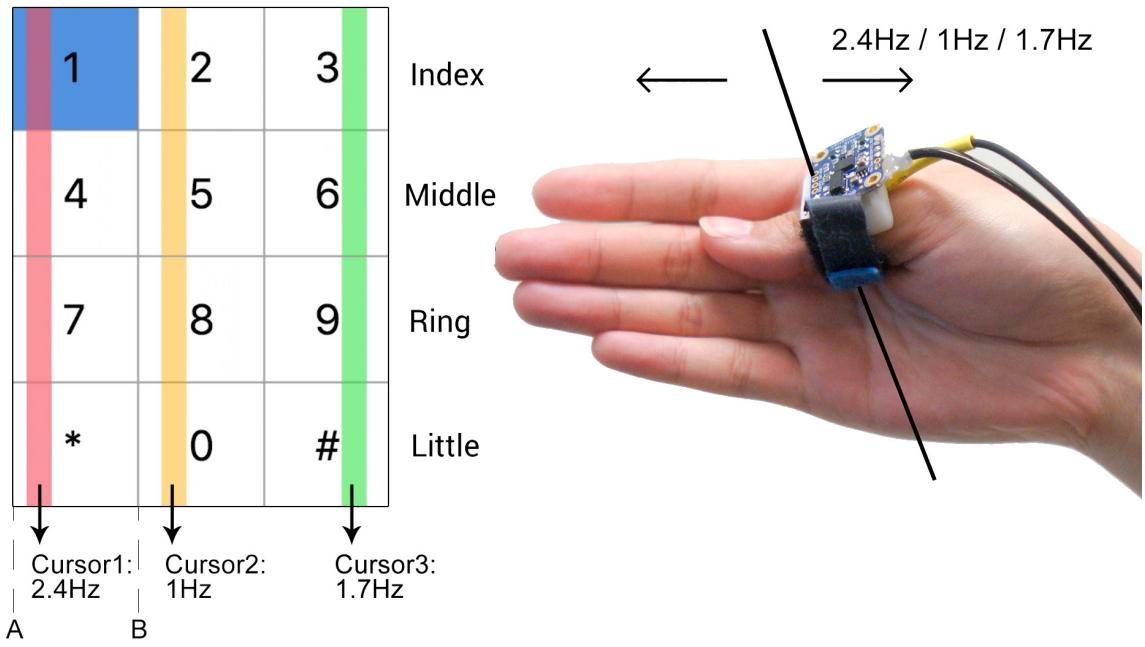


Figure 4.8: The FingOrbits system and experimental setting.

4.2.2 Implementation

Hardware

We use similar hardware as demonstrated in FingerSound. The only difference is we replace the gyroscope sensor with an Inertial Sensing Unit(Bosch BNO055), which is samplted as 200 Hz.

Data Processing

Figure 4.9 gives an overview of the data processing pipeline that is used for FingOrbits. To detect the start and end of an intended gesture, we slide a $100ms$ window with no overlap on the received sound data. For each sliding window, we first mask silent portions by applying an empirically determined threshold. Once the maximum energy of a frame exceeds an empirically determined activation threshold, we mark this frame as the start of a gesture. If the maximum energy for any subsequent, five consecutive frames (i.e., for at least 0.5 seconds) falls below the threshold, we end the gesture.

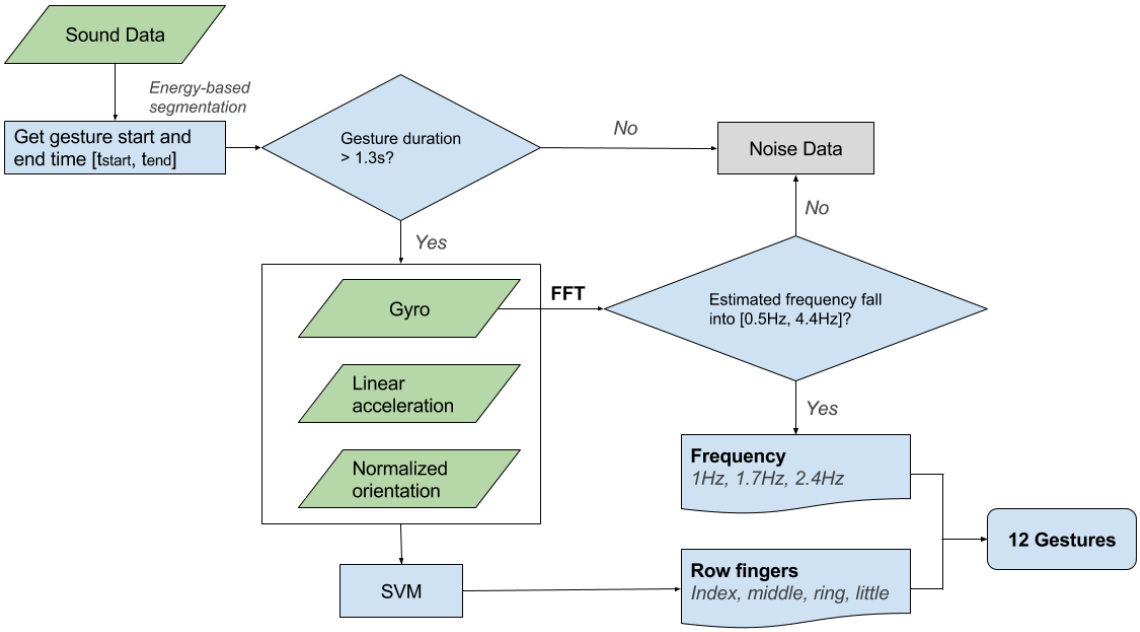


Figure 4.9: Flow chart of FingOrbit's data processing procedure.

If a gesture segment is longer than 1.3s, then we examine the dominant frequency (band) of the gyroscope data (i.e., the frequency with highest power in the whole frequency spectrogram). To estimate this frequency, we first find the axis of gyroscope data with highest energy and then calculate the FFT for the data of this axis. If the estimated dominant falls in the range $[0.5Hz - -4.4Hz]$, we label the extracted segment as a gesture and proceed to the next step. Otherwise, we mark it as noise do not advance.

The range used was determined empirically with the prototype system. Once a gesture is confirmed, we use the following steps to estimate the frequency of the thumb movement and identify on which finger the thumb is moving. First, we compare the dominant frequency of the gyroscope data to a set of preset frequencies (1Hz, 1.7Hz, 2.4Hz) and find the closest match. For instance, if the estimated dominant frequency is 1.2Hz, it would suggest that the user is performing a gesture to match the frequency of 1Hz. These preset frequencies are chosen based on the results of a formative preliminary study that aimed at identifying thumb moving frequencies that are both comfortable to perform by users and discriminative w.r.t. three different states (slow, medium, fast).

To recognize which finger the thumb is rubbing against, we use gyroscope, linear acceleration, and orientation data from the IMU. To remove the influence of body orientation on the orientation data, we normalize the raw orientation data by subtracting the mean value on each axis before further processing. For each sensor, we extract statistical features for each axis: minimum, maximum, mean, energy, variance, standard deviation, zero crossing rate, and entropy.To represent the relationship between axes, we also extract the Pearson-correlation and energy ratio between each pair of the three axes on each sensor. In total we extract 90-dimensional feature vectors (per extracted gesture segment) and feed it to a support vector machine classification back-end to determine the finger on which the gesture was performed.

4.2.3 Evaluation

We asked a total of ten participants (five males, average age: 27.8 ± 2.39, seven novices, three experienced users) to use the system and to perform thumb gestures. The general task was to match frequencies of movement patterns that were displayed on the laptop screen through rubbing the instrumented thumb against one of the other fingers.

Our study was performed in a controlled setting where participants faced a computer screen while resting their arms on the table. The system provided visual and auditory cues to assist the user with the study. During the study, randomly individual cells of the displayed grid were highlighted. Participants then had to match their thumb's rubbing frequency with the corresponding cursor frequency of the cell. Figure 4.8 illustrates the experimental apparatus.

Each participant finished three sessions – practice, training, and testing. Per practice session three repetitions of each gesture / cell were asked to be performed, whereas five iterations per gesture / cell were performed for both training and testing. Participants were guided to perform the rubbing gestures until they saw a match between the frequency of the thumb movement and the corresponding cursor. Gestures recorded during the training

sessions were used for training the analysis system (classification back-end).In both the practice and training sessions, real-time feedback of the recognized column was provided to the user by highlighting the recognized column on the screen .

The three experienced users skipped the practice session and only performed the last two sessions. Both the real-time classification results and the segmented raw sensor data were recorded for further analysis.

Result

The overall accuracy across all ten participants is 89% for recognizing all 12 gestures, where P8 to P10 are the expert users. Figure 4.10 presents the accuracy for each participant in different classification tasks. In the task of recognizing 12 gestures, P1 and P9 presents the highest accuracies of 91.7% and 100% , while P2 and P8 have the lowest accuracies of 80% and 91.5% across novice and expert users.

Disaggregation of the recognition results gives insights into the performance of the two classification steps: *i)* which finger is the thumb rubbing against; and *ii)* which gesture (frequency) is the participant performing. On average, discriminating the four fingers the thumb rubs against works with 94% accuracy. The subsequent step of discriminating the three possible rubbing frequencies works with 94% accuracy.

As the confusion matrix illustrates (Figure 4.11), the gesture that has been recognized with lowest accuracy (82%) corresponds to rubbing the thumb against the index finger with the middle frequency (1.7Hz). The lowest accuracy (89.95%) for classifying three frequencies across 4 fingers is 1.7Hz as shown in Figure 4.12a.

The ring finger was the most confused finger as shown in Figure 4.12b. In fact, most participants perceived rubbing their thumb against their ring and pinky fingers as rather uncomfortable, which explains the poor recognition performance and perhaps a greater variability of the performed gestures.

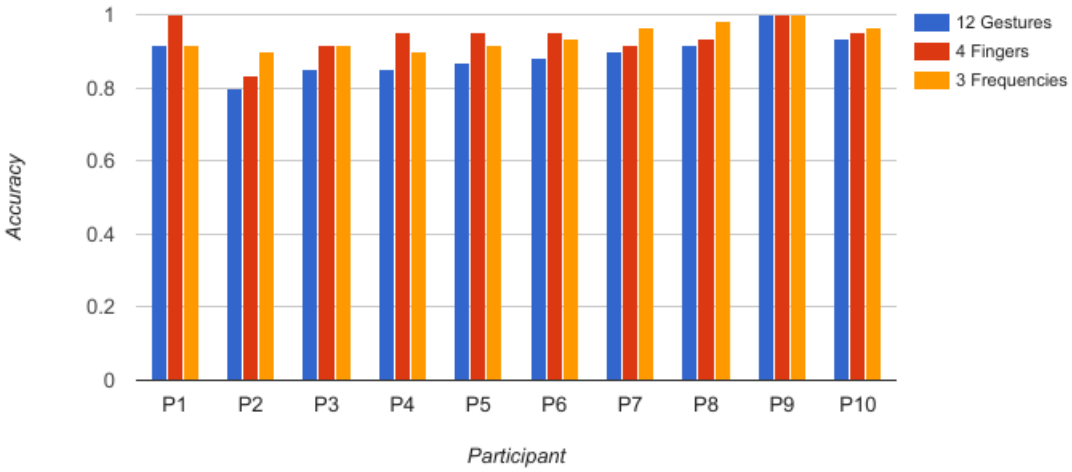


Figure 4.10: Accuracy for each participant

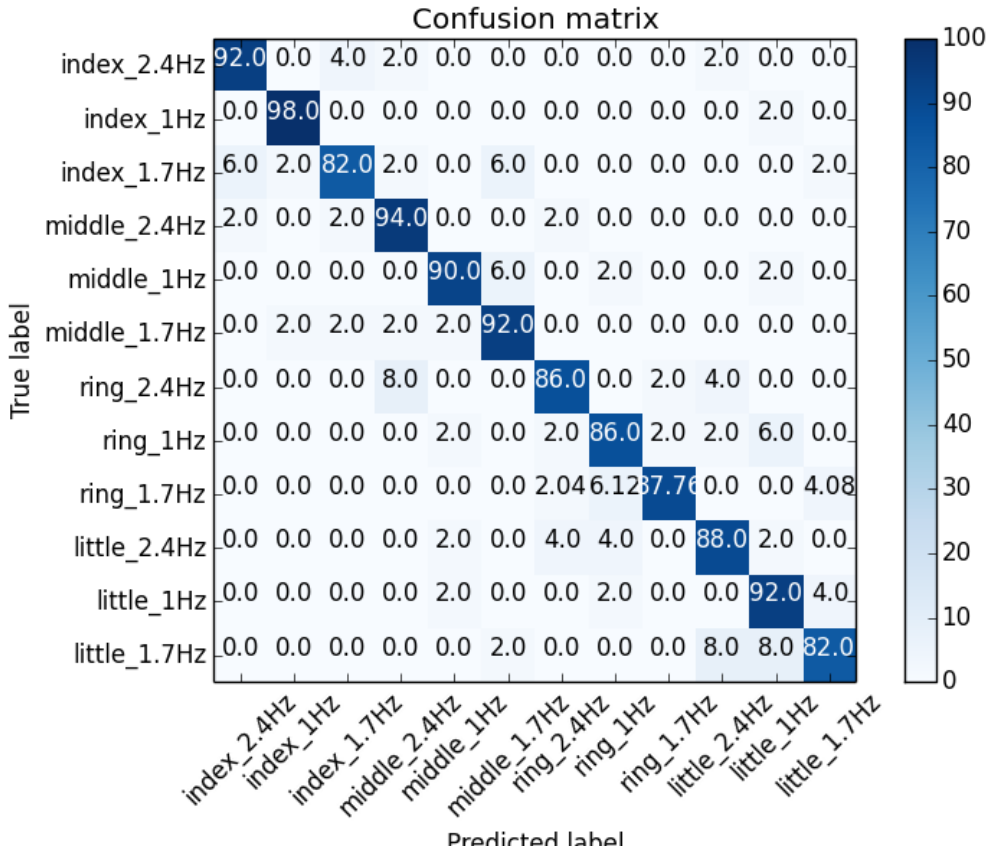


Figure 4.11: Confusion matrix for all 12 gestures

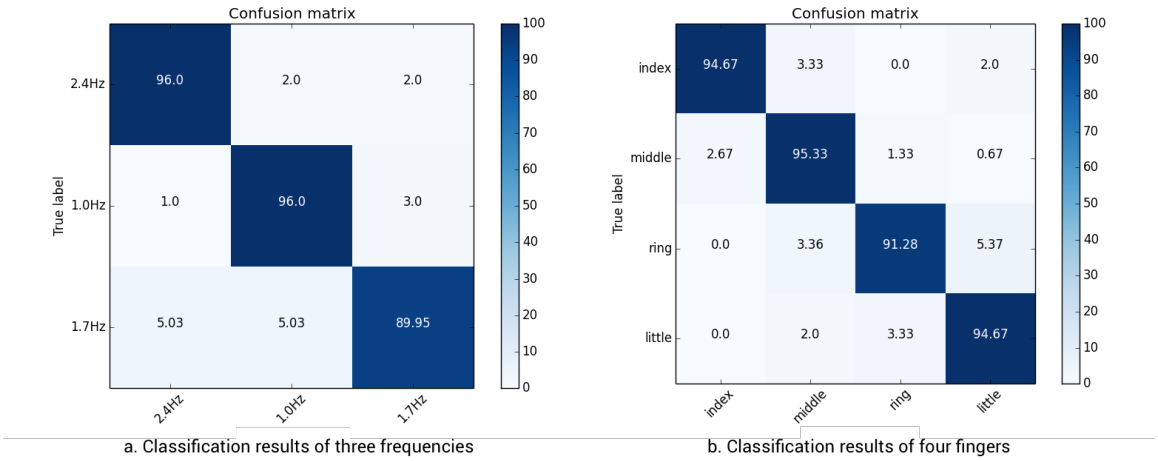


Figure 4.12: Results for discriminating frequencies and fingers separately.

4.2.4 Eyes-free interaction

To investigate how FingOrbits works when the gestures are performed in an eyes-free fashion, we conducted an additional study with our three experienced users. We followed the same procedure as for the previous experiment, except that the arm of the participant was laying down naturally under the table (in the lap), and we did not provide the moving cursor from the testing session. All three participants performed eyes-free FingOrbits gestures for the first time.

The overall accuracy was 84.4%, i.e., a moderate drop of about 10% accuracy compared to the results of the first study. The accuracy of classifying the gestures with rubbing frequencies of 2.4Hz, 1Hz, 1.7Hz and were 96.7%, 98.3%, and 83.3% respectively. A major drop was noticed for the higher frequency of 1.7Hz, which indicates that if FingOrbits is used in an eyes-free fashion without displaying the moving cursor, one should stick to only two, rather distinct rubbing frequencies.

A further analysis revealed that the accuracy of recognizing two frequencies on index finger and recognizing two frequencies on both index and middle finger are 100% and 91.7%, respectively. These preliminary results illustrate that FingOrbits has potential for real-world applications.

4.2.5 Applications

Applications on wearables may not require all 12 input gestures that FingOrbits can recognize. Reasons for a limited recognition lexicon could be usability (e.g., rubbing the thumb against the pinkie is not very comfortable for many) or simplicity that leads to lower cognitive load during interacting.In what follows we will discuss exemplary applications of FingOrbits. We will provide recommendations for gesture sets to be used and –in light of this– link back to the results of our user study.

Number Input

Current interfaces for entering numbers (digits) into heads-up devices such as Google Glass require the use of a paired device (e.g., a smartphone), which slightly undermines the general idea of wearables. FingOrbits could be used to facilitate numerical input through utilizing 10 rubbing gestures while the glasses display the moving cursor.Based on our current apparatus it would be advisable to represent the digits $1 - -9$ using the upper three fingers and the three different frequencies each, and to map the digit 0 to 1Hz movement on the pinkie.Such a setting would result in over 90% recognition accuracy.

Music Player

Interacting with a music player can be reduced to three to four functions such as next song, previous song, pause, play. According to our user study using the index and middle fingers as interaction target for FingOrbits results in high classification accuracy – whilst at the same time being most comfortable to use. As such the combination with the lowest and the highest frequencies would provide a reliably recognizable gesture set with average classification accuracies of the four functions to be performed at about 95% (according to our user study). Further variations of three to four functions of a music player facilitate through FingOrbits are possible.

Quick response to notification

Many applications only require binary input functionality such as accepting or declining phone calls, or turning on/ off a notification. For such scenarios FingOrbits should be used with the index and middle finger and the lowest and highest rubbing frequencies (1Hz and 2.4Hz, respectively). According to our user study such a configuration would lead to almost ideal classification results (99% without collecting any training data from the user).

4.3 FingerPing: Recognizing Fine-grained Hand Poses using Active Acoustic On-body Sensing

4.3.1 Overview

The previous two input technology presented in this chapter enable the user to input by dragging thumbs across fingers for a convenient and socially appropriate wearable input solution. In this subsection, sharing the similar motivation, we present FingerPing [89], a novel wrist- and thumb-mounted sensing solution to enable one-handed input. FingerPing relies on detecting various hand configurations (e.g., the thumb touching the tip of a finger) based on how that configuration impacts the propagation of sound waves injected at the thumb and propagating around the hand. The human body is a good medium for sound propagation [82, 13], and its frequency response varies based on which path a sound wave travels through the body. A change in hand configuration, which results from forming the hand into a variety of different poses or gestures, creates sufficiently distinct propagation paths for sound waves. To utilize this phenomenon for recognizing different hand poses, FingerPing injects acoustical chirps (20Hz–6kHz) ten times per second from the base of the thumb. These chirps travel through the hand and are received by microphones present on the thumb and wrist. The received signals are then classified to match a set of known poses.

To demonstrate the capabilities of FingerPing, we designed and evaluated two sets of

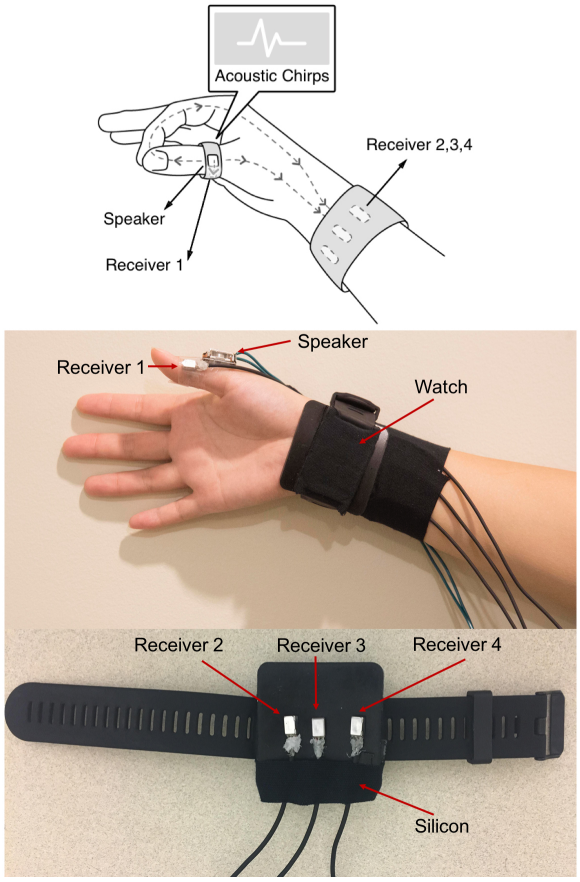


Figure 4.13: FingerPing

poses. The first pose set consists of thumb taps to the 12 phalanges across the four fingers, which can be used for number input and potentially text input with a T9 keyboard[3]. The other pose set consists of the ten number poses from American Sign Language, further demonstrating the flexibility in reliably distinguishing a large number of simple hand poses. Note that our system effectively detects endpoints of gesture input. However, the actual data analysis –after segmentation– is based on classifying *static* hand configurations (poses). Consequently, we denote our approach as pose or hand configuration recognition rather than gesture recognition – even though its purpose is clearly targeting the latter. Our technology may suffer more false-positive errors caused by the on-body acoustic noise during daily activities. However, false-positives from daily activities can be addressed with a reliable activation pose. The user could perform the activation pose to activate the system which would then enable the full set of poses for recognition. Our technology may also request additional calibration for different users. A user calibration procedure can be adopted to address this issue.

4.3.2 The design of hand poses

In this part, we demonstrate the capability of FingerPing by recognizing two sets of hand poses.

The first set of poses enables the user to input digits by touching any of the 12 phalanges of the index, middle, ring, and little fingers (see Figure 4.14) with the thumb. Because of the similarity between the layout of the 12 phalanges and a number-pad, this layout could be used to input digits or potentially input text using T9 keyboard without the heavy mental efforts to memorize different gestures.

The second set of poses are the numbers '1' to '10' from American Sign Language as shown in Figure 4.15. We use this set of poses to demonstrate the potential of FingerPing to recognize various other poses of the hand. In addition, recognizing this set of hand

[3]https://en.wikipedia.org/wiki/T9_(predictive_text)

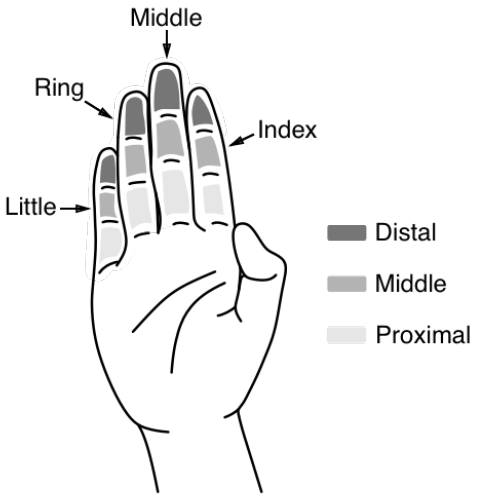


Figure 4.14: Tap on 12 Phalanges

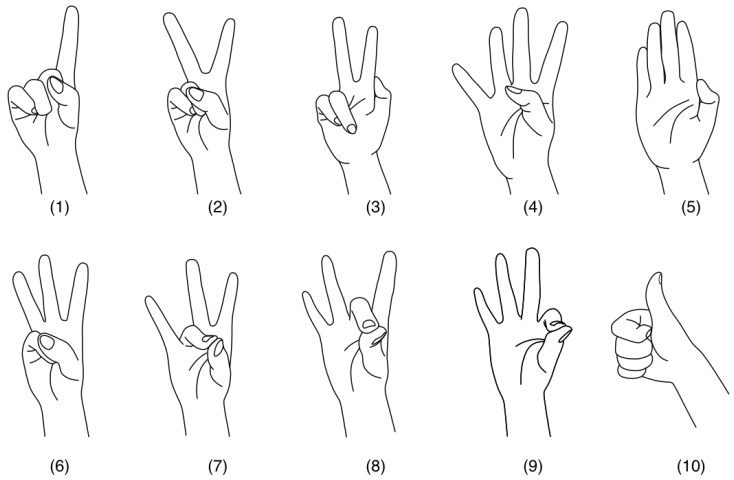


Figure 4.15: Digits '1' to '10' from American Sign Language

poses can potentially be used to translate the digits expression from American Sign Language(ASL) for people who do not understand ASL. For instance, the recognized results can be played in audio to assist the communication. Furthermore, these poses can also be used as shortcuts to access different functions in wearable computers.

4.3.3 Theory of Operation

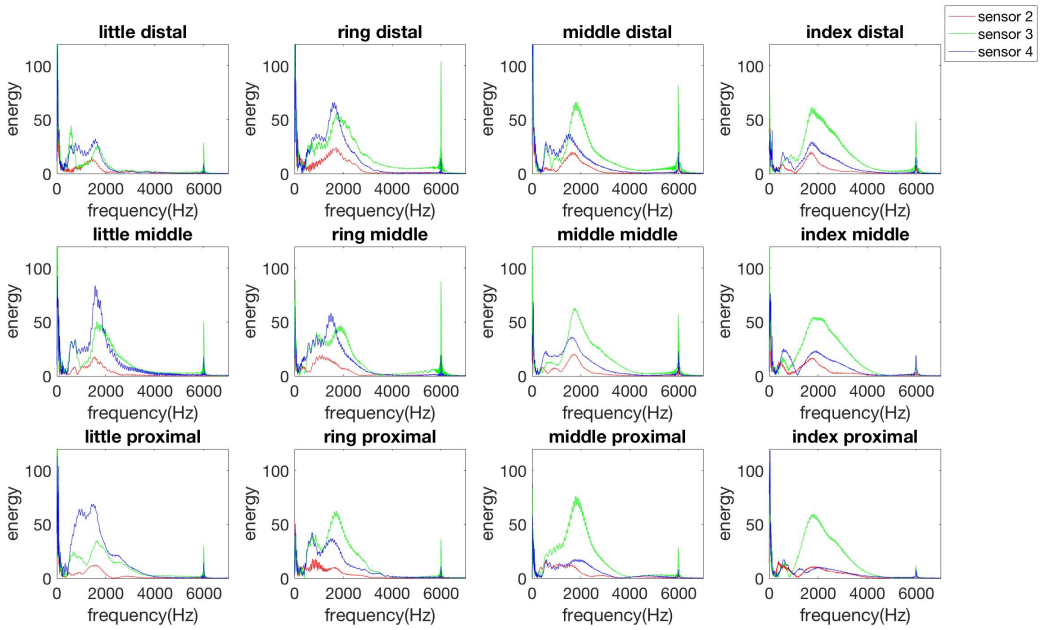


Figure 4.16: Frequency responses of taps on 12 phalanges

FingerPing exploits the physical phenomenon that the spectral properties of sound waves change based on the paths they travel between sender and receiver. Performing different hand configurations has an impact on how the sound travels through the hand. For instance (Figure 4.13), when the thumb is open (not touching any fingers), there is only one major path the signal can travel from the speaker on the thumb to the receivers on the wrist, namely the direct path. Once the thumb touches one of the phalanges, a secondary path is created for sound propagation: starting from the speaker, via the touched phalanx and the corresponding finger, and finally to the receivers. In this case, the signals received would stem from at least two major paths: *i)* the direct path; and *ii)* the ”detour” path, which goes through the phalanx and the finger. Depending on which phalanx is touched, the second path– the ”detour”–changes, which also changes the energy of different frequencies. They can be either amplified or reduced depending on the amount of tissue/ bone in the path taken by the wave. Different components (e.g., tissue, bone) present different acoustic frequency

response. This property of sound wave propagating through the human body constructs the unique fingerprints in frequency response for different hand configurations. The same phenomenon can also be observed for non-touch poses as shown in Figure 4.15, where the second path (detour path) varies depending on which hand pose is performed. Figure 4.16 shows the frequency response of chirps received from three receivers on the wrist area for the first pose set (digits; explanation below).

4.3.4 Implementation

Hardware Design

FingerPing consists of two parts of hardware: *i)* A surface transducer for emitting sound (sender); and *ii)* Four contact microphones –receivers– that capture sound signals after they have traveled through the user's body. In what follows we will describe both components in detail.

The surface transducer[4] is used to emit sound into the hand. The transducer is driven by a function generator (Agilent 33500B) and attached on the thumb using Kinesiology tape which is stretchy and elastic. We use a function generator to send ten chirps of 2 Vpp per second. For each chirp, we first linearly sweep the frequency range from 20Hz to 6,000Hz for 0.05 seconds, and then hold on at 6,000Hz for another 0.05 seconds as shown in figure 4.17. The range for the frequency sweep was optimized in an experimental evaluation (results not shown here) that unveiled that frequencies up to 6,000Hz retained maximum information while propagating in the body, which is also in line with previous findings in the literature (e.g., [82]).

The second part of the hardware includes four contact microphones (Knowles BU-21771) used to capture the signals from the body, each of which is 7.92 mm by 5.59 mm by 4.14 mm in size. These contact microphones provide a low noise floor and very flat frequency response. One of the microphones is attached on the thumb and the remaining

[4]https://www.sparkfun.com/products/10917

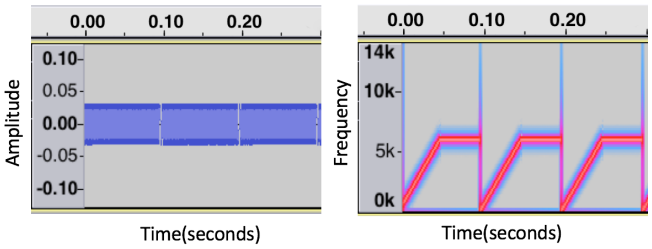


Figure 4.17: Sweep Signal

three are aligned on the wrist as shown in Figure 4.13. We built a watch-like device to attach the microphones to the wrist of the wearer. A piece of silicone is placed between the watch case and the microphones to fix the positions of the sensors and match the acoustic impedance as demonstrated in [90], which contributes to a better quality of the captured acoustic signals.

All microphones are connected to pre-amplifiers to amplify the received signals (factor: 100). The amplified signals are then relayed to a Macbook Pro laptop computer for data processing via an audio interface (Fireface 800). The audio interface samples the audio at 44,100Hz. The laptop runs a software program written in C language using the PortAudio library to communicate with the audio interface. The data read from the audio interface is then sent to a Java program for real-time processing over network socket.

Data Processing Pipeline

The processing of the received data stream (four channels) can be divided into three steps as described below and shown in figure 4.18. Channels 1 to 4 of the audio interface are mapped to the signals from the four microphones, respectively.

Chirp Localization: In order to recognize a given pose, we first explicitly localize (temporally) each chirp within the continuous data stream – segmentation. Channel 1 is the signal from the microphone right next to the speaker. The amplitude of the chirp is highest here and least influenced by potential other noise. We perform peak localization on the audio signal of this channel by finding the maximum absolute amplitude, and then segment

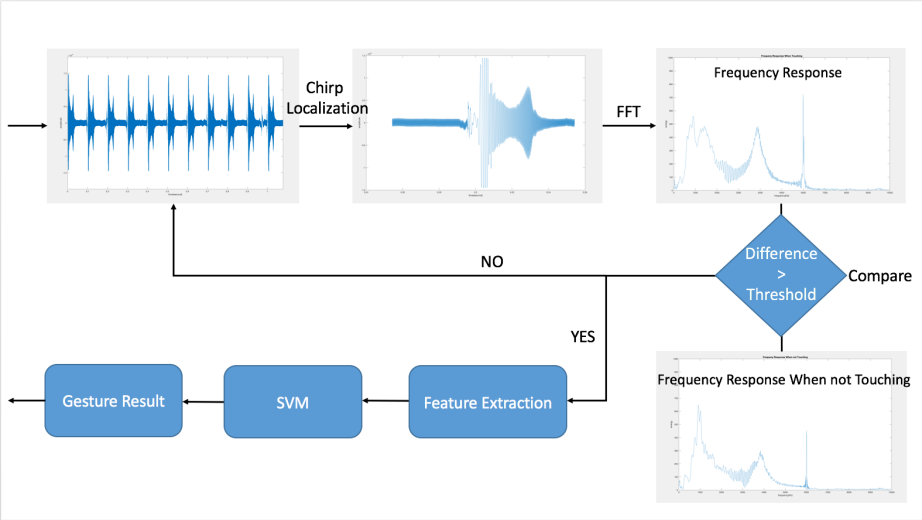


Figure 4.18: Data Processing Pipeline

the chirps from all four –synchronized– channels using a window size of 0.046 seconds (2,048 data points to facilitate the use of Fast Fourier Transformation) centered at the peak position extracted from the first channel.

Pose Segmentation: Pose segmentation is based on comparison to a reference signal. This reference signal is recorded during system start when we ask a user to hold their hand still and open –that is to not perform any pose– and record the received signals from all four microphones as reference.

For each segmented chirp, we perform Faster Fourier Transform (FFT) to extract the energy distribution across frequency 0-10kHz. To detect whether a pose is performed, we calculate the Euclidean distance of the FFT results between the current chirp and the reference chirp recorded during system start as described above. If the distance is larger than an empirically determined threshold, we infer that a pose is being performed. We then use the subsequent 0.5 seconds of data from all four channels for the hand pose recognition. This data is also saved for post-analysis.

Pose Recognition: In the second step, we extract features from each chirp collected from the four channels. For each chirp, we first pass it through a band-pass filter (100 Hz – 5,500 Hz), which is the most informative frequency range based on early exploration.

On the filtered chirp. we then extract 35 features, namely: zero crossing rate, energy, entropy, spectral centroid, spectral flux, spectral chroma, spectral roll-off, and Mel-frequency cepstral coefficients [91]. Then we extend the feature vector to 294 by adding the dominant frequency and its energy, as well as spectral energy bins from 100 Hz to 5500 Hz as extracted through the FFT. Finally, we combine the feature vectors of all four channels resulting in a global descriptor of dimensionality $d = 1,176$, which is then fed into a support vector machine pose classification backend. We use the sequential minimal optimization (SMO) implementation of SVM provided by Weka [92].

Since the pose segmentation step actually sends data segments of a length of 0.5 seconds for pose recognition and each chirp takes 0.1 seconds, each channel may contain up to 5 chirps. The final recognition result of a particular pose is thus based on majority voting over the five individual chirp classifications.

4.3.5 User Study

To evaluate the performance of FingerPing and to understand how users would use our system, we conducted a user study with 16 participants (10 male; average age of 26.6). 8 randomly selected participants were requested to test the first pose set (tapping on 12 phalanges) and the remaining 8 participants were instructed to evaluate the second set of poses (digits '1' to '10' from American sign language).

At the beginning of the study, a researcher first introduced and demonstrated the poses that the users were required to perform. Then the researcher helped the participant to put on the system hardware on their hand and wrist. Each participant was allowed to practice each pose before they proceeded to the actual test. Participants were sitting on a chair duing the study.

The study consists of seven separate sessions. During each session the participants were given visual prompt for the pose to be performed. The first session was a practice session, where each participant was instructed to perform all the poses in a random order

with five instances per hand pose. Sessions 2 through 5 were used for collecting the training data, in each of which the participants were requested to perform each pose in a random order with five instances per pose. No feedback was given for these first 5 sessions. The last two sessions were used as testing sessions, where the participants were instructed to perform each pose five times in a random order. Unlike the practice and training sessions, the participants were given feedback for real-time classification results. If the classified pose was recognized as the one the participant was asked to perform then the interface showed an icon of green (red otherwise). Manual ground truth annotation was provided by observing researchers during the user study.

Results

We removed the mis-performed poses caused by the participants from the final analysis, where participants failed to perform the hand pose as the stimuli indicated. We dropped 87 out of 1760 instances (22 gestures * 80) from 16 participants. As a result, after removing mis-performed poses, there are 1.5 false positive errors in average in each testing session for both two pose sets. The average accuracy across all participants for the phalanges pose and American sign language are 93.77% and 95.64% respectively. The confusion matrix for the two poses sets is shown in Figures 4.22 and 4.23, respectively. For 12 phalange poses, recognition was most accurate for touch events involving the little finger, least accurate when the middle finger is targeted, which can be explained through the very similar path propagation compared to both index and ring finger. The other common confusion exists between the similar positions in the neighbor fingers. For instance, 6.58% little-distal was misclassified as ring-distal and 6.67% middle-middle was misclassified as index-middle. Figure 4.21 shows the accuracy for 12 phalanges respectively.

For ASL poses, number '10' presents the highest accuracy of 100% and number '5' has the lowest accuracy 90.54%. We think the reason why '10' is the most accurate pose is that it is very different from any other poses as figure 4.15 shows. We also noticed there were

much confusion between poses '7','8' and '9', which look similar in shapes. Figure 4.19 and figure 4.20 show the accuracy for each participant on phalanges and ASL poses. It shows that participants in general achieved a relatively high accuracy on recognizing ASL hand configurations. Also, we received some participants reporting tiredness at the end of the study. This was an issue of the study setup, which required the participants to keep focused on performing different hand poses for around 30 minutes, which is rarely the case in daily scenarios.

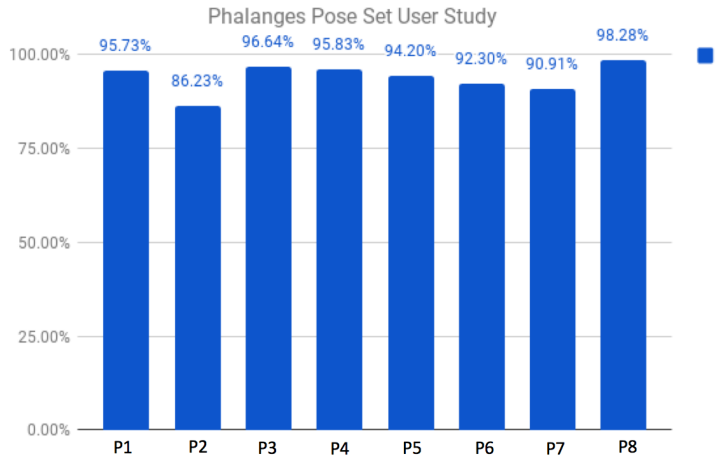


Figure 4.19: Accuracy for each participant on Phalanges poses

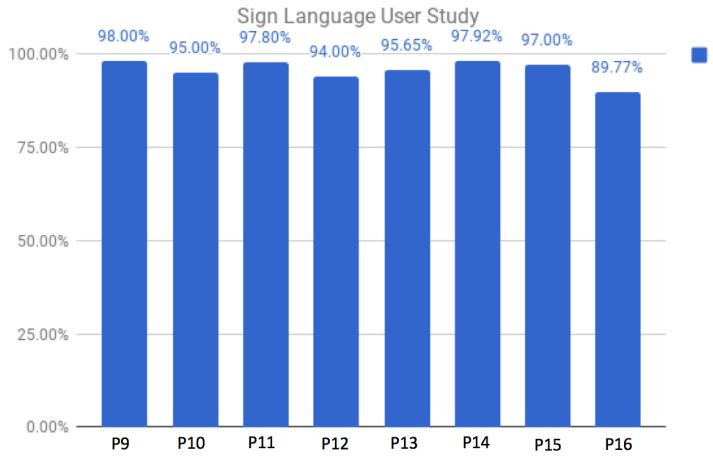


Figure 4.20: Accuracy for each participant on ASL poses

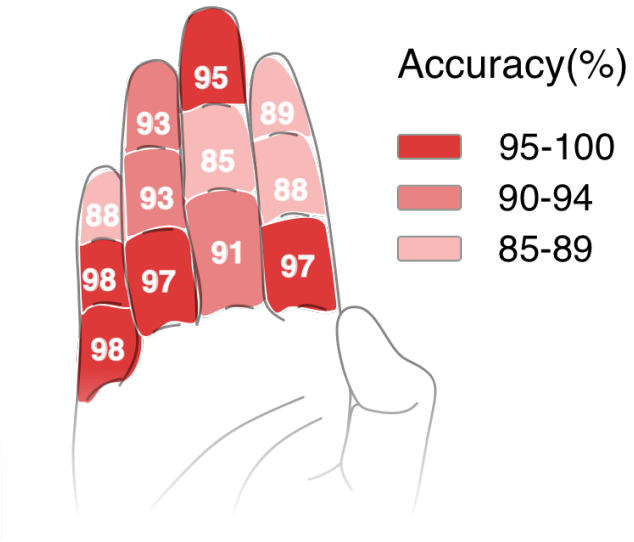


Figure 4.21: Accuracy for 12 Phalanges (Rounded to nearest integer)

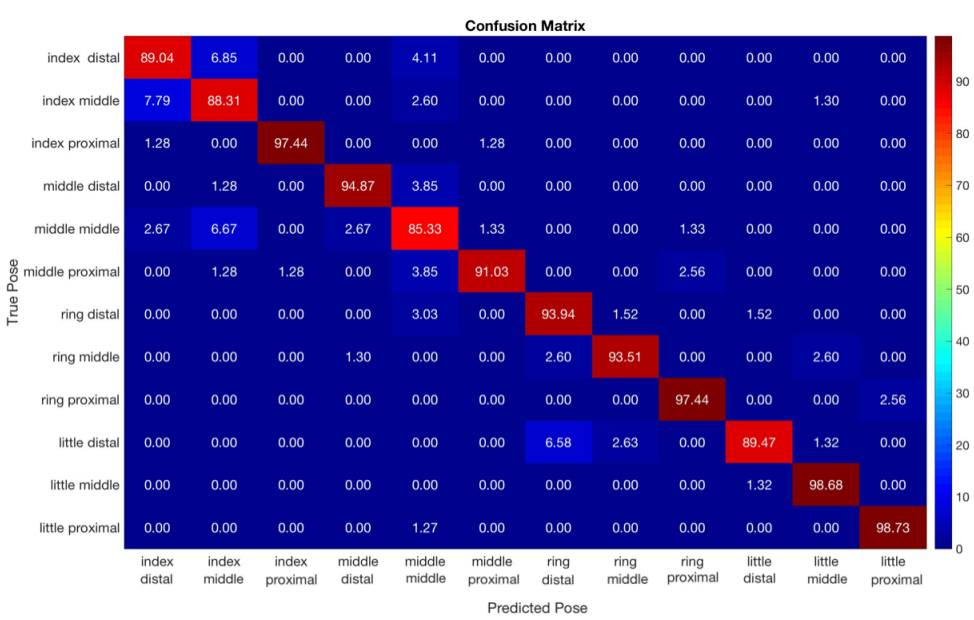


Figure 4.22: Confusion Matrix for recognizing touch events related to the 12 phalanges.

4.3.6 Discussion

Applications

One obvious application for the phalange pose set is to use them directly for a number input system due to structural similarity and human intuition. If the layout is mapped to a

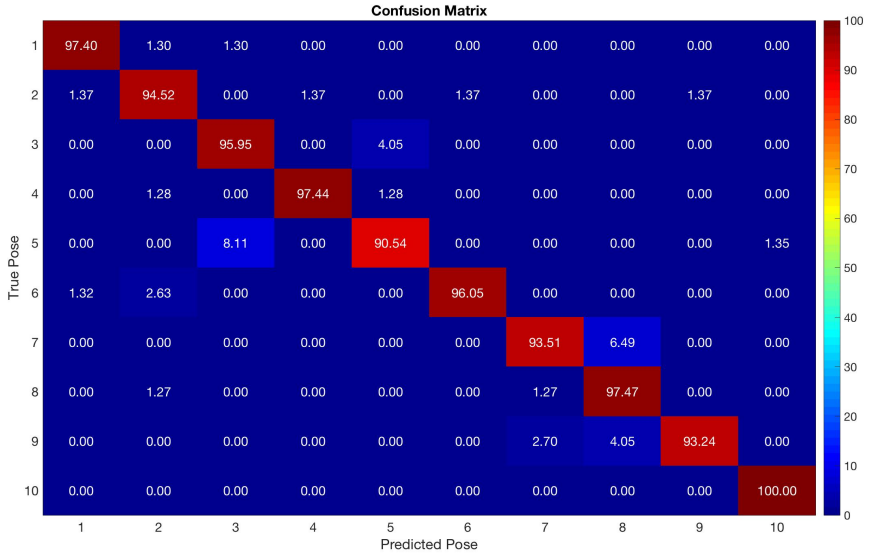


Figure 4.23: Confusion Matrix for recognizing 10 poses from American Sign Language.

T9 keyboard, it can be extended to be used as an input system for text as well.

Depending on the applications, these poses can be combined or selected to form new pose sets which may influence the recognition accuracy. Many applications do not need to have all 12 poses to be functional. For instance, to control a music player, generally only four buttons (next song, previous song, pause, play) are needed, which can be mapped with the distal of the four fingers in phalange pose set with an accuracy of 93.69% in post-analysis using the data collected in the user study. Similarly, we can also use a subset of the phalange poses to control a D-Pad , which is very helpful to navigate through menus with hierarchy on wearables (e.g., Google Glass, smart watches). The buttons of D-Pad can be naturally mapped to the index-middle, middle-distal, middle-proximal and ring-middle positions. Another interesting mapping can be to use the four corner phalanges (index-distal,index-proximal,little-distal,little-proximal), which presented 94.55% accuracy in our post-analysis. These four poses can be used to control music player or shortcuts to applications.

Hardware Improvement

In the current hardware setting, the surface mounted speaker is directly taped on the thumb. However, we actually built different shapes of 3D-printed rings (hard plastic material) to attach the speaker to the thumb. Unfortunately, due to the relatively large size and the rigid shape of the speaker, this solution did not fit well with everyone's hand. One potential solution in the future is to 3D-print rings with flexible material (e.g., rubber), which would be adjustable for different thumb sizes. Another drawback of taping the speaker on the thumb is the airborne communication. In the study, the chirps are audible in the quiet study room. Future design of the ring should consider isolating the speaker to airborne communication by wrapping it up with sound-absorbing material, such that the speaker would not generate much noise to the environments surrounding the participant.

Addressing false-positives

The current implementation utilizes a threshold-based segmentation method to detect the start of the interaction, which can be prone to noise in tough scenarios, such as when the user's hands are touching different objects. One way to address this, is to apply more advanced machine learning pipeline (e.g., hidden Markov model) to automatically transit between different states with a much larger set of training data. The other possible solution is to introduce an activation pose. For instance, to activate the system, the user needs to tap on three phalanges on index finger in a certain, specific sequence, which is hard to trigger by accident. Therefore, the system stays inactive during regular activities and would only be active once the user chooses to perform the activation pose. The activation pose can be selected from the most distinguishable poses such as '10' in the American Sign Language.

Calibration

One limitation of the current system is its potential user or session dependency. A user may need to provide reference data during system start. The reason for this is that the composi-

tion (tissue, bones) of the human body is different for each person, which impacts acoustic frequency response. Certainly, with large amounts of training data from huge user cohorts one could create a generic, user-independent (baseline) model. Utilizing more training data, is to attempt a user adaptive system, where we match a few training gestures made by the user to similar-seeming past users and load in those users' training data (or have a user independent model where we re-train with the addition of the new training gestures, weighted highly). Going further, we may discover a calibration process can minimize the per session or per user differences. The calibration process could be incorporated into the use of device. For example, we might require the user to perform a few poses, explicitly chosen to represent the space of input, to "pair" the device to a phone - for example, 6, 9,1, and 10 (thumbs-up). Alternatively, instead of a single pose, we can require that valid input requires a sequence of poses (i.e., a gesture), selected so that using the change in features for recognition is more session/user independent. Also, sensor placement consistency may be improved by well-designed form factors in the future. More experimentation is needed to prove the effectiveness of these approaches. In this current research, we focused on the general proof-of-concept and left the refinement to a user-independent (baseline) model for future work.

Limitation

Another limitation of the current system is its user dependency. A user needs to provide reference data during system start. The reason for this is that the composition (tissue, bones) of the human body is different for each person, which impacts acoustic frequency response. Certainly, with large amounts of training data from huge user cohorts one could create a generic, user-independent (baseline) model. However, given that the initialization is very short and would only be required when a user first puts the sensing device on, in this current research we focused on the general proof-of-concept and left the refinement to a user-independent (baseline) model for future work.

CHAPTER 5
CONTINUOUSLY TRACKING ON WEARABLES

In the previous chapters, I presented projects that enrich wearable input by providing comprehensive sets of input gestures using different form factor and sensing modalities under different scenarios. However, these techniques are not designed to replace the current input device (E.g., touchscreen), but to enrich the current input vocabulary. Many constraints of touch-based input still exist, such as the limited input area. In order to address these challenges, a new interaction paradigm such as these in previous chapters that track the user's input attention is needed.

I think allowing the user to input in the 3D space around the wearable is one of candidate solutions to replace the touch-based input. In this chapter, I described two technologies: SoundTrak and OriTrak. The first one is a 3D input solution I developed for wearables using active acoustics. The latter one provides continuously input for wearables using relative orientations.

5.1 SoundTrak: Continuous 3D tracking of a finger using active acoustics

5.1.1 Overview

SoundTrak is a novel approach based on active acoustic sensing to expand the human-wearable input operation space from two to three dimensions. Our proposed method hinges on the use of a micro-speaker on the finger or entity to be tracked (e.g., a smart pen), and an array of low-cost microelectromechanical systems (MEMS) microphones placed on a wearable device (such as a smartwatch). By tracking the accumulated phase shift of the measured signal on each element of the MEMS microphone array, we can compute the distance between the finger with the micro-speaker and the microphone array in 3D space

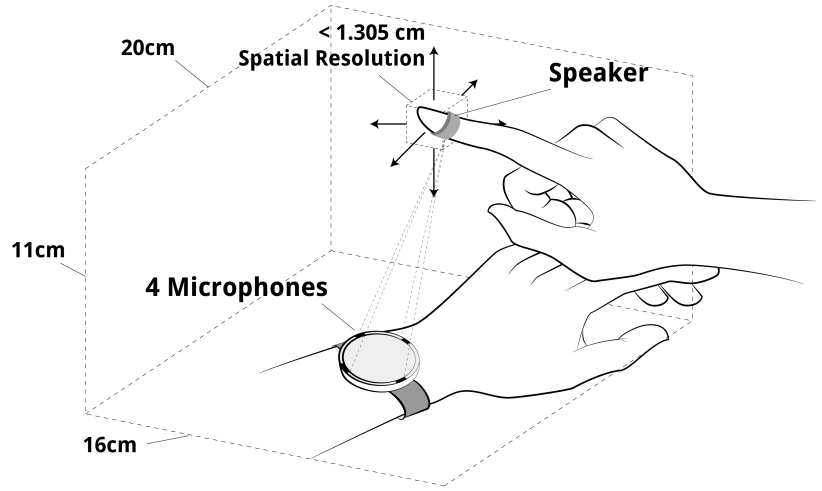


Figure 5.1: The SoundTrak System

in real-time. SoundTrak calculates the absolute x,y,z displacement of the finger to the array with a derived physics model. Thus, no machine learning training is required prior to using this system. Based on our system evaluation, SoundTrak increases the interaction space from 5 cm^2 (the average area of a smartwatch screen) to 3520 cm^3 (the volume around the device), with an average spatial resolution of 1.3 cm, as shown in Figure 5.8. Our technique can be used to expand the design space not only for wearable devices (e.g., smartwatches, Google Glass, virtual reality devices) but also for many other surfaces (e.g., tabletop, blackboard).

5.1.2 Theory of Operation

SoundTrak is built upon a well-understood physical relationship: the further the distance that a sound travels, the more delay is introduced on the receiver side because of the speed of sound. This delay can be accurately detected both by directly measuring time-of-flight or by examining the phase shift of the signal. The time delay introduced in sound propagation has been used previously to detect the distance from a sound source to a receiver in many

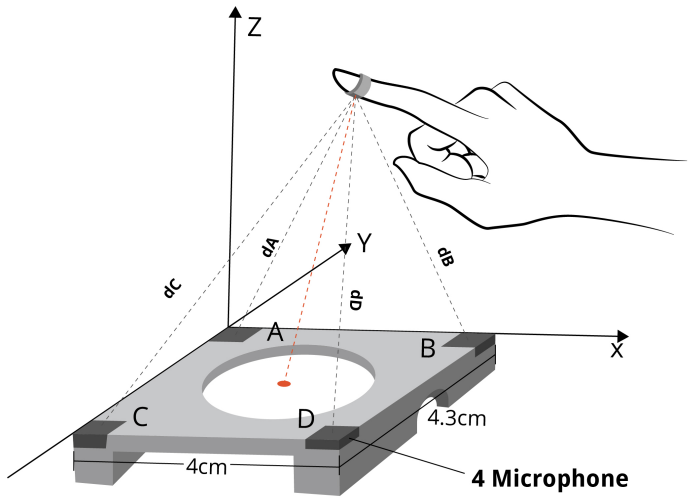


Figure 5.2: The physics model

different scenarios, for example in sonar-based location detection in 2D space [34].

Some research projects calculate the location of an object using the delay of an ultrasonic pulse or chirp. However, to retrieve the delay of the chirp, the sender needs to communicate the sent time of each pulse or chirp to the receiver, which may be challenging between wearable devices. Because such a communication may require additional hardware, it would increase the size and weight as well as the power consumption of a wearable device. In addition, to distinguish the consecutive pulses or chirps on the receiver, a chirp-based method may have limits on the number of pulses or chirps can be sent per second.

SoundTrak is a new 3D position calculation method that tracks the accumulated phase shift of a received acoustic signal. It does not require communication between the speaker and receivers because it can derive the signal distance exclusively at the receiver using phase information. Thus, it is possible to build a miniature device to transmit a single frequency acoustic signal with relatively low energy consumption.

In the following sections, we will present the underlying model and a detailed algorithm to implement the physics model.

Geometric Model

Figure 5.2 shows the geometric relationship in 3D space between the finger-mounted speaker and a custom designed rectangular-shaped watch case. We use A $(x_a, y_a, 0)$, B $(x_b, y_b, 0)$, C $(x_c, y_c, 0)$, and D $(x_d, y_d, 0)$ to denote the corners of the watch case where the receivers are placed, and use A $(x_a, y_a, 0)$ as the grid origin of the coordinate system. The distance from the speaker on the finger to each receiver is denoted as dA, dB, dC, and dD. SoundTrak is built to calculate the coordinates (x,y,z) of the finger, which represents the finger's position in 3D space around the watch.

$$(x - x_a)^2 + (y - y_a)^2 + z^2 = dA^2 \tag{5.1}$$

$$(x - x_b)^2 + (y - y_b)^2 + z^2 = dB^2 \tag{5.2}$$

$$(x - x_c)^2 + (y - y_c)^2 + z^2 = dC^2 \tag{5.3}$$

$$(x - x_d)^2 + (y - y_d)^2 + z^2 = dD^2 \tag{5.4}$$

Based on Figure 5.2, we can derive equations 5.1, 5.2, 5.3, 5.4 to represent the geometric relationships between locations. We need to derive the value of x,y,z using these equations. If we assume all the other variables are known in these equations, x,y,z can be calculated using any three of the four equations. There are four combinations of the three equations: 1 2 3, 1 3 4, 2 3 4, 1 2 4. For example if we choose the equations 5.1 5.2 5.3 to solve the value of x, y, z, we get get equations 5.5 5.6 5.7. In these equations, $x_a, y_a, x_b, y_b, x_c, y_c$ are all preset constants, which represents the coordinates of receivers. Solving equations 5 6 7 gets us the location of the finger.

$$x = \frac{(y_c - y_a)(dA^2 - dB^2 - y_a^2 + y_b^2 - x_a^2 + x_b^2) - (y_b - y_a)(dA^2 - dC^2 - y_a^2 + y_c^2 - x_a^2 + x_c^2)}{2(x_b - x_a)(y_c - y_a) - 2(x_c - x_a)(y_b - y_a)} \tag{5.5}$$

$$y = \frac{dA^2 - dC^2 - y_a^2 + y_c^2 - x_a^2 + x_c^2 - 2(x_c - x_a)x}{2(y_c - y_a)} \tag{5.6}$$

$$z = \sqrt{dA^2 - (x - x_a)^2 - (y - y_a)^2} \tag{5.7}$$

Solution to the geometric model

The set of equations 5.5 5.6 5.7, although mathematically solvable, are computationally demanding. Solving these equations on a computing device requires non-negligible time which adds delay for further processing. Since we aimed to make SoundTrak a real-time system, we chose not to solve these equations by the conventional methods to avoid any processing delays. We chose to solve these equations by finding the value of dA, dB, dC, and dD using acoustic data and substituting these values into equations 5, 6, and 7 to solve them.

At any time t,

$$dA_t = dA_0 + \Delta dA \tag{5.8}$$

$$dB_t = dB_0 + \Delta dB \tag{5.9}$$

$$dC_t = dC_0 + \Delta dC \tag{5.10}$$

$$dD_t = dD_0 + \Delta dD \tag{5.11}$$

Where dA_0, dB_0, dC_0, and dD_0 represent the distances of the speaker from the receivers A, B, C, and D, respectively, at time t=0 and ΔdA, ΔdB, ΔdC, and ΔdD reflects the change in dA, dB, dC, and dD over time t. We start by solving these equations by keeping the speaker at a known location at time t=0, which gives us the value of dA_0, dB_0, dC_0, and dD_0, or the initial calibration point. For our setup, we chose the initial calibration point as the center of the watch (shown as the red dot in Figure 5.2. This helps us directly find out

the value dA_0, dB_0, dC_0, dD_0 as $dA_0 = dB_0 = dC_0 = dD_0 = \frac{\sqrt{(x_a - x_b)^2 + (y_a - y_c)^2}}{2}$.

The values dA_0, dB_0, dC_0, dD_0 are known from the initial calibration and ΔdA, ΔdB, ΔdC, ΔdD is calculated from the acoustic data which helps us calculate dA, dB, dC, dD at any time t and in turn find the value of x,y,z.

Calculating distance between the speaker and receivers using phase information

By definition, phase is the position of a point in time on a waveform cycle.Its value falls into the range of 0 to 360 degrees (or 0 to 2π radians). The phase value can be used to calculate the relative displacement between two waves with the same frequency. This relative displacement d between the waves can be calculated using equation 5.12, where $\Delta\varphi$ is the value of the phase shift, and λ is the wavelength of the sound wave. The wavelength of sound is a well-known value that is a result of the speed and frequency of sound as shown in equation 5.13.

$$d = -\frac{\Delta\varphi \times \lambda}{360} \tag{5.12}$$

$$\lambda = \frac{v}{f} \tag{5.13}$$

SoundTrak uses a sinusoidal wave as our choice of signal and using phase information from the signal received from the speaker, we calculate the distance between the finger and the points A(dA), B(dB), C(dC), D(dD). However, converting the phase information to absolute distance will encounter several technical challenges.

Challenges

Firstly, the phase information of the signal received at the receiver cannot alone be used to determine the distance between the speaker and receiver. The distance is usually extracted by using the phase difference between the signal being sent from the speaker and the receiving signal at the receiver at any given time. However, such a comparison

requires the sender to be able to communicate the time-stamp of each wave cycle to the receivers, which can be challenging for wearable devices. How to synchronize the time stamp between devices is the first challenge.

Secondly, the phase only provides information pertaining to one cycle of the periodic wave. This means that if we plan to calculate relative displacement using phase, the maximum displacement will be limited by the range of phase values in a cycle. And from equation 12, the maximum displacement value which can be calculated using the phase is the wavelength of the signal. In the context of finger tracking, this means that if the displacement of the finger is greater than the wavelength of the signal, calculating displacement using phase may become difficult. For instance, the wavelength of 11025 Hz while propagating in air is about 3.2 centimeters. If the displacement of the finger is greater than 3.2 centimeters, the displacement will contain more than one wave cycle as shown in Figure 5.4a. Therefore, the same phase values may appear multiple times. This will cause ambiguity and the system may not be able to determine the phase shift without additional information.

Lastly, during our early experiments, we observed an initial phase bias at each microphone, which is different from one to another and remains constant over time. It is hard to compare the phase value in the presence of such device-specific phase bias.

Solutions SoundTrak uses the following solutions to addresses the above challenges.

Firstly, to solve the problem of removing any communication between the sender and the receiver, we introduce the concept of a reference signal. The reference signal in our system is a replica copy of the signal originating from the speaker and persists at the receiving end. At the receiving end, the reference signal is used while calculating the phase shift/difference between the signal originating from the speaker to the signal being received at the receiver. To obtain the phase value at the speaker, receivers can refer to this reference signal instead of relying on a communication channel between the speaker and the receiver.

In our system, we collect the reference signal ahead of time from the same function

generator which is used to drive the speaker. The reason to use a prerecorded signal instead of generating our own reference signal at the same frequency is that we found a different periodicity of the signal on different machines (probably due to the different quality of the clocks). We recorded about six seconds of the sine wave signal for each frequency from the function generator. We then removed the incomplete period at the start and end of the signal to make the recorded signal periodic. Therefore, we can manually extend the recorded signal for real-time processing. In summary, by using a prerecorded sender signal as reference signal, no communication between the receiver and sender is required when the system is running in real-time.

The other problem is to find the absolute displacement given only the phase of the receiving signal. To calculate the absolute displacement only using phase information, it is required to know the phase value of the signal at the transmitting end as well as at the receiving signal. But as we do not have any communication channel between the sending and receiving end, to increase the practicality of the system, we do not know the value of the phase at the transmitting end. This value of phase at the transmitting end is a random value based on the sent time. To eliminate this initial value, we do not track the phase but the phase shift using the reference signal as presented in the function GET_PHASE_SHIFT in Algorithm 1. The phase shift result we get is the real phase shift plus a random initial value. However, we can get displacement from the last known location to the current location by comparing the phase shift change. For example at data point 500 in Figure 5.3b, the value -300 indicates that compared with the start value 0, the phase shift has changed by -300, which means the distance between speaker and receiver changed $\frac{300\times\lambda}{360}$ based on equation 5.12. Therefore, if we accumulate the distance change between adjacent points from the beginning of the system, and the initial distance at the beginning is known, we can then calculate the absolute location at any given time. To do this, we use the calibration point as the starting position, which is at the center of the four speakers at z = 0. Since we know the coordinates of the calibration position, by accumulating the displacements at each receiver

over time, we can calculate the distance from the current position to the starting point.

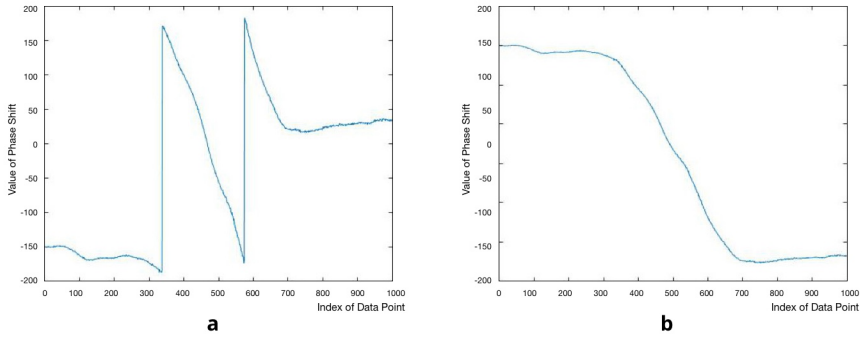


Figure 5.3: Adding extra -360 degrees to overcome periodic limitation. **a.** The phase shift result. **b.** The phase shift result after correction. The x-axis represents the index of each data point, and the y-axis represents the value of phase shift at each data point.

Secondly, to solve the problem of the maximum displacement value limited to just one wavelength, we continuously track the phase shift to detect a change of cycle in the periodic wave. For this, whenever we detect a sudden change of about $+/-360$ degrees in the phase shift value, we accommodate for this change by adding correction values to the original phase shift value. There can be two reasons for this sudden change of the phase shift value. The first reason is that when subtracting the two signals, one of the two signals exceeds its period and the other one remains in the current period, as shown in Figure 5.4a. The second possibility is that the phase shift value exceeds a period because the distance between transmitter and receiver changed more than a wavelength, as shown in Figure 5.3a.

Figure 5.4 shows the result of directly subtracting the reference signal phase (Figure 5.4b) from the received signal phase (Figure 5.4a), which would introduce a peak when any of the signal exceeds the edge of one period as shown. We can easily solve this issue by extending the phase value which exceeds the period slightly over the range -180 to 180 degrees until both signals move to the next period. Thus we get a reasonable phase shift result as shown in Figure 5.4d. The resulting phase shift is a constant, which means location did not change in between the two given times. A closer observation shows that there are only 20 points in Figure 5.4, which equals $20/44100 = 0.0004$s and hence the

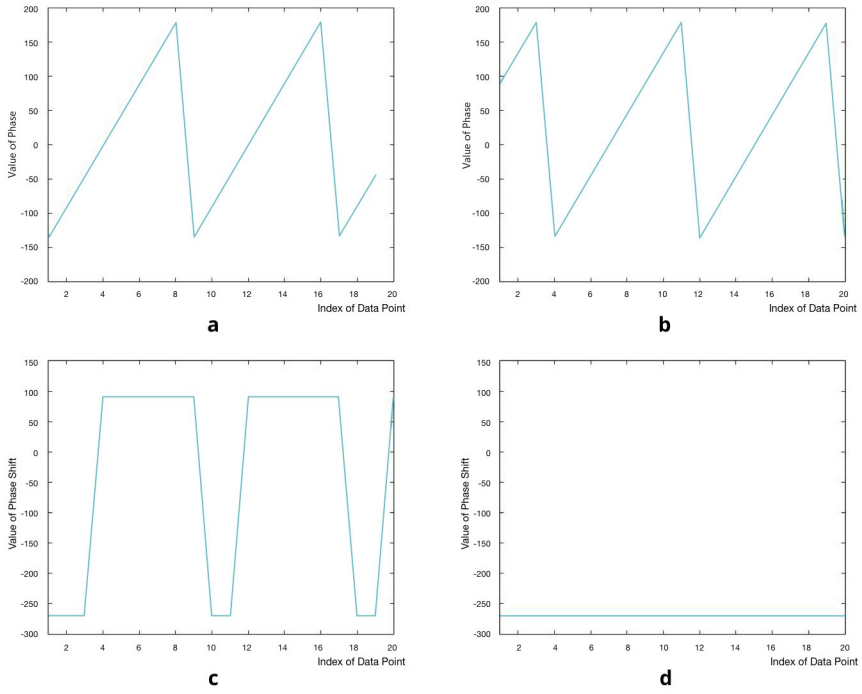


Figure 5.4: Processing phase at different stages. **a.** The phase value of a received signal. **b.** The phase value of a reference signal. **c.** Result of subtracting the reference signal phase from the received signal phase **d.** The phase value after extending it beyond a cycle. The x-axis represents the index of each data point, and y-axis represents the value of phase at each data point

distance could not change significantly in such a short time. This technique addresses the first reason of a sudden change in phase shift due to one signal exceeding its period while another does not.

The second reason for a sudden phase shift is when the distance between transmitter and receiver changes by more than a wavelength. Figure 5.3a shows the phase shift result of a continuously "moving away" gesture, for which the phase shift value should keep decreasing based on equation 5.12, but it suddenly increases 360 degrees when it exceeds a period. We solve this problem by subtracting 360 degrees for the phase shift value after the sudden change, thus making it smooth. Figure 5.3b shows the phase shift result after correction for the periodic problem. This method is feasible with one assumption, that the phase shift value between two adjacent data points would not be close to 360 degrees, i.e.,

the finger will not move about a wavelength far in a unit sample time. For example, if we use 11025Hz (the wavelength will be 0.032 meters) as the signal frequency and 44100 Hz as the sample rate, it means that the finger will not move at a speed that is faster than 1411.2 m/s (0.032 meters/(1/44100)seconds), which is a reasonable assumption. Algorithm 1 below describes this phase shift adjustment.

ALGORITHM 1: Algorithm to overcome periodic limitation

if $phase\_shift - previous\_phase\_shift \approx 360$ **then**
    phase_shift=phase_shift-360 ;
    period_add=period_add-360 ;
if $phase\_shift - previous\_phase\_shift \approx -360$ **then**
    phase_shift=phase_shift+360
    period_add=period_add+360 ;

Figure 5.5: Algorithm 1

Finally, to address the different initial phase offset , we record the offset at each receiver during the initial calibration process and accommodate for these offsets in all future calculations.

5.1.3 System design

To validate our model and implementation of SoundTrak, we built a hardware prototype and data processing pipeline. For our initial prototype, the user needs to wear the speaker in a casing resembling a finger ring.

Hardware Design

Figure 5.7 shows the hardware setup for SoundTrak. The hardware of the prototype contains three major parts: the sender, the receivers, and the data acquisition hardware. We use a miniature speaker (Knowless FK-23451, 5.00mm Length *2.73mm Width * 1.98mm Height) as the sender, which is attached to a 3D printed ring (Figure 5.7B). The speaker is driven by a function generator (Agilent 33500B, Figure 5.7A). The system has four MEMS microphones (STMicroelectronics MP33AB01) as receivers (Figure 5.7C). Each microphone is connected to a separate customized preamplifier (Figure 5.7D). The output from

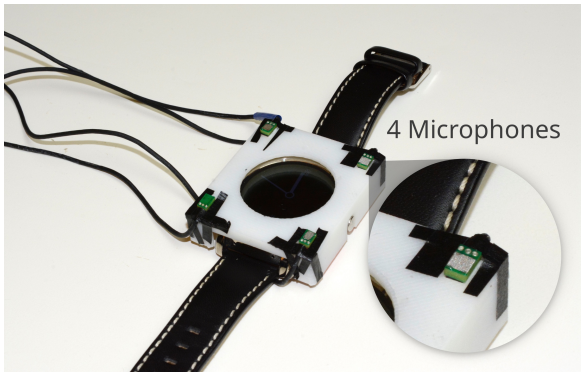


Figure 5.6: LG G Watch with SoundTrak

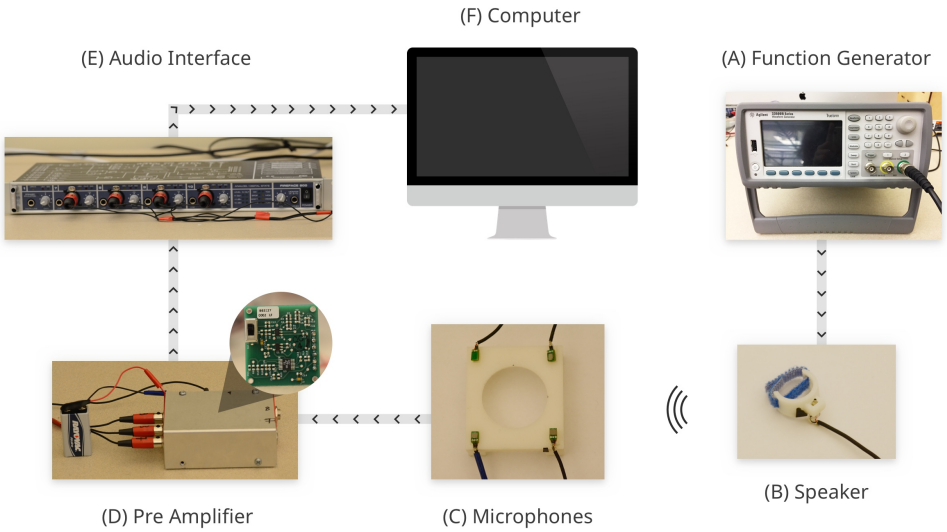


Figure 5.7: Hardware setup for SoundTrak

these preamplifiers then goes to an audio interface (Fireface 800, Figure 5.7E) for data acquisition. We wrote a thin client in C with the PortAudio library to collect data on an iMac (Figure 5.7F). The data is transmitted via a socket to a Java program for real-time processing on the same iMac.

Algorithm Pesuedo Code

The real-time processing algorithm (see Algorithm 2 pseudocode) contains three major functions. UPDATELOCATION is the general data pipeline of calculating locations from

the input sound data. CALIBRATION records the $correction\_value$ at each receiver, which includes information of the sensor bias and the initial distance between speaker and receiver at calibration. The $period\_add$ value is used to accommodate the period change of the $phase\_shift$ value. Using the equations 5.1, 5.2, 5.3, and 5.4, the function LOCATION_FROM_FOUR_RECEIVERS calculates the current coordinates of the ring/speaker. There are three combinations of equations from these four equations used to obtain the four location values, as described in Section 3.1. Finally, the speaker location is assigned as an average of the four location values.

Choice of frequency

In theory, SoundTrak does not have the limit of which frequency can be used in the system. However, we chose the signal frequency based on the following practical considerations.

First, we use a Fast Fourier Transform (FFT) to calculate the phase value of every $N = 512$ points of the received signal. The results of FFT are a list of bins, each of which represents the information of frequency $k \times SampleRate/N$, where k is an integer from 1 to $N/2$ and the sample rate is 44100 Hz. Therefore, we need to use the frequency related to one of the bins. Second, we observed that the phase value is more sensitive to noise when the frequency is under 3k Hz, as more environmental noise exists in this frequency range. Third and most important, the higher the frequency is, the shorter the wave length will be. So for higher frequencies, the same amount of phase shift represents a shorter distance which can have potentially higher resolution. Therefore, we tested our system with a frequency of 11025Hz for system evaluation and 16537.5Hz (inaudible) for the Fitts' Law test for the ease of FFT calculation.

Sensor placement

Four microphones are placed at the corners of a rectangle with a length of 4 cm and a width of 4.3 cm. The sensor placement is mostly determined by the practical challenge of

ALGORITHM 2: Iterative Algorithm

```
function UPDATELOCATION(sound data)
Begin receiving sound data
period_add = 0
call function CALIBRATION()
previous_phase_shift = phase_shift
while gettingdata, do
    for every receiver, do
        bandpass filter
        phase_shift ← GET_PHASE_SHIFT()
        phase_shift = phase_shift + correction_value + period_add
        if phase_shift − previous_phase_shift ≈ 360 then
            phase_shift=phase_shift-360 ;
            period_add=period_add-360 ;
        if phase_shift − previous_phase_shift ≈ −360 then
            phase_shift=phase_shift+360
            period_add=period_add+360 ;
        pre_phase_shift = phase_shift
    end
    return Location ← LOCATION_FROM_FOUR_RECEIVERS()
end
end function
function CALIBRATION
    for every receiver, do
        bandpass filter
        phase_shift ← GET_PHASE_SHIFT()
        calculate correction_value = −phase_shift − 360 ∗ initial_distance/wavelength
        phase_shift = phase_shift + correction_value
        pre_phase_shift = phase_shift
end
    return location ← LOCATION_FROM_FOUR_RECEIVERS()
end function
function GET_PHASE_SHIFT
    for each sample point in received data stream, do
        phase ← FFT
        phase_shift = phase − reference_phase
end
    return phase_shift
end function
function LOCATION_FROM_FOUR_RECEIVERS
    location1 ← calculate location by solving equations (1),(2),(3)
    location2 ← calculate location by solving equations (2),(3),(4)
    location3 ← calculate location by solving equations (3),(4),(1)
    location4 ← calculate location by solving equations (4),(1),(2)
    final_location ← average of location1 , location2, location3 and location4
    return final_location
end function
```

Figure 5.8: Algorithm 2

deploying the sensors on top of a commercial smartwatch. Using a more advanced sensor placement would potentially increase the range of operation and the accuracy of the system. We plan to further investigate this in the future.

5.1.4 Evaluation

Assessing spatial resolution

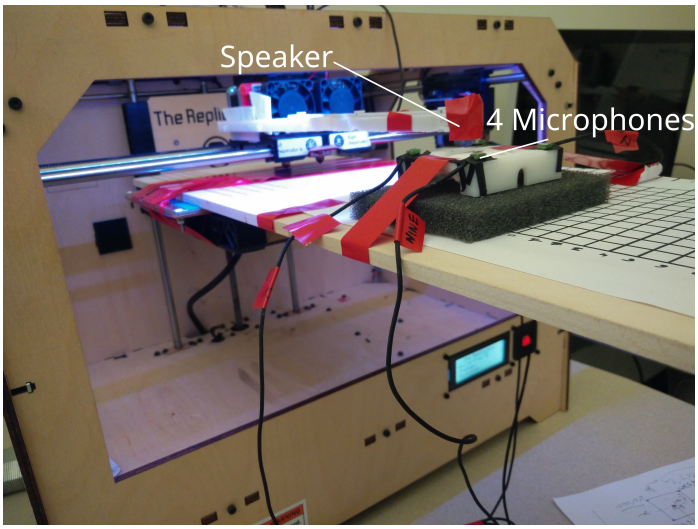


Figure 5.9: System Evaluation Setup with MakerBot

Experiment:

To measure the tracking accuracy of SoundTrak in 3D space we designed an experiment which involved comparison of values recorded by the system against a predefined set of ground truth coordinates. We used an off-the-shelf Makerbot (Replicator) to position the speaker accurately in 3D space as shown in Figure 5.9. The ground truth coordinates were selected from a 3520 cm^3 3D volume space of ($16cm \times 20cm \times 11cm$) with all the coordinate combinations from the set X (cm)= $\{-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10\}$, Y (cm)= $\{-8, -6, -4, -2, 0, 2, 4, 6, 8\}$ and Z (cm) = $\{1, 3, 5, 7, 9, 11\}$. The z-axis refers to the vertical direction (up and down) and the x-axis refers to the horizontal direction (left and right), when the user is facing the Makerbot. The experiment was conducted in a shared large room including uncontrolled common office noise (e.g., people conversation, air conditioner, phone ringing). The Makerbot used for this experiment is moved to a specific coordinate in 3D by injecting corresponding GCode (low-level instructional codes to control RepRap based machines) commands over its serial port. The SoundTrak speaker

was connected to the Makerbot's left extruder. The receivers were placed on an extended wooden plank on the Makerbot's platform. The extension provides flexibility to choose a calibration point and coordinate axes of our choice and moreover it also minimized the effect of noise produced due to the movement of the Makerbot's motor. The data collection pipeline was semi-automatic and two co-authors synchronized the Makerbot's movement and the corresponding location data logging to maximize accuracy and minimize time synchronization issues. Location data for 594 location coordinates was collected in under 2 hours using this approach. The results and accuracy of the system are discussed in the following section.

Results:

Figure 5.10 depicts the ground truth coordinates for the system evaluation and Figure 5.11 shows the corresponding calculated values by the SoundTrak system. For a total of 594 location coordinates in the entire volume of 3520000 mm^3, the mean Euclidean error is 13.05 mm. By providing centimeter-level resolution consistently over a large volume space, the system can support a large variety of interactions for the human finger. These results further show that the system is very well suited as a three-dimensional input system for wearable technology, such as a smartwatch. From our evaluation results, the accuracy of the system is maximum for z = 9 cm (mean error = 0.92 cm) in the Z plane, for x = 0 mm (mean error = 0.87cm) in the X plane and for y = -2cm (mean error = 0.93 cm) in the Y plane.

Figure 5.12 contains six heat maps for Z ranging 1cm to 11 cm. The coordinate system and the placement of the watch used for the system evaluation are represented in the top-most image. From the heatmaps, it is evident that SoundTrak achieves maximum accuracy for the region within Y= -2cm to 6cm and X = - 4cm to 6cm for all the Z range. For Z = 1cm, the location calculation accuracy deteriorates as we move the speaker in the negative X and Y direction. Although the mean error is minimum for Z = 9cm, it is noteworthy that the variance is lesser for the Z = 7cm. Overall, SoundTrak is more accurate for $Z > 5$cm

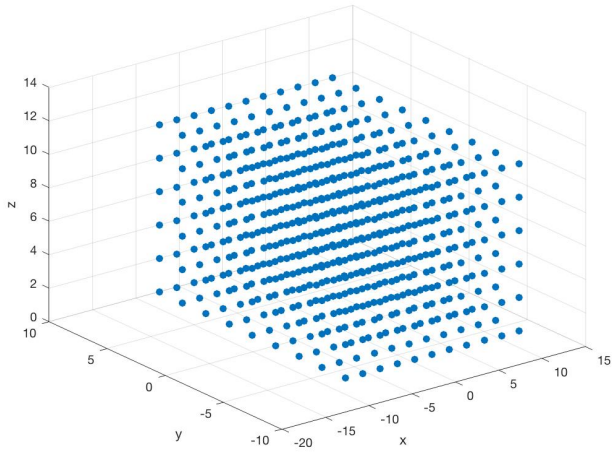


Figure 5.10: The predefined positions in the system evaluation

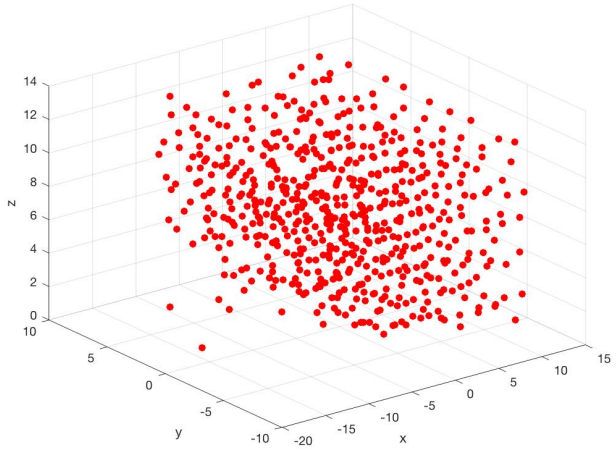


Figure 5.11: The positions calculated by SoundTrak in the system evaluation

compared to $Z < 5$cm. The optimal region for 3D input is defined by the volume Z=7cm to 9cm, X = -2cm to 6cm and Y=2cm to 4cm. Point A in the coordinate system is the reference point and all the values are calculated considering point A as the origin of the system for the testing.

From Figure 5.10 we observe that for the ground truth values of Z = 1cm the predicted values are less accurate. We attribute the higher error rate when z is smaller to two possible causes. First, the speaker is not omnidirectional, it outputs unbalanced energy in different

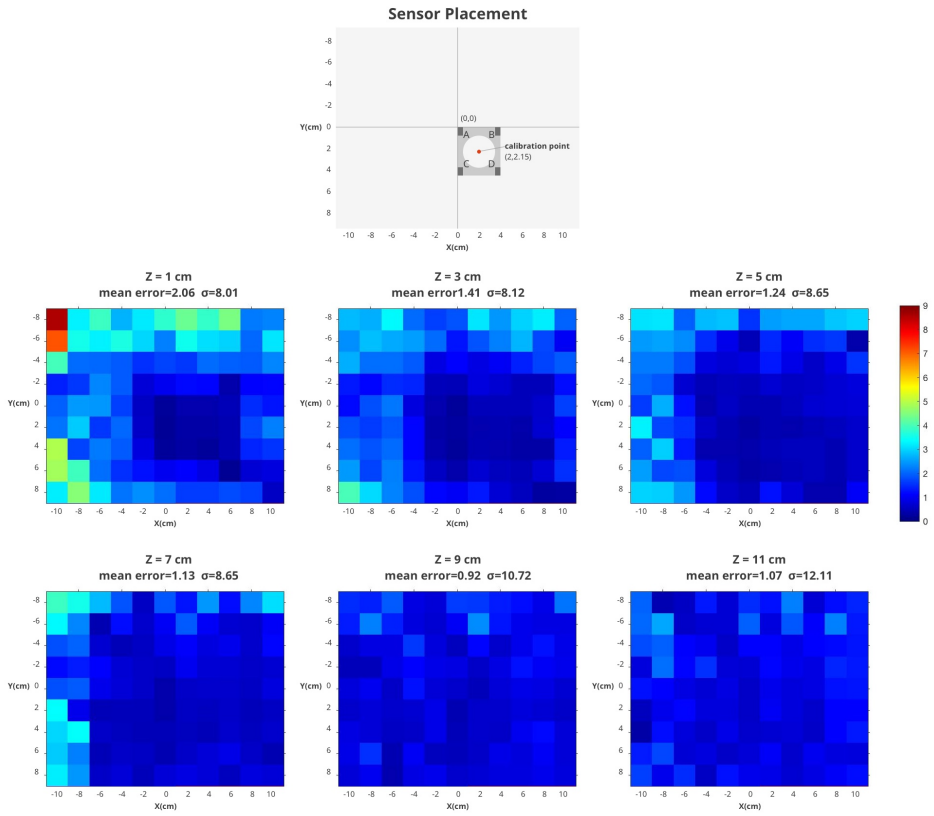


Figure 5.12: Heatmap of euclidean errors calculated along x-y planes at different z (height) values

directions. The direction where the speaker is pointing to would receive more energy than the direction where it is perpendicular to the speaker. Second, the reflected signal on the surface can be relatively stronger when the distance between the speaker and the surface is closer (e.g. z = 1cm), resulting in higher errors.

Evaluation of the drift

Experiment: SoundTrak integrates the phase changes over time to calculate the location of the finger. However, noise produced by hardware and the environment cannot be completely avoided in such systems. This noise in the signal can result in errors while calculating the phase values. When integrated, even a minimal error at each data point may cause a non-negligible influence on the accuracy over time. Furthermore, if the accuracy drops significantly, the participant may have to recalibrate the system again, which would

degrade the user experience.

To evaluate the drift of our system over time, we conducted another experiment using the same frequency (11025 Hz) and hardware setup as the system evaluation experiment described above. We recorded the data for 20 minutes at each of the three locations, where the position on x and y-axis are 0 cm, and the position on z-axis is 1,5,9 cm for each session respectively. 20 minutes is a relatively large time interval for a user interacting with a smartwach continuously and offers a good estimate of how the system would perform over even larger intervals of time. To avoid human error, we generated a file containing location information along with the time-stamp for each location point.

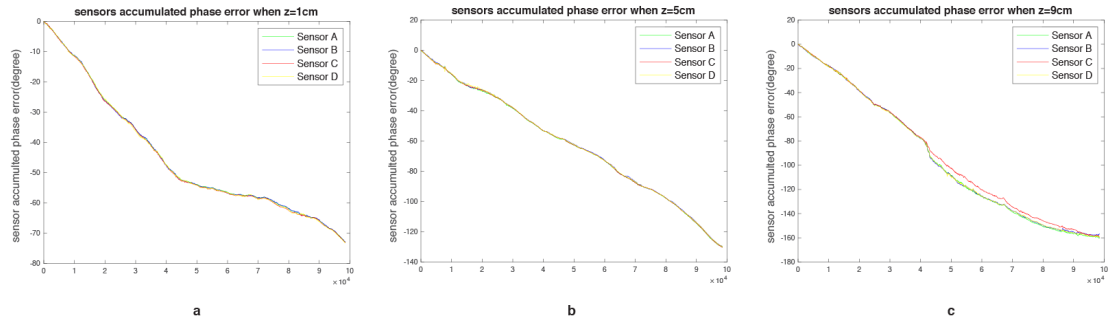


Figure 5.13: Accumulated phase changes on three axes for 20 mins when z = 1,5,9 cm

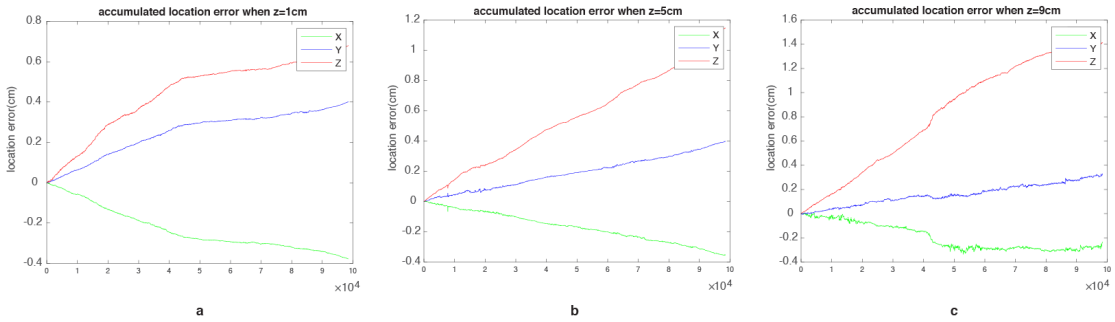


Figure 5.14: Accumulated location changes on three axes for 20 mins when z = 1,5,9 cm

Results: To calculate the drift of phase values on each microphone, we subtract the starting phase value from all the later phase values to derive the relative phase changes over 20 minutes. The results are shown in Figure 5.13. The phase drift on different sensors are

similar at each location. The phase decreased 73, 130, 159 degrees respectively for z = 1, 5, 9 cm in 20 minutes.

Based on these phase value, we further calculated the location drift on x,y,z-axis by using the formula presented in Section 3. Similarly to the method we used to derive the phase drift, we subtracted the original coordinates to all the later ones, and the results are presented in Figure 5.14. In the three 20-minute experiment, the x coordinates decreased 0.375 cm, 0.35 cm and 0.33 cm, the y coordinates increased 0.398cm, 0.397 cm, and 0.314 cm, and the z coordinates increased 0.68 cm, 1.148 cm and 1.412 cm respectively.

The above results show a greater drift for the z-axis than the x and y axes. The reason is the coordinate on z-axis is calculated based on the coordinates of x and y axes, as formula 5.7 described. Any drift of the location along either the x or y axis would be exaggerated when calculating the position on the z-axis. Therefore, while designing interactions in 3D space using SoundTrak, the x-y plane sould be considered as having a higher spatial location accuracy.

There are two possible reasons that may cause the drift. One is the unreliable clock of the functional generator that may introduce accumulated drift over time. Since we did the z=1cm session first and z= 9cm session last, it is possible that the longer time the system is running , the larger drift may be observed. Another possible reason is the system suffers more from multipath effect when the distance between the speaker to the receivers are larger, as we will discuss in details in section 5.1.5.

Despite the drift error, considering most of the interaction on wearable devices would not exceed 20 minutes per session, even such a drift exists, the user may not need to recalibrate the system during an interaction session. However, we plan to further investigate this system drift while the speaker is in motion in the future.

In addition, as Figure 5.13 shows, the phase value does not decrease at a constant rate. Sometimes the phase would suffer a relative and sudden large drop such as the center of the Figure 5.13c shows. We attribute this to the random environmental noise presented

during the whole experiment. Various types of environmental noise appeared during the experimental session, such as the air conditioner noise, loud music played by students, phone rings, conversations among students, as well as the sound caused by opening and closing of the doors. These are all natural sounds that would be expected in normal use of SoundTrak.

User Evaluation

To better understand the usability of SoundTrak from the user's perspective and how it could be used in practice, we conducted a Fitts' Law study. Though SoundTrak is able to track a finger's position in 3D space, we conducted the user study in two 2D planes (vertical and horizontal). The reason is that it is more likely that SoundTrak will be used for devices that still have a 2D output device in the near future. Understanding how SoundTrak would potentially enhance the interaction with the 2D screen is important. Therefore, we decided to conduct a Fitts' Law study, which is widely used to evaluate rapid and targeted movement on 2D screens. In order to map the movement in a 3D space to a 2D screen, we split the 3D coordinate system into the X-Y plane and the X-Z plane, which are horizontal and vertical to the watch face, respectively in the study.

Participants and Apparatus:

In this user study, we recruited 10 participants with an average age of 26.2 (6 males) from a university campus. All of them were first-time users. During the study, participants were sitting in front of the computer running the Fitts' Law software and resting their left arm on the desk. They were wearing an LG Watch Urbane with a 3D printed case on their left wrist. As shown in Figure 5.6, four microphones were placed at the corners of the case on the watch. A miniature speaker was taped to the nail of the index finger on their right hand.

To complete a Fitts' Law task, participants were asked to control a cursor on a nearby computer screen by moving and pointing the speaker on their right index finger around

sensors on the watch on their left wrist. A trackpad was placed on the desk next to the participant's left hand. They were instructed to tap the trackpad to confirm a selection for a Fitts' Law task. The computer was a 27-inch iMac with a resolution of 2560 x 1440 pixels. We mapped every 100 pixels on the screen to 1-centimeter movement. To stabilize the cursor on the screen, we drew the current cursor at the position which was the average of the last 30 location values obtained by SoundTrak.

Procedure and Design:

At the beginning of the study, a researcher helped the participant to put on the watch and the speaker. The participants were then given an introduction of how to interact with the test software, including mapping the finger movement in the relevant 2D plane to the cursor movement on the screen and tapping the trackpad to confirm the selection using their left hand.

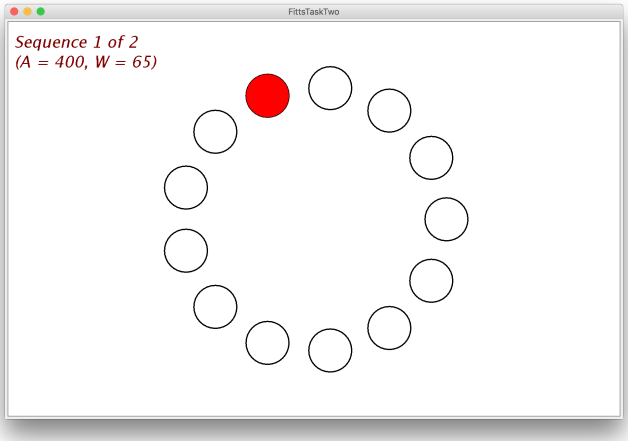


Figure 5.15: Fitts' Law test interface with 13 targets.

The Fitts' law software we used in the study was developed by MacKenzie et al.[1]. Tasks were designed based on the ISO 9241 standard, as modeled by Fitts' Law. We had 13 targets in the test as shown in Figure 5.15. Participants were instructed to move the cursor to select the highlighted target using SoundTrak as accurately and quickly as possible. The system highlighted the next target after selection. A trial ended after all targets were selected. We

[1]http://www.yorku.ca/mack/FittsLawSoftware/

set the parameters to be two conditions (X-Y plane, X-Z plane), two distances (400 and 500 pixels), and three target sizes(40, 65 and 90 pixels). For each condition, there were four sessions. Each session consists of 6 trials (6 = 2 distances x 3 target sizes). In each trail, the participant selected 13 targets. The system was calibrated at the beginning of each trial. In total, each participant completed 48 trials (2*4 sessions, with 6 trials in each session). The order of conditions and the order of trials were randomized. The movement time and the error rate were recorded.

Results:

Learning Effect To understand the learning effect, we conducted repeated-measures ANOVA on the results. Fig 5.16 shows the learning effect for each plane on movement time. The analysis shows the effect was very significant for the X-Y plane ($F_{3,9} = 15.42$, $p < 0.001$), but not significant for the X-Z plane ($F_{3,9} = 3.59$, $p > 0.05$). However, the difference from session 2 to session 4 is not significant for both the X-Y plane ($F_{2,9} = 4.82$, $p > 0.1$) and the X-Z plane ($F_{3,9} = 2.67$, $p > 0.1$) . Therefore, to avoid potential issues caused by the learning effect, session 1 is considered as a training session and all analyses of the results are based on session 2–4 only.

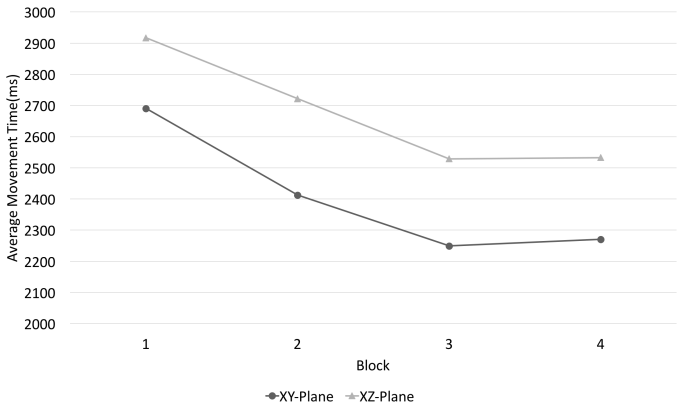


Figure 5.16: Average movement time by plane and session

Movement Time The average movement time per trial for the X-Y plane is 2310.87 ms with a standard deviation(SD) of 249.81, and for the X-Z plane is 2594.52 ms (SD = 362.72). The difference is considered to be very statistically significant ($p = 0.0038$). The analysis of the movement time by the effective index of difficulty (IDe) also indicates that it takes longer time to accomplish more difficult tasks.

Error Rate and Throughput The average error rate per trial for the X-Y plane is 6.03% (SD = 3.38), and for the X-Z plane is 7.91% (SD = 3.93). However, the difference is not statistically significant ($p = 0.0694$).

The average throughput (bps) for the X-Y plane is 1.41 (SD = 0.12), and for the X-Z plane is 1.31 (SD = 0.16). The analysis shows a significant difference between these two planes ($p = 0.0199$). One possible reason is that most of the 2D tracking devices are designed to be operated in the X-Y plane. It is more challenging for most participants to perform movement in the X-Z plane.

Implications The results of the study demonstrate the learnability of our system. Participants were able to get used to the proposed interaction on the watch after only one training session. However, we also noticed the significant difference of the performance between the X-Y plane and the X-Z plane in the study. We attribute this difference to two factors. First, according to the results presented in Sections 5.1.4 and 5.1.4, the calculated location on z-axis is less accurate than the calculated locations on the x and y axes. The lower accuracy could potentially influence the interaction experience and the performance of the participants. The other possible reason is that many participants reported they were more used to performing gestures on the horizontal plane (X-Y plane) because this is how the interaction is designed on most of their computing devices (e.g., phones, watches, IPads). Therefore, the interaction designer may want to first consider X-Y plane while designing finer input gestures to achieve a better user experience. This does not rule out use of the X-Z plane, but does caution about needed precision of the input gesture.

5.1.5 Discussion

Applications

Gesture control for wearables: Interactions with wearable devices are currently limited by the size and the placement of devices. For instance, using fingers to zoom in/out on a map on the smartwatch is constrained by the small screen size. By allowing the finger movement in 3D space, SoundTrak enables an extended and flexible gesture control for wearable devices. Not limited by the physical size of devices, users are able to perform a much larger set of gestures. It is also possible to establish a new set of 3D gestures for smartwatches built upon SoundTrak that can improve the usability of smartwatches in general. For instance, it is not convenient to launch an app on the smartwatch at this moment, since a user needs to go through a long list on the screen for selection. However, if the list is too long, repeating the sliding gesture is needed. With SoundTrak, the user can easily select the application by moving the finger in one direction in the relative large 3D space around without the need to repeat the same gesture again.

Our current SoundTrak prototype tracked only one finger, but there is not fundamental reason why this technique could not be extended to multiple tracked fingers. By placing SoundTrak rings on multiple fingers, users can easily perform multifinger gestures, such as the standard pinch gesture in the peripheral space around the watch to zoom in/out. A much wider range of 3D gestures can now be explored.

Text input: SoundTrak can also be used for text input for devices that do not have a physical keyboard. By tracing the finger movement in real-time, SoundTrak allows users to draw anything in the air and detects the input. For instance, users can reply short messages quickly from their smartwatch using SoundTrak.

Drawing in 3D space:

Most of the current drawing applications are based on 2D tracking technology. Therefore, it is only able to draw the picture in a 2D plane. Though a few smartwatches provides

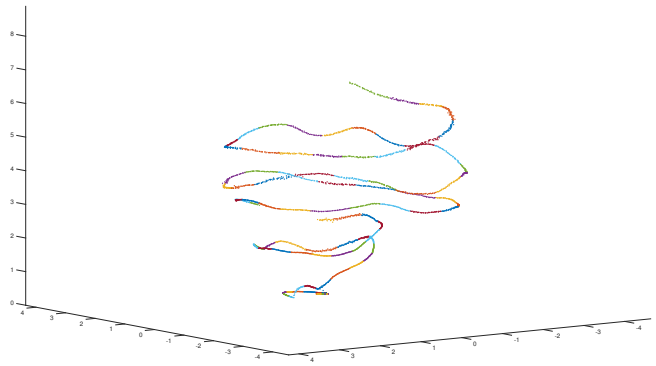


Figure 5.17: Drawing Spiral in 3D with SoundTrak

drawing application, it is particularly challenging to draw on a picture on the tiny screen of the watch. SoundTrak enables the user to draw the picture in 3D space around the wearable devices. Therefore, the user can draw picture in 3D space at any time and at any location with their wearable. Figure 5.17 shows the sample 3D pictures drawn by using SoundTrak.

We should note that we have currently been promoting SoundTrak as a way to provide 3D interaction with a wearable device, such as a smartwatch or a head-mounted device like Google Glass. There is no reason why this 3D interaction should be limited to those devices, and in the case of 3D drawing, we can easily see a motivation for providing this kind of 3D tracking above or around any screen (e.g., table, smartphone, laptop).

Improve interaction on wearable devices with physical spatial awareness: Most of the existing tracking technologies on wearable can only track the relative movement of the finger. The wearable is not aware of the finger's position in the 3D space. Providing the physical spatial awareness to the wearable can be potentially used to redesign the interaction for improving the interaction efficiency and user experience. For instance, we can define any space around the wearable as the location of a virtual button. By moving the finger to that predefined space, a user can click a button. However, clicking the virtual button accurately in 3D space can be challenging. We plan to further explore this in the future.

3D input for other scenarios: As already suggested, the technology presented in SoundTrak is not limited to detecting a single finger's position on wearable devices. It can also be used in other scenarios, such as turning the area around a horizontal or vertical surface into a 3D interaction area. If multiple speakers can be attached to different fingers, it is also possible to capture multiple fingers' position simultaneously. The availability of such "multi point" sensing in 3D space can open up a whole new space for novel gestures interactions with wearable and other devices.

The Effect of Multipath and Environmental Echoes

Like other acoustic based systems, the transmitted acoustic signal does not exclusively travel in the direct path from speaker to microphone; it can be scattered by the fingers or other environmental objects along the way. Therefore, multipath problems may exist. The phase of received multipath signals are different than the straight line path, which may contribute to the error in our system. However, this is not a significant issue in our system.

The multipath signal only contributes to a relatively small fraction of the received signal. With a frequency of 11 kHz, and a 2 cm diameter finger, the dimensionless wavenumber with respect to radius is $ka = 2$. With that ka, the scattered pressure in all directions is roughly 3 dB down with respect to the incident wave amplitude [93], which means that the scattering loss is about one half. What's more, the Inverse Square Law [94] states that the signal intensity is inversely proportional to the square of the distance from the signal source, which can be applied on the sound traveling process in air, both for direct signal and the scattered signal. Therefore, the scattered field incident back to the microphones is always comprised of contributions that have suffered greater spherical spreading loss (longer path length) than the direct field, and also scattering loss. Therefore, their contribution at the microphone array is a small fraction of the total sensed field.

For environmental echoes, presumably from structures even further away than the fingerwatch system, the strength of the multipath field over those longer propagation paths is even

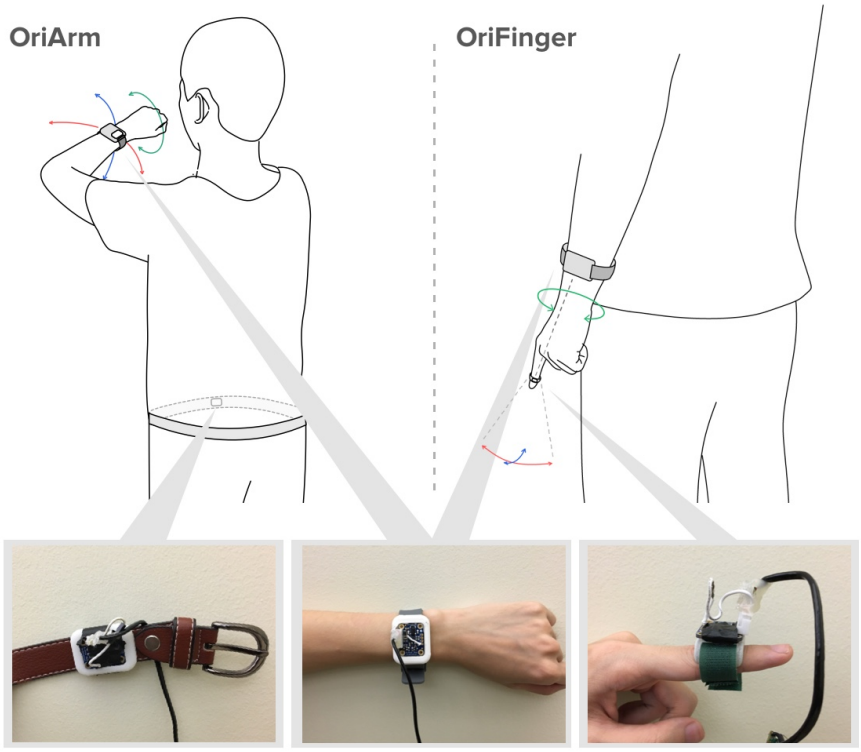


Figure 5.18: Hardware setting of OriTrak

lower.

Combining these factors together, we think the multipath signals and environmental echoes only contribute to a very small portion of the received signal and would not cause significant errors in most of the cases.

5.2 OriTrak: Relative Orientation Tracking For Interaction with Wearables

5.2.1 Overview

In this section, we provide a more general-purpose continuous input technology, that provides valuable information about the fingers or hands. This continuous input can then be used by designers to develop different input techniques for a wide variety of situations, such as painting, gaming, signing a signature, map manipulation, and gesture recognition. It is called OriTrak, which can continuously track the three-dimensional angular position

(orientation)[2] of a wearable device with respect to some part of the body. Two IMUs are mounted on the wearable device(e.g., wristband) and another place on the body (e.g., wrist, torso) respectively. We calculate the angular position by comparing the absolute orientation between two sensors. Absolute orientation is the angular position of an object with respect to earth's canonical coordinate system, whose x, y, z-axes are pointing North, East, and perpendicular to the surface, respectively. The absolute orientation of a wearable device such as a watch varies based on the direction of the body and the wrist. That means that even if a user maintains a fixed pose, the absolute orientation of the smartwatch can still be different when the body is facing different directions. This variability significantly limits the applications for wearable input. For instance, if the absolute orientation of a smartwatch is used for gesture input, the user has to keep the body still during the whole interaction, which is unpleasant experience. If the values of absolute orientation are used to training a gesture recognizer, the recognition system will not work well when the user is facing at a different direction. To address this issue, OriTrak adopts another orientation sensor which is usually attached to the body in addition to the original one on the wearable. By comparing the absolute orientation between these two sensors, we can derive the angular position of the wearable with respect to the body. Absolute orientation is calculated with a 9-axis Inertial Measurement Unit (IMU). Comparing with motion sensors (e.g., gyroscope, accelerometer), which can only sense the relative movement, continuously tracking the angular position in 3D space can potentially support interaction with more expressiveness. Furthermore, since IMUs are widely available on most commodity devices, OriTrak can be quickly adopted into future wearable interaction systems.

We demonstrate two wearable interaction techniques that is developed using OriTrak. The first technique is called OriFinger, which uses a finger-mounted ring to allow the user to move the finger relative to the wrist and control the position of a cursor on some other device. With OriFinger, the user can provide continuous two-dimensional input to a head-

[2]https://en.wikipedia.org/wiki/Angular_displacement

mounted display (e.g., Google Glass). The second technique is called OriArm, which supports one-handed gesture input to a wrist-mounted device (e.g., smartwatch) through movement of the arm relative to the torso. A user study was conducted to evaluate the input efficiency of both interactions.

5.2.2 Theory of Operation

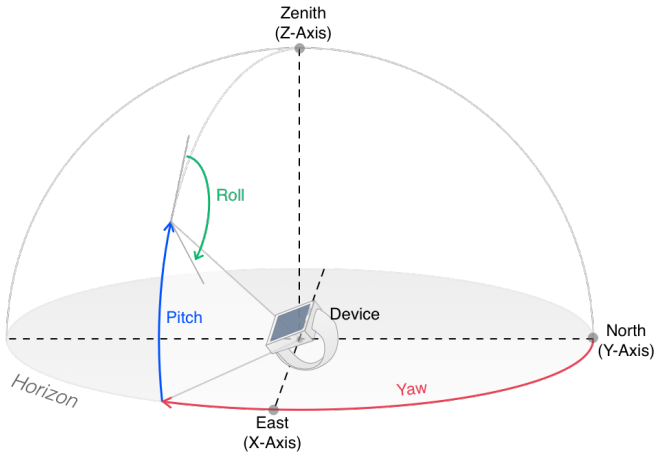


Figure 5.19: Using Euler Angles to Represent Absolute Orientation

Absolute Orientation

OriTrak is developed based on the absolute orientation of the wearable device, which are rotations between the coordinate system of the device with respect to the coordinate system of earth. The x, y, z-axes point to absolute East, North, and the opposite of the gravitation vector (up), respectively in the coordinate system of the earth (see Figure 5.19). We describe the rotations on x, y, z-axes using roll, pitch, and yaw, respectively, which are terms defined in Euler angles [3]. Different sequences of rotations would result in different Euler Angels. In OriTrak, we always use the sequence of 'z-x-y', which is equivalent to 'yaw, pitch, roll'.

[3] https://en.wikipedia.org/wiki/Euler_angles

Angular position with respect to the body

Absolute orientation of the wearable is highly dependent on the direction of the body, even if the body posture is exactly the same. To remove the dependency of the absolute orientation on the body direction, we derive the orientation between the wearable device with respect to some parts of the body using a pair of absolute orientation sensors. OriTrak presents two settings. In the first setting named OriArm, the two sensors are attached on the wrist and torso respectively to calculate the angular position of the wrist with respect to the body. In the other setting named OriFinger, the two sensors are attached on the finger and the wrist respectively to derive the angular position of the finger with respect to the wrist. The orientations between wearables with respect to a certain location on the body is derived by calculating the rotations between Euler angles vectors from two sensors. The result is also represented by Euler angles (roll, pitch, yaw). For instance, in order to acquire the angular position of the wrist with respect to torso in the first setting, the system calculates the rotations between Euler angles provided by the two sensors on the wrist and torso.

5.2.3 Gesture Design

In this paper, we demonstrate two interactions schemes using OriTrak: *i) OriArm*, which provides one-handed gestures for watch interaction by continuously tracking the angular position of the wrist with respect to torso; and *ii) OriFinger*, which provides continuous two-dimensional finger input by tracking the angular position of the finger with respect to the wrist. In this section, we present the detailed design as well as the design rationale of both interactions.

OriArm

OriArm enables the user to interact with smartwatches using one-handed arm and wrist movements, by continuously tracking the relative angular position of the wrist with respect to the body. It is designed to provide efficient one-handed inputs for smartwatches in

scenarios when the hand is occupied. For instance, if a user is carrying a shopping bag or holding a baby in the right hand and wearing the watch on the left hand, in order to interact using the touchscreen of the watch (e.g., reject/ answer phone calls), the user might have to put down the bag or the baby, to free the right hand, which is inefficient and may be undesirable. For such, and other, scenarios OriArm enables simple one-handed gestures to be used for interacting with the watch by moving and rotating the watch wearing arm.

The gesture design of OriArm is to provide alternative one-handed input gestures for a smartwatch. We analyze the input operations on a commodity smartwatch, which can be divided into three steps: First, a user lifts up the arm and unlocks the watch screen by tapping on the screen or pressing a button. Second, sliding the finger on the touchscreen to navigate on a one-dimensional list. Third, once an item of interest appears at the center of the screen, the user taps on the item to make a selection.

We designed three corresponding one-handed gestures for watch interaction using OriArm. First, the user unlocks the watch by holding the wrist at a position where contents of the screen can be seen comfortably, as Figure 5.18 shows. We allow the user to define this position in advance. In most scenarios, holding the arm and watch towards the head (the traditional ”look at wrist watch movement”) expresses the intention of input on a watch by a user. Once the input intentions is detected, the user needs to rotate the wrist as a gesture to confirm the input intention, which also unlocks the screen for further interaction. Second, once the screen is unlocked, a user can navigate through a one-dimensional menu list by moving the arm forward and backward as figure 5.20 shows. In this scenario, the proposed one-handed gesture replaces the conventional on-screen sliding interactions. Third, after scrolling a menu list to localize the menu item of interest, the user can confirm the selection by performing a wrist rotation gesture – an alternative to a tap on screen as it is done in two-hand interaction scenarios. These three gestures together form a complete one-hand watch interaction system.

OriArm is designed to explicitly take social appropriateness into consideration for in-

teractions with wearables. Extending the arm from the chest to the farthest position may look awkward and inappropriate in a social setting. Also, the user can barely see the content of the watch if the arm is too far from the body. Instead of requesting the user to move the arm from the chest to the farthest position to scroll a list, OriArm defines an interaction area of 26 degrees in yaw, which was empirically discovered from pilot studies, as Figure 5.20 shows.

OriFinger

OriFinger allows the user to control a cursor on screen by mapping the angular displacement of the finger to the coordinates in x- and y-axes. Similar to OriArm, it requires a pair of absolute orientation sensors: one in the form of a ring worn by the user, and another one worn on the wrist. We first derive the orientation of the finger with respect to the wrist by calculating the rotations between the angular positions of two sensors. Then we map the orientation to the corresponding axes. A wrist-rotating movement, similar to the one used in OriArm, is used as the selection gesture (left click).

We designed OriFinger to provide fine-grained control for wearables that do not possess such accurate cursor control capabilities. For instance, Google Glass uses an on-device touch-pad, which only allows the user to perform one-dimensional gestures, such as directional sweeps. Missing the precise control of the cursor in the interface limits the functionality. For instance, given the capability of moving a cursor on the screen, a user can potentially input text on a soft keyboard projected on the heads-up display. It can be even more efficient if combined with a sliding-based text-entry method [4]. It can also be used to interact with input devices where screens are usually missing, such as a projector or a large-scale public screen.

OriFinger gestures can also be performed in a discreet fashion when needed, to make the interaction process socially appropriate and private. For instance, instead of pointing

[4]http://www.swype.com/

the finger forwards, the user can discreetly move fingers to input on a device while keeping the hands down. Thus, the user can even interact with their devices without catching others attention which may interrupt conversations.

5.2.4 OriTrak System

System Setup

Hardware design:

We used three form factors to attach the sensors to different parts of the body (wrist, finger, torso) as shown in Figure5.18. To place the sensor on the finger, we designed a 3-D printed ring and glued the sensor on the top. To place the sensor on the wrist, we made a wristband, which is constituted a 3D printed case to hold the sensor, and a purchased watch-band to attach the case to the wrist. In order to place the sensor on torso, we attached the sensor to a 3D printed case and taped the case on the belt to make it wearable.

Each form factor above has a Bosch BNO055 to sense the absolute orientation. It is a 9-degree IMU with a built-in processor running internal algorithms to derive reliable absolute orientation from the IMU. The BNO055s are connected to an Arduino Uno via the Inter-Integrated Circuit (I2C) and sampled at 100Hz. To provide a more comfortable wearable experience, we replace wire connection with a Bluetooth module, which communicates sensor data to a MacBook Pro for real-time analysis in our prototype. The whole hardware system is powered with a Li-Po battery (7.4V, 500mAh) as the power source.

Software Infrastructure: To receive sensor data via Bluetooth connection, we run a Node.js server script on a MacBook Pro. The received data is processed in JavaScript using the library THREE.js [5] for angle transformation. We choose JavaScript as the major programming language because of its excellent compatibility across different platforms and the good support for real-time data visualization. Using JavaScript, we were able to test early prototypes on laptops and smartphones.

[5] https://threejs.org/

Data Processing Pipeline

In this section, we will describe the data processing pipeline for OriTrak. First, we present how we derive the orientation between two devices using absolute orientations. Then we describe the gesture recognition algorithms for OriFinger and OriArm, respectively.

Calculating Relative Orientation

As discussed in previous sections, absolute orientation represents the angular position of the sensor with respect to the coordinate system of the earth. Consequently, absolute orientation on a wearable device is dependent on the direction the user is facing, such that the body has to be still during the whole interaction process. In order to overcome direction dependency, we calculate the orientation between coordinate systems of two absolute orientation sensors. that is, we calculate the orientation of the wearable with respect to a second sensor's coordinate system. For instance, in OriArm, we use two absolute orientation sensors, one on the wrist and one on the user's torso. Instead of directly using the absolute orientation from the sensor on the wrist, we calculate the relative orientation between the two sensors. This derived orientation represents the angular position of the wrist with respect to the body, which should be the same if the user keeps the same posture but faces a different direction. Because the relative orientation between the wrist and body is not changing.

For the following descriptions we define the two absolute orientation sensors as the sensor of interest (SI) and the sensor of reference (SR), which represent the sensor we want to track and the sensor used for deriving relative orientation (SD), respectively. For OriArm, SI and SR are placed on the wrist and the torso, respectively. By placing SR on the torso, we can acquire the three-dimensional angular position of the wrist with respect to the body. Similarly, for OriFinger, the angular position of the finger with respect to the wrist is derived by using the absolute orientation from SI and SR which are now placed on the finger and wrist, respectively. As a result, the derived angular position would not be

influenced by the arm and body movements as long as the finger keeps the same posture against the wrist.

$$quaternion_{SD} = quaternion_{SI} \times Inverse(quaternion_{SR}) \tag{5.14}$$

The raw format we received from absolute orientation sensors are represented as quaternion, which represents the three-dimensional rotations. Different from Euler angles which uses three variables, quaternion has four parameters to describe rotations. A further discussion on quaternion is beyond the scope of this paper. More information are accessible online[6]. After we receive the quaternion vectors from the two sensors, we calculate the orientation between SI and SD using apply equation 5.14, in which $quaternion\_SI$ is multiplied by the inverse of the $quaternion\_SR$. The derived $quaternion\_SD$ is transformed to Euler angles for the ease of gesture recognition. We apply an average filter of size 8 on the derived relative orientation, which is used to later data processing.

Gesture Recognition for OriArm

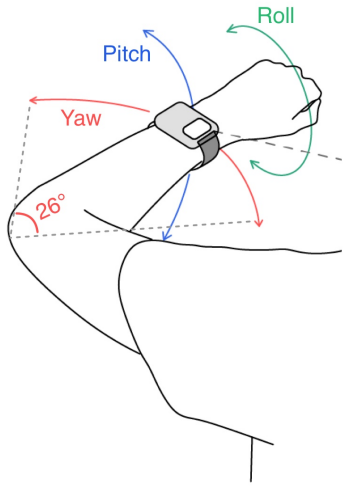


Figure 5.20: Gesture Design of OriArm

As discussed in Section 5.2.3, OriArm supports three different one-handed gestures to

[6]https://en.wikipedia.org/wiki/Quaternion

input on a smartwatch, including gestures for activation, menu navigation and selection. We will present how we implement OriArm in the following sections.

Sensor Direction Calibration: OriArm operates under the assumption that the z-axis of SR, which is placed on the body (torso), is aligned well with the direction of the body as shown in Figure 5.18. However, in reality, it is very challenging to perfectly mount SR on the body such that, the z-axis remains perpendicular to the center of the body surface. Therefore, an angular offset would likely exist between the z-axis of SR and the direction of the body. To address this issue, we quantify the offset by introducing a calibration gesture. To perform this calibration gesture, the user needs to lift up the arm with SI and align the arm with the direction of the body. When doing so, the y-axis of the sensor (yaw) is parallel to the direction of the body. We record the difference of yaw angle between SI and SR, which is subtracted from the later yaw angle of absolute orientation on SR for compensating the angular offset. For instance, if we record a 10 degree offset, we will subtract this 10 degree from all the yaw angle on SR later. When the user points the arm towards the front, left, and right of the body, the yaw value of the orientations between the wrist (SI) with respect to the body (SR) are 0, 90, and 270 degree, respectively. By introducing a –simple, that is with little effort to perform single initial– calibration gesture, SR can be placed on any part of torso without much limitation on sensor placement.

Unlock Gesture: OriArm allows the user to unlock the screen of a watch using one-handed gestures. The unlock gesture is constituted of two steps: input intention detection, and gesture-based confirmation.

In the first step, the user signals the input intention by holding the arm at a predefined unlock position. Before using OriArm, a prospective user customizes this position by lifting the arm and holding it at the position where they feel most comfortable to interact with the watch (Figure 5.20). Because different users have different heights and arm lengths, the angular position of the watch with respect to the body varies. Our system records this angular position (denoted as OA) for each user to recognize customized unlock gestures.

When the screen is locked or dimmed, OriArm keeps examining the incoming absolute orientation from SI and SR to detect the unlock gesture to start. We calculate the relative angular position between SI and SR, which are compared with OA to find the difference on roll, yaw pitch angles. If the calculated difference of roll, yaw, and pitch angles falls into the range of 35, 25 and 25 degrees respectively for 200 ms, the first step of unlock gesture is completed. These thresholds were empirically determined. The 200ms timeout is used to prevent false-positive errors caused by random arm movements .

Once the system detects the input intention, the screen lights up and the user needs to perform a wrist-rotating movement to conclude the unlock gesture. We apply a rule-based method to detect this gesture by examining the angular acceleration and velocity of the roll angle. Once both values pass empirically-determined thresholds in a window of 600ms, a gesture is detected. The system unlocks the screen and proceeds to the next step: menu navigation.

Menu Navigation: Once the screen is unlocked, the user can navigate through a menu by moving the arm towards or away from the body, which results in changes of the yaw angle (Figure 5.20). For the sake of social appropriateness, only the movement falling into the range of 26 degrees whose center is at the user-defined OA position, is functional. For instance, if the yaw angle of OA is 20 degrees, the active range in yaw for interaction is from 7 to 33 degrees. We use the 26 degrees of arm movement in yaw angle to scroll through the whole menu list. Therefore, the more items are in the menu, the less angle range is assigned to one menu item. For instance, if the menu has two items and the activation position is 20 degree on yaw angle, the arm movement on yaw angle between 7 to 20 degree will highlight the first item and the movement between 21 to 33 degrees in yaw angle will highlight the second item. Similar to the menu list on a touchscreen of a smartwatch, an item is select-able only when it is at the center of the menu.

One-handed Gesture for Menu Selection: The user can select the item at the center of the menu by performing the wrist-rotating gesture. The detection of this gesture is similar

to the movement used in the unlock gesture, which is detected by analyzing the angular velocity and acceleration of the roll angle. The only difference here is that once the system detects the acceleration of the roll passes a threshold, it locks the menu to the currently highlighted item at the center until the gesture is concluded. Such that rotation of the wrist would not influence the selected item. Once a selection gesture is detected, The highlighted item is marked as selected and the corresponding functions is activated where applicable .

Gesture Recognition for OriFinger

OriFinger allows the user to control a cursor in two dimensions by moving a finger and trigger a left-click event by rotating the wrist.

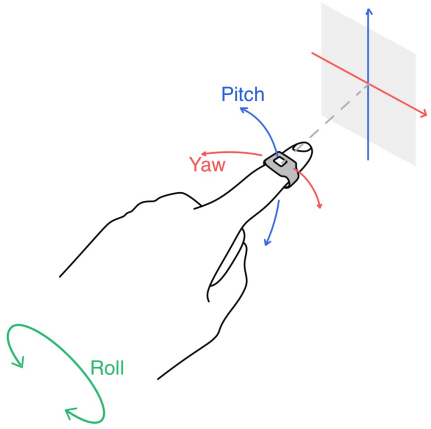


Figure 5.21: OriFinger Gesture

Cursor Movement: OriFinger uses two absolute orientation sensors, placed on the index finger and wrist with the aid of a ring and wristband, respectively, to derive the angular position of the finger with respect to the wrist. To control the movement of a cursor, we map the angular displacements in yaw and pitch angle to the cursor movement in x and he y-axes (Figure 5.21). Intuitively, moving the finger up and down, or left and right, would move the cursor in y-axis or x-axis respectively.

In our early explorations, we found different users have ranges of movement while moving index fingers towards different directions, namely the yaw and pitch angles. Sim-

ply applying the same range of angular positions for all users while mapping the angular displacements with the cursor position, would likely influence the interaction experience, as some may not even be able to move their fingers in the same way. To address this issue, we ask each user to define a central position which intuitively gives them enough freedom to operate in all four directions (up, down, left, right). We record the orientation of this position (denoted as OC), defined as the center of the screen.

We use equations 5.15 and 5.16 to map the orientation to the x- and y- coordinates on the screen. First, we calculate the difference of yaw and pitch angles between OC and the current orientation. These differences are then divided by the empirically determined ranges to acquire the ratio position. We empirically set the range to be 25 degrees and 35 degrees for yaw and pitch angles respectively. To translate the ratio position to the coordinates on the screen, we multiply the ratio position by the dimension of the screen. The results are the transformed coordinates on the screen with the assumption that the original point $(0, 0)$ is at the center. However,the actual original position is usually at the top-left corner on the screen. To compensate these offsets, we add half of the screen dimensions (x and y) to the result in the last step. As a result, the top-left corner on the screen is the new original point $(0, 0)$ in the system. In order to make the cursor position more stable, we apply a Kalman filter on the calculated x,y coordinates.

$$X_{coordinate} = (Yaw - Yaw_{OC})/(Yaw_{Range}/2) \times X_{ScreenDimension}/2 + X_{ScreenDimension}/2; \tag{5.15}$$

$$Y_{coordinate} = (Pitch - Pitch_{OC})/(Pitch_{Range}/2) \times Y_{ScreenDimension}/2 + Y_{ScreenDimension}/2; \tag{5.16}$$

Button Click Event: Similarly to OriArm, OriFinger also adopts a wrist-rotating move-

ment with which the user can trigger a click event on the screen. The detection algorithm is similar to the one presented for OriArm, except we use the roll angle from SR for gesture detection instead of the relative angular position between SI and SR.

5.2.5 User Study

To evaluate OriTrak, we conducted a lab-based user study with 11 participants having an average age of 22.8 (five male). In the study, all participants tested OriArm first and OriFinger second. Each study was no more than two hours long. The hardware setting used is same to what we have discussed in section 5.2.4.

OriArm Evaluation

In the OriArm evaluation, the participants were asked to select a highlighted item from a menu list on a wrist-mounted device using OriArm and a touchscreen, respectively. The purpose of this comparison study is to demonstrate that interactions through OriArm can be pursued as effective as more conventional (two-hand) wearable interactions and in comparable time, however thereby putting a much lower burden on the user through, in this case, enabling one-handed input. We compare interaction capabilities provided by OriArm to conventional touchscreen based interaction with wearables.

Procedure: For the OriArm evaluation, we used a smart phone (Samsung Galaxy S7 edge) on the wrist to simulate a smartwatch, instead of using an actual one. The reason for not using an actual watch is that we found non-negligible delays randomly ocurring while we transmitted data from the laptop to the watch (LG G Watch) due to the internal memory optimization mechanism in Android Wear system. Such a delay can significantly influence the interaction experience. Therefore, we decided to use a smartphone as the testing device which showed the testing interface, fetched the results from the laptop, and updated results on the screen. We developed a HTML5 JavaScript-based web application running on a web-browser that showed a menu on the right side on the screen (Figure 5.22). We limit

the dimension of the menu to be 15mm (width) $\times$ 32 mm (height), which is comparable to the dimension of a menu list displayed on a commodity smartwatch.

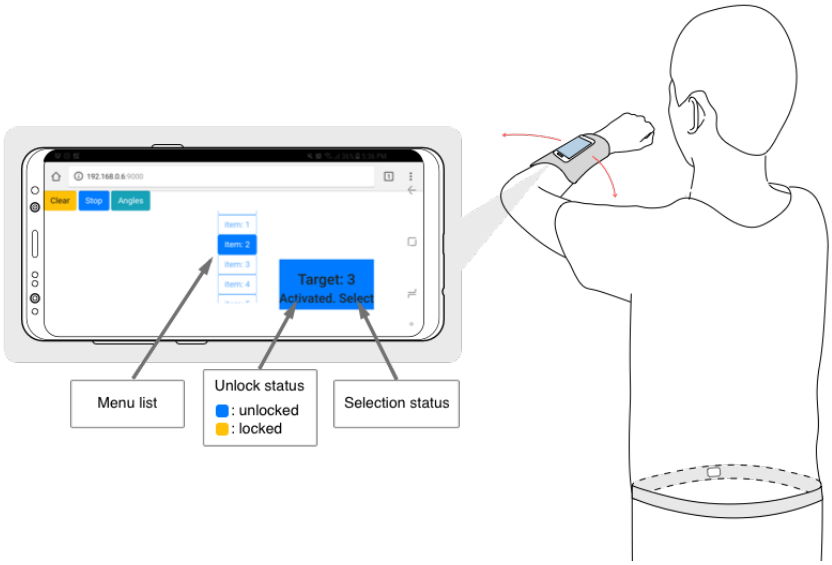


Figure 5.22: The testing user interface of OriArm

At the beginning of the study, a researcher helped the participants to wear the wrist band with the SI and the smartphone, and put the belt with SR on the body. Then the researcher demonstrated how to perform OriArm's one-handed gestures to interact with a "smartwatch". The participants were allowed to practice before they actually started the testing tasks.

In order to compare the interaction experience between OriArm and a touchscreen on a watch as we already discussed, the participants were requested to complete the same tasks using both, OriArm and the touchscreen. Each task had three separate sessions, in which the users were instructed to select a target item from a menu of 2, 6, and 18 items. Each session contained two trials. In each trial, the participant was asked to select a randomly generated target item from the menu 4 times. For the ease of comparison, we use the same sequence of tasks within each trial for both OriArm and touchscreen tests.

Every participant defined their own unlock position before sessions began. To simulate a real watch interaction experience, the participants were asked to put their arm down before

and after making each selection. Each menu selection can be divided into the following four steps:

Step 1: At the beginning of each selection task, a "ding" sound was played as a reminder. The participants were instructed to lift their wrist up to their predefined unlock position on hearing the sound. Once the system successfully detected the unlock position, the color of the object on the screen was changed from blue to yellow. In the testing of touchscreen, the system did not look for this unlock position.

Step 2: The participants performed a selection gesture by quickly twisting the wrist (OriArm) or tapping the screen (touchscreen) to confirm the unlock gesture.

Step 3: Once the screen was unlocked, the participants started scrolling down or up the menu by either moving their arms towards or away from the body (OriArm), or by sliding on the screen (touchscreen). The goal is to move the target highlighted with blue color to the center of the list.

Step 4: Once the target item was moved to the center at the list, the user confirmed the selection by performing the twist-rotating gesture (OriArm) or tapping on the screen (touchscreen) to perform a selection gesture.

Once a session of each interaction was concluded, the participants were asked to fill out the NASA_TLX survey to evaluate their mental and physical efforts during the session. In total, each participant made 4 (repeats) $\times$ 2 (trials) $\times$ 3 (menus with 2,6,18 items) $=$ 24 selections for each interaction scenario (OriArm and touchscreen). In order to evaluate the input efficiency, we recorded a time-stamp on each step. The time when the "ding" sound stopped was recorded as the "Start Timestamp". The time when the user touched the screen or rotated the wrist to unlock the screen was recorded as the "Unlock Timestamp". When the user tapped the screen or rotated the wrist to make a selection, the time was recorded as the "Selection Timestamp". We define the time duration between the "Start Timestamp"

and the "unlock Timestamp" as the unlock time and the time duration between the "Unlock Timestamp" and the "Select Timestamp" as the scrolling time.

Accuracy:

The average accuracies for using OriArm and touchscreen are shown in Figure 5.23. The participants were able to make selection from menu with 2, 6, 18 items with the accuracy of 100%, 93.2%, 85.2% using OriArm and 100%, 100%, 98.9% using the touchscreen. If the selected item did not match the target item, we counted it as a error. The accuracy to make a selection dropped when the number of items increased for both interactions. There was no statistically significant difference on the accuracy between two interactions for the menu with 2 items (the same accuracy for both interactions) and the menu with 6 items ($p = 0.103, p > 0.1$). When the menu had 18 items, there was a significant difference between the two interactions ($p = 0.018, p < 0.05$). This analysis indicates that when the number of items is relatively small, OriArm presents a similar accuracy of making a selection as when using the touchscreen-based method.

This is not a surprising result to us. Since when the number of items increases to 18, each item only maps to a spatial angular range of $1.44 = 26$ (total range in yaw angle) / 18 (items) degrees. In order to select the correct item, the participants have to keep the wrist steady within a range of 1.44 degree before performing the selection gesture, which is both mentally and physically demanding. Recalling from the section 5.2.3, we limit the movement to such a small range because of social appropriateness. One obvious solution to improve the performance is to increase the range of movement when the social appropriateness is not an important concern. For instance, if the user is alone while using the watch, it would not be inappropriate to move the arm slightly beyond the 26 degrees range in order to interact with the watch.

Even though OriArm demands more time to select an item as compared to the touchscreen, in scenarios where the other hand is occupied, OriArm is still more efficient when operating the watch, as compared to freeing one hand first in order to use the touch screen.

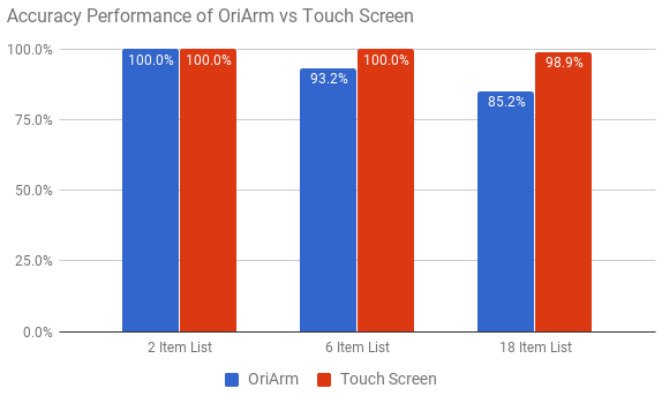


Figure 5.23: The Accuracy of OriArm study

Time Performance:

Figures 5.25 and 5.24 show the average unlock time and scrolling time on the menu of $2, 6, 8$ items. The bar with blue and red color represent the time for OriArm and touchscreen interaction respectively. The average unlock time for using OriArm and touchscreen was 1.55 seconds and 1.10 seconds, respectively. The average scrolling time for using OriArm and touchscreen was 1.88, 3.72, 5.30 seconds, and 1.30, 1.53, and 2.36 seconds for the menu of 2, 6, 18 items, respectively. The participants on average needed more time to make a selection using OriArm than using the touchscreen. Statistical significance test shows that there is a significant difference across all three sessions ($p = 0.03, p = 5.32e-5, p = 1.45e-5$ for menus of 2, 6, 18 items). It is worth mentioning that we implemented the menu list on a web page, where the speed of scrolling was faster than the actual menu widget on a smartwatch. Therefore, potentially the user would take longer to make menu selection on a real smart watch.

Cognitive load: Based on the NASA-TLX forms we collected from the participants, in average, the task load index(TLI) for making a selection from menu of 2, 6, 18 items are $29.1, 28.9, 36.7$ for using OriArm, and $22.8, 18.7, 19.9$ for using touchscreen as shown in figure 5.26. It shows that using OriArm to make a selection presents a higher task load than using touchscreen on make a selection. However, there is no significant difference on TLI

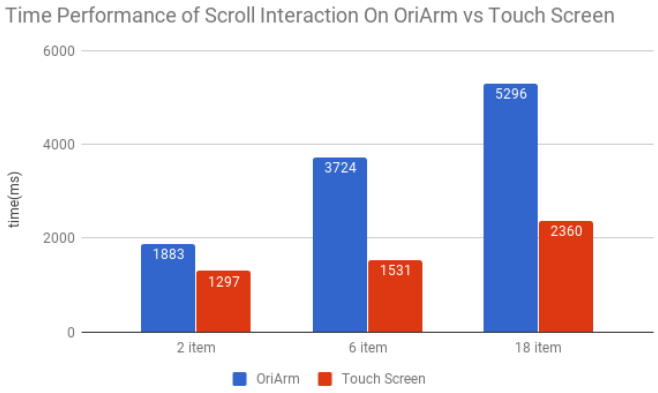


Figure 5.24: Time performance for 2 menu items

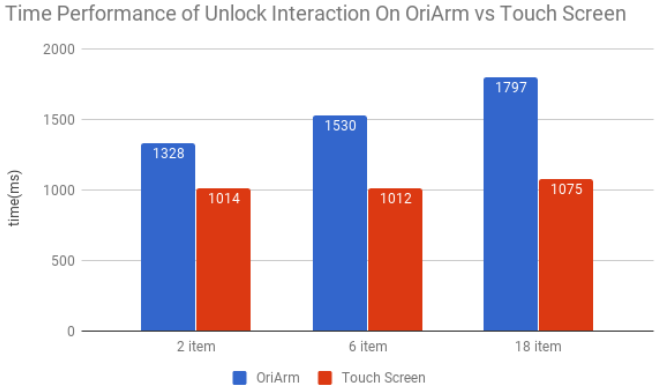


Figure 5.25: Time performance for unlock

when the menu only has 2 items (Rp=0.39, p¿0.1R).

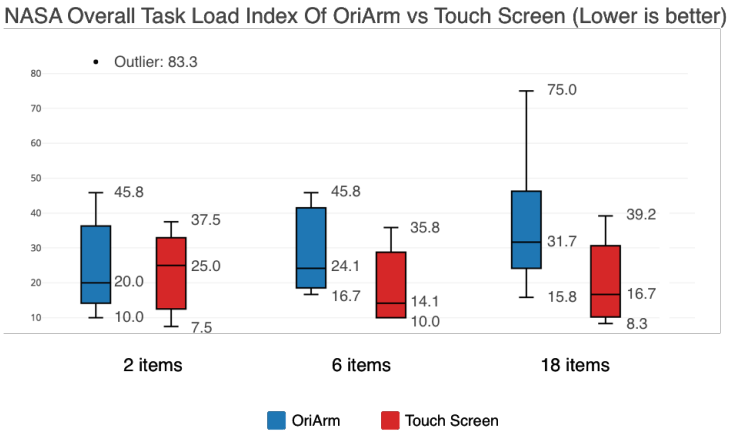


Figure 5.26: The cognitive load of OriArm study

OriFinger Test

Procedure:

The procedure of OriFinger test is similar to OriArm test. At the beginning of the study, a researcher helped the participants wearing the wristband on the right wrist and the ring on the right index finger. Then the researcher demonstrated how to perform the OriFinger gestures to control the cursor on the screen. The participants were allowed to practice OriFinger gestures until they felt comfortable to start the testing sessions.

During the whole study, all participants were instructed to keep their hands and arms down, such that they can perform OriFinger gesture in an eyes-free manner. Intuitively, if the users had kept their hands and arms up, they would probably perform these gestures faster and more accurate, because they can see their hands while performing gestures. However, as we have discussed in Section 5.2.3, one important design principle of OriTrak is to design gestures that are socially appropriate. Asking the participants to perform the gesture in an eyes-free fashion can be mentally and physically challenging, but it is less disturbing to people around. Before the testing sessions started, we asked each participant to define their central position which was used to calibrate the mapping mechanism as we have discussed in the previous section.

In the testing sessions, the participants were asked to conduct a Fitts' law study, which was designed to evaluate the rapid and targeted movement on a 2-D screen. We ran the standard Fitts' law study software [7] on a Macbook Pro in this study. Nine targets were displayed on the screen in the test. The participants were asked to move the cursor using OriFinger to select the highlighted target as soon and accurately as possible. After all nine targets were selected, a trial concluded. We set the parameters on the software to be 9 targets, two target sizes $(50, 90)$ and two target distances $(350, 500)$. We tested OriFinger in two settings: In the first setting, the participants used their left hand to tap on a trackpad as the left-click event on the screen (denoted as trackpad-assisted input TI) and use the right

[7] http://www.yorku.ca/mack/FittsLawSoftware/

index finger wearing a ring to move the cursor. In the second setting, the only difference compared to TI is that the left-click event was triggered by the wrist-rotating gesture (denoted as only gesture-based input GI). In both settings, the participants used OriFinger to move the cursor on the screen. In total, each participant completed four sessions for each of the two settings. The first session was the practice session and the last three sessions were testing sessions. In each session, the user completed 4 trails (2 target sizes $\times$ 2 distances). The sequence between the two settings was randomized.

The reason we tested OriFinger with the trackpad is that we found the bottleneck of the input efficiency was the wrist-rotating gesture. Rotating the wrist may influence the position of the cursor. In order to perform the gesture accurately, the user tends to be more careful, which slowed down the interaction flow. Replacing this wrist gesture with other more efficient gestures (e.g., head gesture) can potentially improve the input efficiency. Using the trackpad to make a selection can test the upper limits that OriFinger can potentially achieve in the future if given more efficient selection gesture.

Movement Time: In testing sessions (session $2 - -4$), the average movement time of making a selection for GI was 2336.21ms with a standard deviation (SD) of 483.52, and for TI was 1384.59ms (SD = 180.14). The difference is considered to be statistically significant ($p = 1.03E - 15, p < 0.001$). The analysis of the movement time by the effective index of difficulty (IDe) also indicates that it takes longer time to accomplish more difficult tasks.

Learning Effect We ran an ANOVA analysis on the results of average moving time to understand the learning effect. Figure 5.27 shows the learning effect for two settings on average movement time of making a selection. We observed significant differences on average moving time between session 1 (practice session) and session 4 (last testing session) in both GI ($F = 7.32, p < 0.05$) and TI ($F = 5.72, p < 0.05$). However, the difference between the second and fourth session was not statistically significant on both GI ($F = 1.54, p > 0.1$) and TI ($F = 0.75, p > 0.1$). Also, there was no statistically significant difference between the third and fourth session on both GI ($F = 1.45, p > 0.1$)

and TI ($F = 0.05, p > 0.1$).

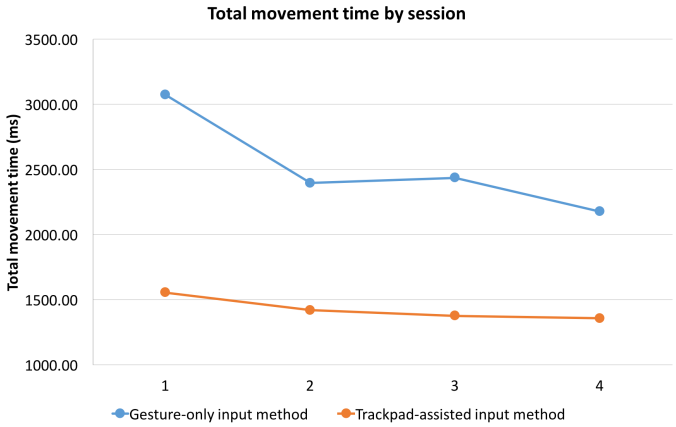


Figure 5.27: Learning curve of OriFinger study

Error Rate and Throughput In testing sessions (session $2 - -4$), the average error rate of making a selection in Fitts' law study was 5.89% (SD = 4.49) for GI and 3.37% (SD = 2.84) for TI. The difference was statistically significant ($p = 0.008, p < 0.05$). The average throughput (bps) was 1.45 (SD = 0.29) and 2.39 (SD = 0.36) for GI and TI respectively. There was a statistically significant difference between these settings ($p = 2.02E - 17, p < 0.001$) on throughput. One possible reason for these difference is performing a wrist-rotating gesture was more physically and mentally demanding than tapping on the trackpad. However, participants demonstrated the similar accuracy using GI compared with using TI. Considering each user only used OriFinger for about 30 minutes, we expect the user can achieve an even higher input efficiency if given more practice in the future.

Cognitive Load: The overall NASA task loads for GI and TI are shown in Figure 5.28. We found a significant difference ($p = 0.023, p < 0.05$) between the two settings. The task load for GI was higher than TI. Some participants reported it was challenging to perform a gesture to make a selection when the cursor approached the edge of the screen.

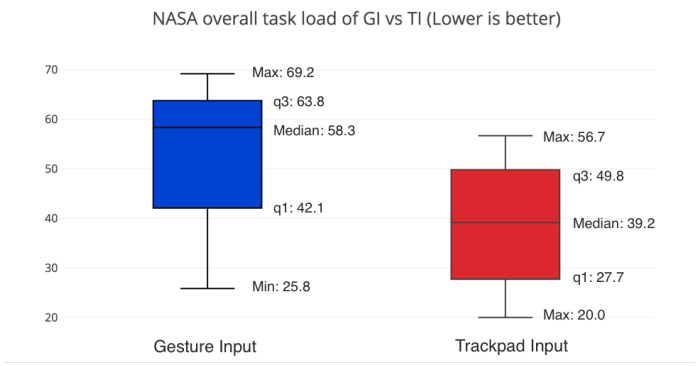


Figure 5.28: Cognitive load of OriFinger study

Sensor Drifting Test

One common issue for motion-based tracking system is the drifting effect over time. These systems calculate the position by accumulating the displacements between data samples. For instance, it integrates the angular speed (gyroscope) or the acceleration (accelerometer) over time to estimate the position. Usually, the longer time the sensor has been used or more distance the sensor moves, more accumulated errors of the tracking position were observed.

OriTrak uses Bosch BNO055, which is a 9-axis IMU with an on-board processor and algorithms to calculate absolute orientation. According to its data sheet, this sensor has implemented an internal algorithm to correct drifting effects over time. In our user study, we have not observed any significant drifting issues. To further quantify the drifting effect, we conduct an experiment on our system under three settings.

Drifting over time: In the first setting, we placed the two sensors on fixed locations for 30 minutes. We recorded the orientation between SI and SR calculated by our system. There was 0 degree drifting during the 30 minutes. We think the internal drifting correction algorithm was playing a key role to maintain the stable orientation while the sensors are still.

Drifting when both sensors move:

In the second setting, we tested the drifting errors when SI and SR were moving together on the same rigid object. In this scenario, the angular position between SI and SR was

invariant during the experiment. As figure 5.29 shows, a wooden board of 52cm that is similar to the length of the arm was attached on the mount, such that it rotates in the same direction and same angle as the mount rotates. SI and SR were attached at two ends on a wooden board. SI was placed on the farther end of the board. We configured the mount such that it only moves vertically, resulting in pitch changes, or horizontally, resulting in yaw changes. We manually move the mount to the target angle each time with an angle meter on the scale. Because each line of the angle meter represents 2 degrees. Though we have done our best to match align the wooden board with the line, our confidence level of the ground truth is ± 0.5 degrees.

To understand the influence of different moving angles on the drifting effect, we moved the arm of the mount for a different degree α on yaw or pitch each time. For yaw angle, we moved the arm with 6 settings. Each setting has a different moving angle α in yaw, which is $30, 60, 90, 120, 150, 180$ degrees, respectively. For pitch angle, due to the limits of the mount, we only evaluated three settings with the moving angle α in pitch, which is $30, 60, 90$ degrees. For each setting, we moved the board horizontally or vertically for five trials. where each trial was constituted of four steps:

- The arm started at the center where the yaw or pitch was 0 degree. First, the arm turned anti-clockwise for α degrees horizontally or up for α degrees vertically, which moved the arm to -α degrees in yaw or pitch.
- Second, the arm turned clock-wise for α degrees horizontally or down for α degrees vertically, which moved it back to the center position (0 degrees in yaw or pitch).
- Third, the arm turned clock-wise for α degrees horizontally or down for α degrees vertically, which moved the arm to + α degrees in yaw or pitch.
- Fourth, the arm turned anti-clockwise for α degrees horizontally or up for α degrees vertically, which was at the center position again (0 degrees in yaw or pitch).

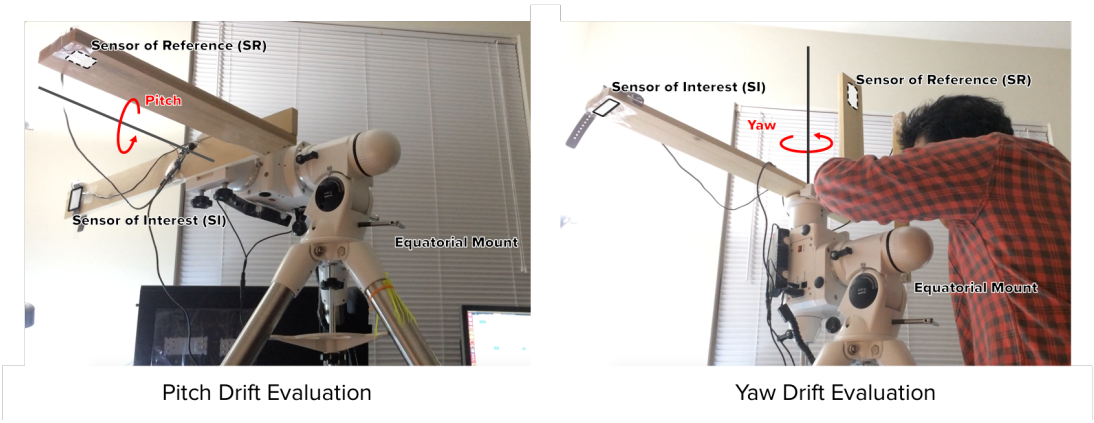


Figure 5.29: System Setup for Drifting Effect Test When both SI and SR moves

After each step, the angular position between SR and SI was recorded, which was then compared to the ground truth for calculating drifting errors. Since both SI and SR were moving together, we expect the angular position between SI and SR to be 0 during the whole experiment. Any offset was counted as errors. As a result, we moved the arm horizontally or vertically for 20 (5 trials $\times$ 4 steps) times for α degree.

Figure 5.30 and 5.31 shows the recorded drift errors after each step. After moving the arm vertically (pitch) for 600 (30 degrees $\times$ 20 steps), 1200 (60 degrees $\times$ 20 steps) and $1,800$ (90 degrees $\times$ 20 steps), when the arm returned to the starting position, the drifting errors on pitch were $-0.15, 1.12, -0.10$ degrees, respectively. After moving the arm horizontally (yaw) for 600 (30 degrees $\times$ 20 steps), $1,200$ (60 degrees $\times$ 20 steps), $1,800$ (90 degrees $\times$ 20 steps), $2,400$ (120 degrees $\times$ 20 steps), $3,000$ (150 degrees $\times$ 20 steps) and $3,600$ (180 degrees $\times$ 20 steps), when the arm returned to the starting position, the drifting errors on yaw were $1.00, 1.43, 0.68, 4.17, 1.02$, and 4.07 degrees.

Drifting when only SI moves: The procedure of experiment on the third setting was exactly the same as the second one, except SR was fixed on the mount and only SI which was fixed on the farther end of the wooden board was moved with the arm during the experiment. This setting simulates the most common scenario in daily activities where SR (body, wrist) is relatively still and only SI (wrist, finger) moves. After each step, the angular

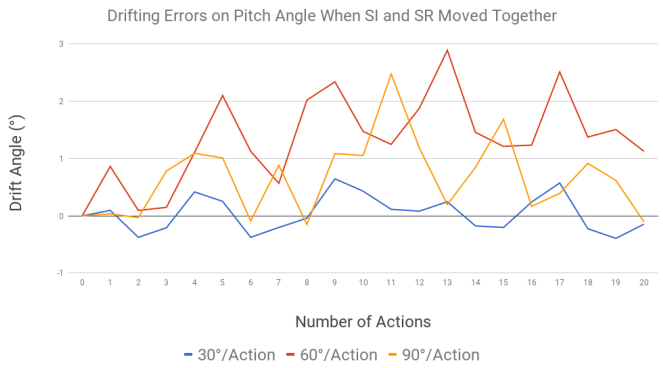


Figure 5.30: Drifting errors of pitch when SR and SI moved together

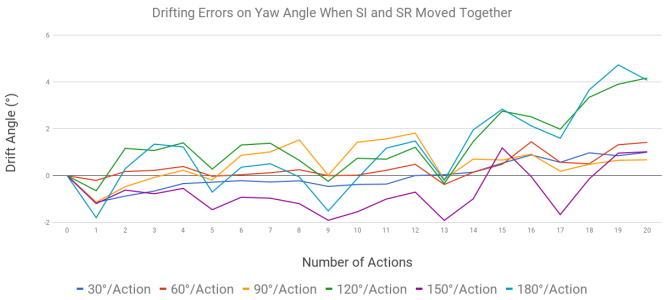


Figure 5.31: Drifting errors of yaw when SR and SI moved together

position between SR and SI was recorded, which was then compared with the ground truth for calculating drifting errors.

Figures 5.32 and 5.33 show the recorded drift errors after each step. After moving the arm vertically (pitch) for 600 (30 degrees $\times$ 20 steps), $1,200$ (60 degrees $\times$ 20 steps) and $1,800$ (90 degrees $\times$ 20 steps), when the arm returned to the starting position, the drifting errors on pitch were $-1.04, -0.45, 1.84$ degrees, respectively. After moving the arm horizontally (yaw) for 600 (30 degrees $\times$ 20 steps), $1,200$ (60 degrees $\times$ 20 steps), $1,800$ (90 degrees $\times$ 20 steps), $2,400$ (120 degrees $\times$ 20 steps), $3,000$ (150 degrees $\times$ 20 steps) and $3,600$ (180 degrees $\times$ 20 steps), when the arm returned to the starting position, the drifting errors on yaw were $-5.50, 1.12, -1.31, -1.86, 1.39$, and -0.81 degrees.

Results Discussion: The above results present a consistent results on drifting effect compared to what we observed in the user study. Given these drifting error results, the

interaction experience of OriTrak should not be significantly influenced, especially considering that most of OriTrak interaction only use the relative displacement of angular positions.

Comparing the results between the second and third setting, it looks like the system suffered more drifting errors when only SI moved. One possible reason was the human error (± 0.5 degrees) we mentioned earlier did not influence the experiment results in the second setting, but had contributed to the errors in the third setting. Because the ground truth of yaw or pitch between SI and SR was always zero degrees even the arm of the mount was not placed exactly at the target angular position in the second setting. For instance, if the arm of the mount is supposed to be moved at 60 degree at yaw, we actually place it at 58 degree due to the human error, which introduces a 2 degree offset. In the second setting, even this 0.5 degree offset exists, the ground truth is invariant which is 0 degree. Therefore the difference between the calculated value of yaw and the ground-truth is not changed. However, in the third setting, to calculate the drifting error, we compare the difference between the calculated value of yaw with the ground-truth which is supposed to be 60 degree. The 0.5 degree offset will be counted as the system error in our metrics.

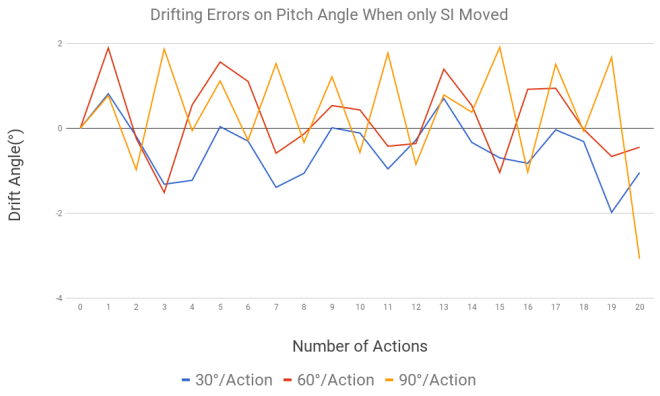


Figure 5.32: Drifting errors of pitch when only SI moved

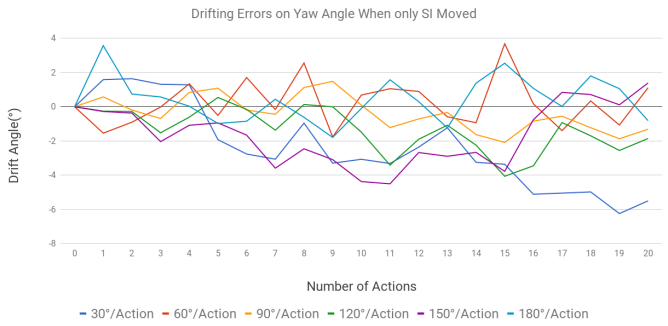


Figure 5.33: Drifting errors of yaw when only SI moved

5.2.6 Discussion

Improvement on Gesture Design

In OriArm, we directly map the 26 degrees range of the yaw angle to the whole menu list. Therefore, the more items are in the list, the thinner each range of the yaw angle can be sliced to each menu item. This potentially limits the number of items that a user can efficiently interact with using OriArm. An alternative design solution can solve this issue. It maps each item with a constant range of yaw angle. For instance, if we map the 6 degrees with each item in the menu, only 4 items can be displayed and selected in a range of 26 degrees. To interact with more items in the list, the menu keeps scrolling up or down to shift the content of the 4 items that are displayed and select-able, if the wrist approaches the edge of the range (0 degrees, 30 degrees). Once the target item appears on the screen, the user can move the wrist back into the predefined range to make a selection.

Applications

In this section, we discuss the applications for OriArm and OriFinger.

OriArm

In the user study, we demonstrated using OriArm to interact with a menu on the smartwatch by using the yaw angle from orientation. The menu is a metaphor representing the one-

dimensional horizontal or vertical list which is widely used in the user interface on the smartwatch. To navigate through these lists, the user needs to repeat sliding vertically or horizontally on the touchscreen. With OriArm, the user can move the arm to navigate through these one-dimensional lists with only one-hand. It can be used to make a selection on a list of items such as buttons, applications, songs or files. Furthermore, it can also be used to input in the scenarios where more precise control is needed, such as volume and numbers.

OriFinger: OriFinger can be used as an alternative input device for mouse on a wearable device, in scenarios where using a physical mouse is not feasible. For instance, most Head-Mounted displays (e.g., Google Glass, VR/AR headset) do not have a mouse to input. Some only have one-dimensional input capabilities. OriFinger can provide 2-dimensional input resolution to these devices. It can also be used to input on devices that do not have an input device such as a projector or a large-scale display. Furthermore, for devices that have a tiny size of the touchscreen (e.g., smartwatch), the input resolution is limited by the screen. For instance, it is challenging to sign a full name on the smartwatch because the size of the finger is too large compared with the touchscreen. On all the devices above, OriFinger can be used for gesture recognition, button selection, painting, signing a signature etc.

Activity recognition using OriTrak

OriTrak tracks the relative orientation between a wearable and another sensor somewhere on the body. In this paper, we only demonstrated the possible interactions with wearables supported by OriTrak. However, knowing the relative angular position of a wearable with respect to the body can be used to solve a broader set of problems beyond mere interaction, such as human activity recognition.

Human activity recognition using body-worn inertial measurement units is one of the main research and application area in mobile and ubiquitous computing. Over the years a

tremendous variety of methods and applications have been develeoped (see, for example, [95] for a recent survey of the field). With the advent of smartwatches, wrist-worn IMUs have become very popular for activity recognition tasks, which is mainly reasoned by the convenient and especially familiar (to users) sensor position that comes with a watch-like devices. For example, wrist-mounted IMUs as they are integrated in off-the-shelf smartwatches have been used to detect eating moments [96]. The majority of HAR systems have used gyroscopes or accelerometers, which can, however, only provide information about the relative displacements of the wrist (in case of wrist-worn IMUs as sensing platform). Such systems only know the speed or the acceleration of the wrist movement on each of the axes, but have no knowledge of the spatial position of the wrist with respect to the body, such as where the movement starts and ends. Where the gesture is performed with respect to the body is crucial to detect certain activities. In the aforementioned example of eating detection, knowing the starting and ending angular position of each wrist movement with respect to the body can effectively reduce the false-positive/ negative predictions caused by other similar movements which cannot distinguish using only the gyroscope or accelerometer. Furthermore, if we know the angular position of the wrist with respect to the head, we can more accurately estimate whether the user is looking at the smartwatch which can be used to provide natural interaction experience by detecting the input intention of a user or enabling input using head movements.

Practicability

Requirement of wearing two devices: Our current system implementation demands two pieces of wearables to be fully functional, which might seem less practical at this moment. Because currently most users may wear no more than one wearable device. However, given the high speed growth on the wearable markets and raised attention on wearable research, it is reasonable to expect an average user will wear more than one computer on the body (e.g., watch, Glass, Jewelry, Belt) in the near future.

Using the sensors from the smartphone and smartwatch: Most of the smartphones and smartwatches are equipped with an IMU. And the Android system already provides API for retrieving orientation from these sensors. Another possible hardware configuration is using the smartphone as SR which represents the direction of the body. To calculate the angular position of the watch with regards to the body, we just need to use the absolute orientation from the watch and the phone, and also knowing the position of where the phone is placed. However, one limitation of this hardware setting is, that depending on where the phone is placed on the body, the orientation may not be stable. For instance, if the phone is placed in the pocket on the jeans, when the user is in motion, the angular position can be influenced. Furthermore, the different sample rates and synchronization between sensor data from two devices can be challenging while implementing OriTrak on this hardware setting. We plan to further test this configuration in the future.

Possibility of input with only one wearable: OriTrak adopts two absolute orientation sensors to make the interaction independent of the body direction. However, the user can still input with only one absolute orientation sensor if the body direction does not change, which is possible in certain scenarios. For instance, when a user inputs on a smartwatch, it is very common that they would not move the body while input. OriArm will still work by replacing the relative orientation between SI and SR, with the absolute orientation of the sensor on the wrist.

Using sensors from commodity devices: As we have discussed in the previous section, most commodity devices are already embedded with a nine degrees of freedom inertial measure unit, which can be used to calculate the absolute orientation with appropriate algorithms. Android already provides orientation as a virtual sensor for the developer. Therefore, our proposed interaction technique can be potentially directly applied on these commodity devices. The reason we did not use the absolute orientation from a commodity device in our study was that we did not have full control over the sensor sample rate and the data communication efficiency on a commodity Android device. A lower sampling

rate may influence the interaction experience. Our goal is to investigate the optimized interaction experience using orientation.

Orientation between any two devices

The algorithm presented in OriTrak can be used to derive the orientation between any two devices, which do not have to be wearables. For instance, if a projector/ large-scale display is embedded with an absolute orientation sensor, we can derive the orientation between the smartwatch on the user and the display device. Using the orientation, the user can perform 3-D gestures to interact with the slides or more precisely control the volume during the presentation. Similarly, the smartwatch can also be used to input on any other device embedded with an absolute orientation sensor such as a smartphone, a laptop.

Limitations

The absolute orientation is derived using a fusion algorithm on IMU including a gyroscope, an accelerometer, and a magnetometer. The sensor we used (Bosch BNO055) has built-in algorithms to correct sensor drifting over time. However, the orientation may suffer a significant offset if the system is suddenly exposed to a strong magnetic field.

Another limitation is that OriArm only allows the user to define one unlock position, where the user has to hold the arm in front of the body. In real applications, the user may want to define multiple unlock positions. For instance, if the user wants to interact with the watch in eyes-free fashion, while keeping the arm down, a new unlock position is needed. We plan to further explore this issue in the next step.

5.2.7 Summary

In this paper, we presented OriTrak, a method for continuously tracking the relative orientation of two body-mounted inertial measurement units. One of the sensing units is integrated into or attached to the wearable device, such as a smartwatch or a ring, and the other one is

mounted on somewhere else on the body, for example, on the torso, e.g., integrated into a belt. We demonstrated how to use the automatically derived relative orientation information to facilitate new, simpler interaction schemes when operating wearables thereby focusing on two implementations of OriTrak: OriArm and OriFinger. The former allows the user to input on a watch with one-handed gestures. The latter interaction scheme provides a two-dimensional finger input on a screen. A user study with 11 participants was conducted to evaluate the input effectiveness of the proposed techniques enabled by OriTrak, which shows the participants can achieve interaction performance and effectiveness that is comparable with traditional two-handed interaction but requires substantially less effort from the user.

CHAPTER 6

CONCLUSION AND FUTURE OPPORTUNITIES

In the 1990s, Mark Weiser [97] envisioned the world in the 21st century will be fulfilled with computers of three different sizes (inch-scale, foot-scale, and yard-scale). He thought everyone would own more than one computer in their daily lives. During the past decades, most of his vision has come true [98]. However, we also realize his vision is not completely precise, as some important components were missing from his original description, such as cloud computing, crowd-sourcing, and wearable computing [1]. These originally missing "computers" construct the new generation of computing, which introduces numerous opportunities and challenges. For instance, wearable computers are much smaller than traditional computers, which makes interaction more challenging. The input devices on traditional computes (e.g., mouse, keyboard) are no longer usable on most of the wearables. My thesis explores how to improve the wearable interaction experience through novel gestures.

I applied two different approaches while developing novel input techniques. One approach is to design and implement novel gestures on off-the-shelf devices by re-purposing the existing sensing modalities. Because this approach does not require any additional hardware, the invented techniques are usually much easier to be adopted into products. As such, this approach is more practical in the short-term. The other approach I take is to build customized hardware using carefully designed form factor, sensors, and processors. Although the technology using customized hardware would take a longer time to be applied to products, it allows me to explore research questions in the long-term, such as what is the next generation of wearable interaction.

For each project, I usually build the system from the bottom to the top, including understanding the physical phenomenon, building sensing platforms, designing form factors,

processing data, and designing appropriate algorithms. As I described earlier, these invented techniques not only contain easy-to-deploy solutions on commodity devices (Chapter 2) but also have techniques that use customized hardware (Chapter 3 and Chapter 4). Additionally, each input technique was evaluated with carefully designed lab-based user studies for better understanding the interaction experience. In total, I conducted eight user studies with over 120 participants, which resulted in seven paper publications and five pending patents.

6.1 Summary of Prior Chapters

In the first two chapters of my thesis, I introduced the challenges with current wearable devices. Then I explored prior research efforts on improving wearable interaction experiences and highlighted three research questions that my thesis aims to address. Finally, I presented projects in chapter 3 to chapter 5 to address the three proposed research questions.

In chapter 3, I addressed my first research question: "What novel input gestures can be developed to improve the interaction experience on wearables by only using the sensors on an off-the-shelf wearable device?" I presented two techniques, which provide a comprehensive set of input gestures by shifting the input area from the screen to surfaces around the watch. These gestures improve the interaction experience by providing additional input bandwidth for off-the-shelf smartwatches without adding any new hardware. These solutions re-purposed built-in sensors (e.g., gyroscope, accelerometer, microphones) on commodity smartwatches for gesture detection. The first one is called WatchOut, which allows the user to input by tapping or sliding on the side, bezel, and band of a smartwatch. 14 input gestures were detected using inertial measurement unit inside the watch and state-of-the-art data-processing and machine learning algorithms. The WatchOut gestures can be used as the shortcuts to quickly access applications/functions on smartwatches or alternative input methods to free the screen from occlusion. The second technology is called TapSkin, which extends the interaction to the skin (back of the hand) around the watch.

It can detect up to 12 distinct tap positions on the skin in the form of a number-pad. It uses both the inertial sensing and microphone to capture the motion and acoustic signature of different tap locations. These on-skin input gestures can be used to either access customized applications or input numbers. In total, WatchOut and TapSkin provide over 20 new gestures for commodity smartwatches without the need to modify the existing hardware. Thus, it can be relatively easy to be deployed on off-the-shelf devices. As a result, both of these interaction techniques have been licensed to a start-up to enhance the watch interaction experience.

In chapter 4, I answered my second research question: "What novel gestures can be developed to address the current input challenges on wearables using customized hardware?". I invented three new wearable input techniques using customized hardware. The first one is called FingerSound, a one-handed eyes-free input technology to recognize unistroke thumb gestures. These finger gestures are captured using a thumb-mounted ring comprising a contact microphone and a gyroscope sensor. A customized machine learning algorithm was designed to recognize up to 42 common unistroke gestures, such as numbers, Graffiti text input, and directional slides. Through these gestures, FingerSound provides an alternative wearable text input solution, which is currently missing on many wearable devices, especially for head-mounted displays (Virtual Reality Headset). The second project is FingOrbits, which uses similar hardware to that in FingerSound, but provides an additional 12 one-handed finger gestures. A user performs FingOrbits gestures by rubbing the thumb back and forth at different speeds on four different fingers. By recognizing the speed of rubbing and which finger is being used, 12 gestures (three speeds * 4 fingers) can be recognized. Similarly to FingerSound, FingOrbits can be used to interact with a heads-up display or smartwatch. The third technique is called FingerPing, which is a novel sensing technique that can recognize various fine-grained hand poses by analyzing acoustic resonance features. A surface-transducer mounted on a thumb ring injects acoustic chirps to the body, which are received by four receivers distributed on the wrist and thumb. Different

hand poses create distinct paths for the acoustic chirps to travel, creating unique frequency responses at the four receivers. FingerPing can differentiate up to 22 hand poses, including the thumb touching each of the 12 phalanges on the hand as well as 10 poses from American Sign Language (ASL). By mapping the 12 phalanges to a T9 keyboard, a user can turn the hands to an input pad, which can be used to input numbers or even text. Beyond these 22 poses, FingerPing can potentially be used to recognize other customized fine-grained poses. These three projects share many similar characteristics. For instance, they all use a thumb-ring as the form factor for one-handed wearable input. Also, they all can be used as the input for wearables, VR/AR, and potentially all personal computers in the future.

In chapter 5, I demonstrated two projects that addressed the third research question: "What kinds of low-level input events can be developed using novel sensing modalities? How would such new input events being used for wearable interaction?". These two projects are able to continuously track the position of different body parts in 3D space using different types of wearable devices. The first one is SoundTrak, an active acoustic sensing technique that enables a user to interact with wearable devices in the surrounding 3D space by continuously tracking the finger position with high resolution. The user wears a ring with an embedded miniature speaker sending an acoustic signal at a specific frequency (e.g., 11 kHz), which is captured by an array of miniature, inexpensive microphones on the target wearable device. Through a customized physics model and relevant algorithms, SoundTrak is able to track the finger position in a volume of space ($20cm \times 16cm \times 11cm$) with an average accuracy of 1.3 cm. With SoundTrak, the wearable input is no longer limited by the size of the device. A user can input freely in the space around the device. Sharing the similar spirit of tracking the positions of body parts in 3D space, I developed the second technique called OriTrak. It is a method for low effort user interaction with wearables based on continuous measurement of the relative orientation of two body-mounted inertial measurement units (IMUs). One IMU is in a wearable device, for example, a wrist-worn smartwatch. The other IMU is mounted somewhere else on the body (e.g., on a finger or at

the waist). When one of the IMUs is largely fixed with respect to the other one, the absolute position measurements from both IMUs can be used to accurately estimate the relative orientation of the other IMU to the fixed one. These automatically derived relative orientation estimates form the basis for intuitive, single-handed interaction techniques that lower the effort necessary for wearable interaction. We demonstrate the effectiveness of the proposed method in two practical systems: OriArm and OriFinger. The former implements single-handed input for wearables using hand gestures, whereas the latter interaction technique is based on finger input. Through these two technologies, wearable devices can continuously track the position of different body parts in 3D space. Given the continuous stream of 3D positions of different body parts of interest, the interaction designer can develop new interaction paradigm for wearables, similar to developing applications on traditional computers using a mouse.

Name	Hardware	Sensing Modalities	Form Factor	Tracking Positions	Algorithm	Algorithm Dependency	Discrete/ Continuous	Gestures
WatchOut	Commodity	Inertial Measurement Unit	Watch	Wrist	Machine Learning	User-indpendent	Discrete	14 gestures Tap/slide on watchcase and band
TapSkin	Commodity	Inertial Measurement Unit +Acoustic	Watch	Wrist	Machine Learning	User-dependent	Discrete	11 gestures NumberPad on the back of the hand
FingerSound	Customized	Inertial Measurement Unit +Acoustic	Ring	Thumb	Machine Learning + Pattern Recognition	User-dependent	Discrete	Performing 42 unistroke gestures by dragging thumb across fingers.
FingOrbits	Customized	Inertial Measurement Unit +Acoustic	Ring	Thumb	Machine Learning + Physics-based Modeling	User-dependent	Discrete	12 gestures Sliding thumb on fingers
FingerPing	Customized	Acoustic	Ring+Wrist-mounted device	Hand Poses	Machine Learning + Physics-based Modeling	User-dependent	Discrete	12 Phalange Taps + 10 American Sign Language Poses
SoundTrak	Customized	Acoustic	Ring + Watch	Fingers	Physics-based Modeling	User-independent	Continuous Tracking	Continuously tracking the position of the finger in 3D space
OriTrak	Customized	Orientation	Ring+Wristband+ headband+belt	Various body parts (Finger, Wrist, Head)	Physics-based Modeling	User-independent	Continuous Tracking	Continuously tracking the 3D angular position of different body parts

Figure 6.1: Summary of thesis works

In figure 6.1, I summarize the key characteristics of all these projects, including sensing modalities, algorithms, algorithm dependency, form factors, gesture design.

6.2 Methodology Summary

While working on these projects for the past five years, I collectively developed my own research methodology for technical HCI research on the topic of novel sensing and applied machine learning. The technical development process can be divided into the five parts: understanding the problems, interpreting the physical phenomenon, hardware prototyping, data processing, and algorithm design. In this section, I will describe my experience with each part.

6.2.1 Understanding the problems

The first step of any technical HCI research work should be understanding the research problems in-depth, including understanding the user needs, proposing solutions, and identifying the novelty and contribution. Unfortunately, this step is quite often skipped in practice, which can cause the significant problems in the later process of a research project. Here, I will describe the common practice I apply on my projects.

A technical HCI project is usually designed to improve the interaction experience between humans and computers. Therefore, the first step is always to understand the human needs. In other words, what is the problem that need to be addressed from the user's perspective? To understand the problem, there are two approaches. One is to find the previous research or works that have been done to understand the problems. These works usually extract insights from interviews with the target users and publish results in CSCW and CHI communities. Another approach I personally adopt more is to directly try out the technology and being the user. For instance, I have worn wearable devices in daily practice since 2013. I was in the first generation of Google Glass Explorer and purchased the first round-shape smartwatch (Moto 360). I wore Google Glass for at least one year in daily activities.

These experiences helped me define the problem space of my later thesis works. For instance, the first time I wore the Glass, I immediately realized I cannot easily input numbers and text on the Glass . In order to input, I have to use the smartphone app instead, which is extremely inconvenient. This is one important motivation for why I chose to work on wearable interactions as my thesis topic. In addition, I also realized other important issues that are beyond the scope of my thesis, such as privacy. One common issue I encountered everyday while wearing Glass is that people around me expressed concern about whether Glass is taking videos or pictures of them without their permission. This is a valid concern, because Glass by design does not provide any indication on the functionality or its working status to the surrounding people. These first-hand experiences are key to understanding the problem. I would suggest any researcher in the field should try out the technology first before working on it.

After identifying the problem, I think about different solutions. For each of the solutions, I am not only thinking about the technical feasibility. More importantly, I look at the literature to find any similar prior works to avoid simply replicating the previous research. This step is especially important for junior researchers, who may be relatively new to the field and thus has a limited understanding of the previous works. Checking the literature first before starting work on a problem can avoid a waste of effort. I have seen too many examples around me where a group of researchers found a published work that had already done what they implemented right before the paper submission, which wasted their months or even years of efforts. Therefore, identifying the novelty and contribution of the work by thoroughly examining the prior works is the most important step before starting working on a problem.

6.2.2 Understanding the physical phenomenon

Once the research topic or problem is confirmed and candidate solutions are proposed, the first step is to understand the physical phenomenon underneath the activity to be detected

or tracked. I usually start by thinking through what are the possible physical phenomena that can directly reflect the activity of interest. For instance, to track the movement of the thumb, the most obvious physical phenomenon is the motion of the thumb, which can be captured by motion tracking sensors (e.g., inertial measurement unit) as I demonstrated in FingerSound. Beyond the most straightforward answer, there are often other less obvious physical phenomena that can also exist. For instance, the movement of the thumb can change the air pressure around the hands or the signal strength around the smartwatch. If needed, I may try out different sensing modalities before making a decision. A thorough examination of a physical phenomenon at the beginning would help to guarantee of the success of a project.

The popular physical phenomenon that have been explored in technical HCI works are light, motion, vision, acoustics, physiological signals, electrical field, magnetic field, and electromagnetic field. Each physical phenomenon usually presents different pros and cons.

Motion sensing is most commonly seen on commodity devices, such as smartphone and smartwatch. But it suffers from a lot of motion noise from daily activities. To make motion data useful, cleaning data with appropriate signal processing techniques is usually required.

Vision sensing requires at least one camera to capture the data. The good side of vision sensing is it does not require any other instrumentation of the user or the environment other than the camera itself. But the downside is it requires a lot of processing and battery power to process the data, which is still a big challenge for wearables at this moment.

Acoustical sensing is widely used for a lot of applications on both wearables and traditional computers. Similarly to motion sensing, it is also widely available on various types of devices and requires a lot of data processing to clean up the environmental noise. In addition, acoustical data can be more user-dependent. For instance, tapping on the same location of different person would generate a different acoustical signature. Thus, to build a user-independent model requires a much larger training data set and a more compli-

cated machine learning model (e.g., Hidden-Markov Model), as demonstrated in the speech recognition community.

The electrical field, magnetic field, and electromagnetic field are ubiquitous in the environment. Our human bodies are good conductive mediums for these signals. Therefore, we have seen quite a few projects that explore using these physical phenomenon for interaction or for activity recognition. However, the challenge is repeatability of these signals, as these fields keep changing in the environment. For instance, even when the user is standing at the same location, the magnetic field can be different each time.

Regarding on-body sensing, quite a few physiological signals have been explored, such as Electrocardiography (EMG), Electroencephalograph (EEG), skin conductance, and bio-impedance. These sensing modalities have been used for various technical HCI projects, such as on-body interaction. However, all these sensing methods suffer from one limitation, that the received signal is not repeatable between different users or different sessions for the same user. In other words, the developed technology usually is user-dependent or session-dependent.

As I have summarized above, acoustical and motion sensing are more available and the received signals are more repeatable. Therefore, most projects in my thesis are based on these two types of sensing methods.

6.2.3 Hardware prototyping

After choosing the most appropriate physical phenomenon to track certain activities of interest, I pick the corresponding sensing modalities to convert these physical phenomenal into digital signals, which are later processed in the computers. Usually, for each type of sensor, there are at least hundreds of different models. These sensors are designed to capture the same physical phenomenon under different scenarios. I usually first narrow down the list by choosing the best-matched sets and then compare the key characteristics (e.g., sample rate, frequency response curve) on data-sheets to find the best one. For

instance, in FingerPing, in order to capture on-body sound, I chose contact microphones over the air-borne microphone for better sound quality. Usually, the advanced sensors are more customized, which also requires the connecting component to be customized. The most common customized component is the amplifier. For instance, in FingerPing, FingerSound, FingOrbits, and SoundTrak, the acoustical sensors have a very low noise-floor. But these systems required a customized amplifier to achieve the best performance.

Once all the hardware components are decided, I start assembling these components together. The most common skills I use are soldering (both soldering iron and surface-mount soldering) and wiring. The part I would like to highlight here is the wiring. Choosing the appropriate wires are important for a successful hardware prototype. In general, I would recommend using shielded wires that are softer and lighter if possible, because more rigid wires can stretch the sensors while in motion, which generates a lot of noise.

After the sensing and computing platforms are assembled together, I build different wearable form factors to hold this hardware. I usually print a case using a 3D printer, which is customized to the size of the hardware. Then I hot-glue the hardware on the case if needed and use different fabrics (e.g., Velcro) to attach the whole device comfortably and firmly on the user. For instance, in FingerSound, I glued the gyroscope and the contact microphone on the 3D printed thumb-ring. Velcro is attached to the ring to make it wearable for different users.

6.2.4 Data Processing

Once hardware prototyping is complete, I collect sensor data for the activities of interest and analyze the data off-line to find the best data processing algorithm. A common practice I use for data analysis is using a visualization method to examine different characteristics for visual patterns. For instance, I usually plot the raw sensor data and the spectrogram to examine the pattern in time and frequency domains. If I can find a visual pattern, I proceed to the next step, which is to clean up the data with state-of-the-art digital signal processing.

The purpose of applying signal processing on the data is to highlight the pattern and reduce noise. Low-pass and band-pass filters are the most common methods I have used on my projects. Depending on where the informative frequency is, I choose different types of filters. For instance, a low-pass filter is usually used when the data suffers from high-frequency noise, while the information is presented in the lower-frequency range. For example, in FingerSound, the gyroscope data was sampled over 4000 Hz, which means it can capture the signal up to 2000 Hz. However, most of our movement is below 50 Hz. Keeping the high-frequency noise can reduce the recognition performance for pattern recognition and machine learning. Therefore, I pass the data through a low-pass filter to highlight the temporal pattern under 50 Hz.

Besides the above filters, a Kalman filter is another tool I use for my projects on continuously calculating the positions. For instance, in SoundTrak, when I map the finger position in 3D to the cursor's position on the screen, I apply a Kalman filter, because the position is an estimated position which always has offset or uncertainty compared with the true position due to noise. In other words, the calculated position is not stable as showing on the screen. A Kalman filter improves the estimated locations by considering the previous locations and modeling the random noise. It can dramatically improve the stability of the calculated locations.

6.2.5 Algorithm design

Once the data have been cleaned and segmented (if needed), they are passed to the next stage for generating the final results using appropriate algorithms. The two types of algorithms I use are machine learning and physics-based modeling.

Many people tend to directly use machine learning without taking advantage of other more appropriate tools, such as building math formulas using physics if possible. Machine learning is very powerful. However, it also suffers from many shortcomings, especially for technical HCI projects, because ML requires a lot of training data to make the model

generalizable under different scenarios such as different users or devices. Unfortunately, the novel nature of technical HCI research (e.g., gesture recognition, activity recognition) makes it difficult to collect enough data for providing first proof-of-concept. Because we are mostly working on recognizing or tracking new activities or gestures, which have never been explored before, in many cases we do not have the prior data set that can be reused to form a large-enough data-set for ML algorithms. As a result, carefully choosing the appropriate method is one key factor in this step.

In practice, I always think physics-based modeling first. If the problem can be solved by solely using physics and math, I would not try machine learning, as shown in Sound-Trak. These physics-based modeling algorithms usually do not require any training data and can be generalized to different users and devices. In contrast, the machine-learning based algorithm usually requires a massive amount of data from users. However, to be able to apply physics-based modeling, the researcher has to develop a deep understanding of relevant physics phenomenon and working principles of the electronic components, as well as mastering some mathematical skills to solve physics problems.

Physics-based modeling is used when the physical phenomenon is clear and straightforward. However, when the problem is too complicated to be modeled by only using physics and math, machine learning is needed to model the complicated phenomenon. Even when I decide to use machine learning, I still tend to apply the insights from physics to the feature extraction process, instead of directly using feature engineering. For instance, in Finger-Ping, my previous work shows that lower-frequency sound propagates better on a human body. Therefore, when I extract frequency domain features, I limited the frequency band to the more informative lower-range without losing any information. I am not opposed to Deep Learning which is famous for feature engineering. It is one strategy of providing a convincing proof-of-concept given a very limited size of training data.

I also choose carefully between different ML algorithms. For instance, if there is helpful temporal information in the data, I tend to choose the algorithms that can keep the temporal

patterns, such as Hidden Markov Models or dynamic time-warping. The prior one is more advanced but requires a much larger set of data. The latter one can perform well even with a smaller data set, as I demonstrated in FingerSound. If the data does not present many temporal patterns, a support vector machine (SVM) is proven to the most popular ML algorithm for real-time classification tasks in practice, because it is less influenced by the small data sets. When a SVM builds models using training sets, only the data points on the edge of each class (called support vectors) matter. These support vectors set up the boundaries between different classes. Thus, the training data points other than the support vectors would have less influence. This feature helps me to run real-time classification with relatively small data-set.

6.3 Summary of Conclusion

In summary, my thesis has made a significant contribution to the field of wearable computing, especially in the area of wearable interaction. I invented a set of techniques that can be used on various types of wearables. Supported by my inventions, wearable interaction is no longer limited to on-device input, but is extended to the surrounding 3D space. The user does not need to look at the device while performing input. Instead, a comprehensive set of one-handed eyes-free input gestures are allowed.

6.4 Future Opportunities

In this section, I discuss future challenges and opportunities that can be extended upon my thesis work.

6.4.1 Understanding wearable interaction in the wild

Similar to much of the prior work in this field, the invented interaction techniques presented in this thesis have not been deployed and fully evaluated in the wild. This unfortunate reality was due to limited resources and time. However, this is one of the most significant

missing pieces in all of these novel interaction techniques. This work (including my own thesis work) was usually tested in a controlled lab environment, where the user simply performed pre-assigned tasks, rather than in a context where people will use these interactions for real applications. For instance, FingerSound allows the users to perform one-handed text input. It is designed to input on different wearable devices, especially for heads-up displays (VR headset). But the user study was conducted in a controlled-lab environment with the display on a computer screen instead of a real heads-up display. It is usually acceptable in academia to present such proof-of-concept in a controlled user study. However, to move these techniques towards the goal of being deployed in real applications and devices, they have to be thoroughly tested and understood within the corresponding context in the wild.

Evaluating the system in the wild would introduce more technical challenges, especially for wearables. Most wearables are always on the user, which means they need to be functional in different dynamic contexts. For instance, a user may want to respond to a message on the head-mounted display (e.g., Google Glass) while he is walking on campus. The first challenge is to effectively detect the input intention of a user and avoid false positive and false negative errors. For gesture-based systems, an activation gesture is highly recommended. But choosing the appropriate activation gesture would demand another study which can last for weeks or months. The second challenge is to understand how would these input technique work while the users are in dynamic scenarios (in motion), and to improve the system according to the study results. The third challenge is to understand the social appropriateness of these input gestures. During the current user studies, the participants were recruited to operate these input techniques in the confines of a lab. However, some input techniques may be socially inappropriate when performing in the public. Evaluating the social appropriateness of these input technique is another topic worth exploring in the future.

6.4.2 Affordance of wearable interaction

Compared to touch-based input, gesture-based input usually suffers from the lack of affordance. On one hand, the design of the gesture is not always self-explanatory. The user has to learn the design of gestures before they are able to effectively operate gestures in applications. A well-recognized guideline of designing gestures with appropriate affordances should be investigated.

On the other hand, there is often a lack of tactile feedback when performing the gestures, especially in 3D space. For instance, SoundTrak can track the position of the finger in 3D space. However, one thing we learned from the user study was that people have much worse spatial perception in 3D space as compared to 2D space. One possible explanation is moving hands or fingers in 3D space offers a lot of freedom but lacks sufficient tactile feedback. This results in relatively poor control of fingers or hands in 3D space, especially in eyes-free wearable interaction.

6.4.3 Towards user independent ML models

Many of these machine learning-based input technique adopt a user-dependent model. It requires the user to provide training data before they can actually use it. This is another obstacle we need to address before deploying these techniques in consumer products. Fortunately, many of these techniques are not fundamentally user-dependent, but rather are limited by the small size of training data. In the future, we as a community should try to collectively contribute training data for certain input techniques of high interest, in order to explore the possibility of building user-independent models. Once we have a much larger data-sets, more advanced machine learning techniques (e.g., Hidden Markov Model, Deep learning) can be applied, which may further improve the performance. This road map has been proven successful in other areas. For instance, in speech recognition, given the existence of millions of training data, the user does not have to provide training data before using the service. I would argue a similar approach can be transferred to technical HCI

community on certain recognition techniques of high interest to the community. Such an effort on collecting a huge amount of data was beyond the scope of my research thesis,but it is feasible if we can collectively build up our own standard training data sets within the community.

6.4.4 Power-harvesting on wearables

The small size of wearable devices not only influence the interaction experience, but also limit the size of the built-in battery. The small battery size results in a small battery capacity. As a result, the wearable devices cannot afford power-hungry hardware and applications. For instance, Google Glass can only record video for 20 minutes before the battery dies. The applications on wearables are limited because of the battery, which influences the overall user experience.

To address the power issue, one straightforward solution is to wait for revolutionary battery technology to appear from materials science, which can provide significantly much more power than the current ones. But this is out of the scope of this work. As Ubiquitous Computing researchers, we can make efforts to address this challenge from different angles. We can harvest energy from the signals (e.g., RF signals, Acoustic Signals) in the environment to power the wearables. We can also be smart on when to activate the wearable devices. However, one of my favorite solution candidates for the power issue is to transmit energy from the computing devices with more battery (e.g., smartphone) to the ones that are more sensitive to the size. For instance, in SoundTrak, we use devices in two form factors: a smartwatch and a ring. The only function the ring has is to generate acoustical signals at a constant frequency using a miniature speaker, which demands some power. Constantly charging a battery on a ring can be challenging and inconvenient. A possible solution is to replace the battery on the ring with an ultrasonic energy harvester and generate ultrasonic signals on the watch. As such, the ring can harvest energy from the watch wirelessly using the air-borne ultrasonic signals when needed.

6.4.5 AR/VR interaction

AR/VR currently is getting a lot of attention from the industry. However, the cumbersome hardware set is still a burden to the user experience. The user can not bear wearing the whole set of hardware in daily life or even for a couple hours. Simplifying the hardware set is another interesting challenge for the future wearable interaction. It would be very difficult to make the headset or immersive displays smaller and more comfortable, but it is relatively easier to improve the input hardware. Most VR sets on the market require the user to hold a hand controller for input. A possible solution in the future is to replace the controller with a much smaller wearable device, such as a ring or a wristband. The user could then input in a VR environment purely by performing gestures without occupying the hands.

6.4.6 Activity recognition

One inevitable part of future wearable interaction is to continuously track and detect the input intention of a user. In most cases, the system needs to track the position of different body parts, such as finger, hand, head, as we demonstrated in the OriTrak project. The position of different body parts can not only be used for interaction, but also for activity recognition. For instance, knowing the 3D position of the hand and the head in relation to the body, can potentially provide information to facilitate recognition of fine-grained activities such as, eating, drinking, or using smartphones, which are currently hard to distinguish. In the future, naturally understanding the human body language by tracking the full body postures can be used for seamlessly interpreting human intents for various applications such as interaction, health sensing, and activity recognition.

6.5 Final Thoughts

My thesis work had demonstrated advancements towards the future wearable of interaction, where wearable users can input information more effectively and naturally. However, improvements to interaction will not be enough to deliver the future of wearables. More importantly, wearable computers need to have killer applications that traditional computers cannot provide. Activity logging and health sensing can be such killer applications. The techniques demonstrated in this thesis can potentially be transferred to develop these applications. Once wearables are able to provide killer applications and allow the user to effectively and naturally exchange information, wearables will be standing at the center of the computing world.

Appendices

REFERENCES

[1] G. D. Abowd, "Beyond weiser: From ubiquitous to collective computing," Computer, vol. 49, no. 1, pp. 17–23, 2016.

[2] C. Harrison, D. Tan, and D. Morris, "Skinput: Appropriating the body as an input surface," in Proc. CHI, ACM, 2010, pp. 453–462.

[3] A. Mujibiya, X. Cao, D. S. Tan, D. Morris, S. N. Patel, and J. Rekimoto, "The sound of touch: On-body touch and gesture sensing based on transdermal ultrasound propagation," in Proc. ITS, ser. ITS '13, St. Andrews, Scotland, United Kingdom: ACM, 2013, pp. 189–198, ISBN: 978-1-4503-2271-3.

[4] T. S. Saponas, D. S. Tan, D. Morris, R. Balakrishnan, J. Turner, and J. A. Landay, "Enabling always-available input with muscle-computer interfaces," in Proc. UIST, ser. UIST '09, Victoria, BC, Canada: ACM, 2009, pp. 167–176, ISBN: 978-1-60558-745-5.

[5] A. Dementyev and J. A. Paradiso, "Wristflex: Low-power gesture input with wrist-worn pressure sensors," in Proc. UIST, Association for Computing Machinery (ACM), 2014.

[6] C. Zhang, A. Bedri, G. Reyes, B. Bercik, O. T. Inan, T. E. Starner, and G. D. Abowd, "Tapskin: Recognizing on-skin input for smartwatches," in Proc. ISS, ser. ISS '16, Niagara Falls, Ontario, Canada: ACM, 2016, pp. 13–22, ISBN: 978-1-4503-4248-3.

[7] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "Fingerio: Using active sonar for fine-grained finger tracking," in Proc. CHI, ser. CHI '16, Santa Clara, California, USA: ACM, 2016, pp. 1515–1525, ISBN: 978-1-4503-3362-7.

[8] Y. Zhang, J. Zhou, G. Laput, and C. Harrison, "Skintrack: Using the body as an electrical waveguide for continuous finger tracking on the skin," in Proc. CHI, ser. CHI '16, Santa Clara, California, USA: ACM, 2016, pp. 1491–1503, ISBN: 978-1-4503-3362-7.

[9] G. Laput, C. Yang, R. Xiao, A. Sample, and C. Harrison, "Em-sense: Touch recognition of uninstrumented, electrical and electromechanical objects," in Proc. UIST, ser. UIST '15, Daegu, Kyungpook, Republic of Korea: ACM, 2015, pp. 157–166, ISBN: 978-1-4503-3779-3.

[10] G. Cohn, S. Gupta, T.-J. Lee, D. Morris, J. R. Smith, M. S. Reynolds, D. S. Tan, and S. N. Patel, "An ultra-low-power human body motion sensor using static electric field sensing," in Proc. Ubicomp, ACM, 2012, pp. 99–102.

[11] J. Gong, X.-D. Yang, and P. Irani, "Wristwhirl: One-handed continuous smartwatch input using wrist gestures," in Proc. UIST, ser. UIST '16, Tokyo, Japan: ACM, 2016, pp. 861–872, ISBN: 978-1-4503-4189-9.

[12] C. Loclair, S. Gustafson, and P. Baudisch, "Pinchwatch: A wearable device for one-handed microinteractions," in Proc. MobileHCI, 2010.

[13] G. Laput, R. Xiao, and C. Harrison, "Viband: High-fidelity bio-acoustic sensing using commodity smartwatch accelerometers," in Proc. UIST, ser. UIST '16, Tokyo, Japan: ACM, 2016, pp. 321–333, ISBN: 978-1-4503-4189-9.

[14] K. Lyons, T. Starner, D. Plaisted, J. Fusia, A. Lyons, A. Drew, and E. W. Looney, "Twiddler typing: One-handed chording text entry for mobile phones," in Pro. CHI, Association for Computing Machinery (ACM), 2004.

[15] C. Amma, M. Georgi, and T. Schultz, "Airwriting: Hands-free mobile text input by spotting and continuous recognition of 3d-space handwriting with inertial sensors," in Proc. ISWC, IEEE, 2012, pp. 52–59.

[16] D. Kim, O. Hilliges, S. Izadi, A. D. Butler, J. Chen, I. Oikonomidis, and P. Olivier, "Digits: Freehand 3d interactions anywhere using a wrist-worn gloveless sensor," in Proc. UIST, 2012.

[17] L. Chan, Y.-L. Chen, C.-H. Hsieh, R.-H. Liang, and B.-Y. Chen, "Cyclopsring: Enabling whole-hand and context-aware interactions through a fisheye ring," in Proc. UIST, 2015.

[18] H.-R. Tsai, C.-Y. Wu, L.-T. Huang, and Y.-P. Hung, "Thumbring: Private interactions using one-handed thumb motion input on finger segments," in Adjunct Proc. MobileHCI, 2016.

[19] D.-Y. Huang, L. Chan, S. Yang, F. Wang, R.-H. Liang, D.-N. Yang, Y.-P. Hung, and B.-Y. Chen, "Digitspace: Designing thumb-to-fingers touch interfaces for one-handed and eyes-free interactions," in Proc. CHI, 2016.

[20] K.-Y. Chen, K. Lyons, S. White, and S. Patel, "Utrack: 3d input using two magnetic sensors," in Proc. UIST, 2013.

[21] W. Kienzle and K. Hinckley, "Lightring: Always-available 2d input on any surface," in Proc. UIST, ser. UIST '14, Honolulu, Hawaii, USA: ACM, 2014, pp. 157–160, ISBN: 978-1-4503-3069-5.

[22] S. H. Yoon, Y. Zhang, K. Huo, and K. Ramani, "Tring: Instant and customizable interactions with objects using an embedded magnet and a finger-worn device," in Proc. UIST, ser. UIST '16, Tokyo, Japan: ACM, 2016, pp. 169–181, ISBN: 978-1-4503-4189-9.

[23] H.-R. Tsai, M.-C. Hsiu, J.-C. Hsiao, L.-T. Huang, M. Chen, and Y.-P. Hung, "Touchring: Subtle and always-available input using a multi-touch ring," in Adjunct Proc. MobileHCI, ser. MobileHCI '16, Florence, Italy: ACM, 2016, pp. 891–898, ISBN: 978-1-4503-4413-5.

[24] K. Lyons, D. Nguyen, D. Ashbrook, and S. White, "Facet: A multi-segment wrist worn system," in Proc. CHI, Association for Computing Machinery (ACM), 2012.

[25] H. Xia, T. Grossman, and G. Fitzmaurice, "Nanostylus: Enhancing input on ultra-small displays with a finger-mounted stylus," in Proc. UIST, Association for Computing Machinery (ACM), 2015.

[26] I. Oakley, D. Lee, M. R. Islam, and A. Esteves, "Beats: Tapping gestures for smart watches," in Proc. CHI, Association for Computing Machinery (ACM), 2015.

[27] R. Xiao, G. Laput, and C. Harrison, "Expanding the input expressivity of smartwatches with mechanical pan, twist, tilt and click," in Proc. CHI, Association for Computing Machinery (ACM), 2014.

[28] D. Ashbrook, K. Lyons, and T. Starner, "An investigation into round touchscreen wristwatch interaction," in Proc. MobileHCI, Association for Computing Machinery (ACM), 2008.

[29] G. Blasko and S. Feiner, "An interaction system for watch computers using tactile guidance and bidirectional segmented strokes," in ISWC, Institute of Electrical & Electronics Engineers (IEEE).

[30] S. T. Perrault, E. Lecolinet, J. Eagan, and Y. Guiard, "Watchit: Simple gestures and eyes-free interaction for wristwatches and bracelets," in Proc. CHI, Association for Computing Machinery (ACM), 2013.

[31] M. Funk, A. Sahami, N. Henze, and A. Schmidt, "Using a touch-sensitive wristband for text entry on smart watches," in Extended Abstract on Proc.CHI, Association for Computing Machinery (ACM), 2014.

[32] F. Kerber, P. Lessel, and A. Krüger, "Same-side hand interactions with arm-placed devices using emg," in Extended Abstracts on Proc.CHI, ACM, 2015, pp. 1367–1372.

[33] J. Kim, J. He, K. Lyons, and T. Starner, "The gesture watch: A wireless contact-free gesture based wrist interface," in Proc. ISWC, Institute of Electrical & Electronics Engineers (IEEE), 2007.

[34] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "Fingerio: Using active sonar for fine-grained finger tracking," in Proc. CHI, ACM, 2016, pp. 1515–1525.

[35] G. Laput, R. Xiao, X. . Chen, S. E. Hudson, and C. Harrison, "Skin buttons: Cheap, small, low-powered and clickable fixed-icon laser projectors," in Proc. UIST, Association for Computing Machinery (ACM), 2014.

[36] M. Ogata and M. Imai, "Skinwatch: Skin gesture interaction for smart watch," in Proc. AH, Association for Computing Machinery (ACM), 2015.

[37] Y. Zhang, J. Zhou, G. Laput, and C. Harrison, "Skintrack: Using the body as an electrical waveguide for continuous finger tracking on the skin," in Proc. CHI, ACM, 2016, pp. 1491–1503.

[38] D. Ashbrook, P. Baudisch, and S. White, "Nenya: Subtle and eyes-free mobile input with a magnetically-tracked finger ring," in Proc. CHI, ser. CHI '11, Vancouver, BC, Canada: ACM, 2011, pp. 2043–2046, ISBN: 978-1-4503-0228-9.

[39] F. Kerber, A. Krüger, and M. Löchtefeld, "Investigating the effectiveness of peephole interaction for smartwatches in a map navigation task," in Proc. MobileHCI, ACM, 2014, pp. 291–294.

[40] Y. Bernaerts, M. Druwé, S. Steensels, J. Vermeulen, and J. Schöning, "The office smartwatch: Development and design of a smartwatch app to digitally augment interactions in an office environment," in Proc. DIS, ACM, 2014, pp. 41–44.

[41] C. Xu, P. H. Pathak, and P. Mohapatra, "Finger-writing with smartwatch: A case for finger and hand gesture recognition using smartwatch," in Proc. MobileHCI, ACM, 2015, pp. 9–14.

[42] H. Wen, J. Ramos Rojas, and A. K. Dey, "Serendipity: Finger gesture recognition using an off-the-shelf smartwatch," in Proc. CHI, ACM, 2016, pp. 3847–3851.

[43] W. McGrath and Y. Li, "Detecting tapping motion on the side of mobile devices by probabilistically combining hand postures," in Proc. UIST, ACM, 2014, pp. 215–219.

[44] C. Zhang, A. Guo, D. Zhang, C. Southern, R. Arriaga, and G. Abowd, "Beyondtouch: Extending the input language with built-in sensors on commodity smartphones," in Proc. IUI, Association for Computing Machinery (ACM), 2015.

[45] C. Zhang, A. Parnami, C. Southern, E. Thomaz, G. Reyes, R. Arriaga, and G. D. Abowd, "Backtap: Robust four-point tapping on the back of an off-the-shelf smartphone," in Adjunct Proc. UIST, ser. UIST '13 Adjunct, St. Andrews, Scotland, United Kingdom: ACM, 2013, pp. 111–112, ISBN: 978-1-4503-2406-9.

[46] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in Proc. MobiCom, ACM, 2000, pp. 32–43.

[47] W. Huang, Y. Xiong, X.-Y. Li, H. Lin, X. Mao, P. Yang, and Y. Liu, "Shake and walk: Acoustic direction finding and fine-grained indoor localization using smartphones," in IEEE INFOCOM, IEEE, 2014, pp. 370–378.

[48] Y. Ma, X. Hui, and E. C. Kan, "3d real-time indoor localization via broadband nonlinear backscatter in passive devices with centimeter precision," in Proc. MobiCom, ACM, 2016, pp. 216–229.

[49] P. V. Nikitin, R. Martinez, S. Ramamurthy, H. Leland, G. Spiess, and K. Rao, "Phase based spatial identification of uhf rfid tags," in IEEE RFID Virtual Journal, IEEE, 2010, pp. 102–109.

[50] C. Hekimian-Williams, B. Grant, X. Liu, Z. Zhang, and P. Kumar, "Accurate localization of rfid tags using phase difference," in IEEE RFID Virtual Journal, IEEE, 2010, pp. 89–96.

[51] A. Brenner and P De Bruyne, "A sonic pen: A digital stylus system," IEEE Tran. Computers, vol. 19, no. 6, pp. 546–548, 1970.

[52] M. J. Stefik and J. C. Heater, Ultrasound position input device, US Patent 4,814,552, 1989.

[53] EPOS, Epos ultrasonic pen, [Online; accessed 11-February-2017], 2007.

[54] W. Mao, J. He, and L. Qiu, "Cat: High-precision acoustic motion tracking," in Proc. MobiCom, ACM, 2016, pp. 69–81.

[55] C. Microsystems, Chirp, [Online; accessed 11-February-2017], 2013.

[56] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in Proc. MobiCom, ACM, 2016, pp. 82–94.

[57] K.-Y. Chen, K. Lyons, S. White, and S. Patel, "Utrack: 3d input using two magnetic sensors," in Proc. UIST, ser. UIST '13, St. Andrews, Scotland, United Kingdom: ACM, 2013, pp. 237–244, ISBN: 978-1-4503-2268-3.

[58] C. Holz and M. Knaust, "Biometric touch sensing: Seamlessly augmenting each touch with continuous authentication," in Proc. UIST, ser. UIST '15, Daegu, Kyungpook, Republic of Korea: ACM, 2015, pp. 303–312, ISBN: 978-1-4503-3779-3.

[59] T. G. Zimmerman, "Networks : near-field intrabody," vol. 35, pp. 609–617, 1996.

[60] E. R. Post, M. Reynolds, M. Gray, J. Paradiso, and N. Gershenfeld, "Intrabody buses for data and power," in Proc. ISWC, IEEE, 1997, p. 52.

[61] M Fukumoto and Y Tonomura, "body coupled fingerringi: wireless wearable keyboard," Proceedings of CHI 1997, pp. 147–154, 1997.

[62] J. Bae, S. Member, H. Cho, and K. Song, "The signal transmission mechanism on the surface of human body for body channel communication," Microwave Theory and Tech, vol. 60, no. 3, pp. 582–593, 2012.

[63] N. Cho, J. Yoo, S. J. Song, J. Lee, S. Jeon, and H. J. Yoo, "The human body characteristics as a signal transmission medium for intrabody communication," Microwave Theory and Tech, vol. 55, no. 5, pp. 1080–1085, 2007.

[64] J. Yoo, N. Cho, and H.-J. Yoo, "Analysis of body sensor network using human body as the channel," Proc. BodyNets, 2008.

[65] K Fujii, M Takahashi, and K Ito, "Electric field distributions of wearable devices using the human body as a transmission channel," Trans. Antennas and Propagation, vol. 55, no. 7, pp. 2080–2087, 2007.

[66] K. Hachisuka and A. Nakata, "Development and performance analysis of an intrabody communication device," Transducers, pp. 1722–1725, 2003.

[67] J. Park and P. P. Mercier, "Magnetic human body communication," Proc. EMBS, vol. 2015-Novem, pp. 1841–1844, 2015.

[68] S. Franklin and S. Rajan, "Personal area network for biomedical monitoring systems using human body as a transmission medium.," Journal of Bio-Science, vol. 2, no. 2, pp. 23–28, 2010.

[69] H. J. Yoo and N. Cho, "Body channel communication for low energy bsn/ban," Proc. APCCAS, pp. 7–11, 2008.

[70] S. Stenfelt and R. L. Goode, "Transmission properties of bone conducted sound: Measurements in cadaver heads," The Journal of the Acoustical Society of America, vol. 118, no. 4, pp. 2373–2391, 2005.

[71] I. Hwang, J. Cho, and S. Oh, "Privacy-aware communication for smartphones using vibration," in Proc.RTCSA, IEEE, 2012, pp. 447–452.

[72] N. Roy, M. Gowda, and R. R. Choudhury, "Ripple: Communicating through physical vibration," in NSDI 15, 2015, pp. 265–278.

[73] T. Deyle, S. Palinko, E. S. Poole, and T. Starner, "Hambone: a bio-acoustic gesture interface," Proc. ISWC, pp. 1–8, 2007.

[74] M. Fukumoto and Y. Tonomura, "Whisper: a wristwatch style wearable handset," Proc. CHI, pp. 112–119, 1999.

[75] L. Zhong, D. El-Daye, B. Kaufman, N. Tobaoda, T. Mohamed, and M. Liebschner, "Osteoconduct: Wireless body-area communication based on bone conduction," in Proc. BodyNets, ser. BodyNets '07, Florence, Italy: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007, 9:1–9:8, ISBN: 978-963-06-2193-9.

[76] C. Harrison, H. Benko, and A. D. Wilson, "Omnitouch: Wearable multitouch interaction everywhere," in Proc. UIST, ACM, 2011, pp. 441–450.

[77] M. Weigel, V. Mehta, and J. Steimle, "More than touch: Understanding how people use skin as an input surface for mobile computing," in Proc. CHI, ser. CHI '14, Toronto, Ontario, Canada: ACM, 2014, pp. 179–188, ISBN: 978-1-4503-2473-1.

[78] J. Rekimoto, "GestureWrist and GesturePad: Unobtrusive wearable interaction devices," in Proceedings Fifth International Symposium on Wearable Computers, Proc. ISWC.

[79] T. S. Saponas, D. S. Tan, D. Morris, R. Balakrishnan, J. Turner, and J. A. Landay, "Enabling always-available input with muscle-computer interfaces," in Proc. UIST, ACM, 2009, pp. 167–176.

[80] T. Deyle, S. Palinko, E. S. Poole, and T. Starner, "Hambone: A bio-acoustic gesture interface," in Proc. ISWC, Institute of Electrical & Electronics Engineers (IEEE), 2007.

[81] D. Merrill and H. Raffle, "The sound of touch," in Extended Abstracts on Proc. CHI, ACM, 2007, pp. 2807–2812.

[82] C. Zhang, S. Hersek, Y. Pu, D. Sun, Q. Xue, T. E. Starner, G. D. Abowd, and O. T. Inan, "Bioacoustics-based human-body-mediated communication," Computer, vol. 50, no. 2, pp. 36–46, Feb. 2017.

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

[84] K. Lyons, D. H. Nguyen, S. Seko, S. White, D. Ashbrook, and H. Profita, "Bitwear: a platform for small, connected, interactive devices," Adjunct on Proc. UIST, pp. 73–74, 2013.

[85] I. S. MacKenzie and S. X. Zhang, "The immediate usability of graffiti," in GI, vol. 97, 1997, pp. 129–137.

[86] H. Istance, R. Bates, A. Hyrskykari, and S. Vickers, "Snap clutch, a moded approach to solving the midas touch problem," in Proc. ETRA, ser. ETRA '08, Savannah, Georgia: ACM, 2008, pp. 221–228, ISBN: 978-1-59593-982-1.

[87] G. A. ten Holt, M. J. Reinders, and E. Hendriks, "Multi-dimensional dynamic time warping for gesture recognition," in Proc. Advanced School for Computing and Imaging, vol. 300, 2007.

[88] A. Esteves, E. Velloso, A. Bulling, and H. Gellersen, "Orbits: Gaze interaction for smart watches using smooth pursuit eye movements," in Proc. UIST, 2015.

[89] Z. Cheng, X. Qiuyue, W. Anandghan, M. Ruichen, J. Sumeet, H. Yizeng, L. Xinyu, C. Kenneth, P. Thomas, S. Thad, I. Omer, and A. Gregory D., "Fingerping: Recognizing fine-grained hand poses using active acoustic on-body sensing," in Proc. CHI, ser. CHI '18, Montreal, Canada, 2018, ISBN: 978-1-4503-3574-4.

[90] T. Rahman, A. T. Adams, M. Zhang, E. Cherry, B. Zhou, H. Peng, and T. Choudhury, "Bodybeat: A mobile system for sensing non-speech body sounds," in Proc. MobiCom, ser. MobiSys '14, Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 2–13, ISBN: 978-1-4503-2793-0.

[91] T. Giannakopoulos and A. Pikrakis, Introduction to Audio Analysis: A MATLAB Approach, 1st. Academic Press, 2014, ISBN: 0080993885, 9780080993881.

[92] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, Data Mining: Practical machine learning tools. Morgan Kaufmann, 2016.

[93] M. C. Junger and D. Feit, Sound, structures, and their interaction. MIT press Cambridge, MA, 1986, vol. 225.

[94] Wikipedia, Inverse square law, [Online; accessed 11-February-2017], 2017.

[95] O. D. Lara and M. A. Labrador, "A survey on human activity recognition using wearable sensors," IEEE Communications Surveys & Tutorials, vol. 15, no. 3, pp. 1192–1209, Jul. 2013.

[96] E. Thomaz, I. Essa, and G. D. Abowd, "A practical approach for recognizing eating moments with wrist-mounted inertial sensing," in Proc. Ubicomp, ser. UbiComp '15, Osaka, Japan: ACM, 2015, pp. 1029–1040, ISBN: 978-1-4503-3574-4.

[97] M. Weiser, "The computer for the 21st century," Scientific American, vol. 265, no. 3, pp. 94–104, 1991.

[98] G. D. Abowd, "What next , ubicomp ? celebrating an intellectual disappearing act," Proc. Ubicomp, pp. 31–40, 2012.