Copyright
by
Sangki Yun
2016

The Dissertation Committee for Sangki Yun
certifies that this is the approved version of the following dissertation:

Towards Accurate Object Tracking using Acoustic Signal

Committee:

Lili Qiu, Supervisor

Aloysius Mok

Mohamed G. Gouda

Gustavo de Veciana

Towards Accurate Object Tracking using Acoustic Signal

by

Sangki Yun, B.E., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2016

Towards Accurate Object Tracking using Acoustic Signal

Publication No. _____

Sangki Yun, Ph.D.
The University of Texas at Austin, 2016

Supervisor: Lili Qiu

We are living in the era of mobile computing where people are surrounded by many smart devices such as smartphone, smart watch, smart TV, and Virtual Reality (VR) headsets. For them, providing intuitive user interface is crucial to satisfy the needs of the users, but their limited form-factor makes them difficult to support natural user interface. Object tracking can provide new opportunity to design intuitive gesture based user interface. First, it can provide more convenient User interface (UI) than traditional controllers and can be used to control a wide variety of Internet of Things devices. For example, it is difficult to use button based controller in VR headsets because users may not see the controller. By tracking the position of the controller or hand, it can control VR applications more intuitively. Second, it can be used to support motion based gaming, which is getting increasingly popular.

In this dissertation, we provide accurate object tracking methods that are useful to design intuitive user interface for mobile systems. In particular, we focus on exploiting the acoustic signal to track the movement of the object. While the vision based and the RF signal based object tracking have been extensively investigated, acoustic signal based tracking has not been under explored. The advantage of the acoustic signal based tracking is that it can be enabled by widely available speakers and microphones and can be processed in software without any extra hardware. Using the acoustic signal, we provide two different ways of tracking: 1) tracking mobile devices such as smartphone and smart watch, and 2) device-free tracking that tracks a hand without wearing a device that exploits the reflected signal from the moving hand. In both scenarios, we provide sufficient tracking accuracy so that the mobile or the hand is used as a mouse in the air.

First, we develop a system that can accurately track the movement of a mobile device using the acoustic signal. The device to be controlled (*e.g.*, smart TVs) sends the acoustic signal using its speaker, and the mobile device tracks the movement. More specifically, the tracker sends inaudible sound pulses at a few selected frequencies, and uses the frequency shifts to estimate the speed and distance traveled. we then develop techniques to quickly calibrate the distance between speakers and narrow down the devices initial position using its movement trajectory. Based on the information, we continuously track the devices new position in real time. This is feasible because many devices, such as smart TVs, PCs, and laptops, already have multiple speakers. Our

evaluation and user study demonstrate that our system achieves high tracking accuracy (e.g., median error of around 1.4 cm) and ease of use.

Next, we provide a device-free motion tracking system. It tracks the movement of the hand relying the reflected acoustic signal, which is more challenging. To realize it, we propose a novel approach that can estimate the distance and velocity using a single chirp signal and combine both information to accurately locate the moving hand. Through micro-benchmarks and user studies, we show our system achieves the medium tracking error of 1.94 cm using 2 speakers and 2 microphones on the same computer.

Finally, we improve the accuracy of the device-free tracking so that it is applicable for gesture based user interface for smart watches and VR headsets. We design single carrier based data communication system over acoustic channel and observe the channel impulse responses of the reflected signal. By observing the phase change, we can track the movement of the finger accurately. In our experiments using the speaker and the microphones of the commercial mobile device, we show that the finger movement can be tracked very accurately with the tracking error less than 1 centimeter.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

With the proliferation of mobile and smart devices such as smart phone, smart appliances, tablets and smart TVs, we are living in the era of ubiquitous and pervasive computing where devices ceaselessly interact with each other and human. In mobile and ubiquitous computing, object tracking has been considered as an important issue. Using the location and movement information of the object, we can recognize the context of the user and provide a service accordingly. For example, having the location of the user in a shopping mall, we can provide an information specific to the location (*e.g.*, advertisement or coupon) to the user without user intervention. Also, by accurately tracking the movement of mobile devices or user body parts, we can enable gesture and motion recognition based user interface (UI). Due to the enormous amount of applications, object tracking has received significant research attention over last few decades. However, the existing solutions still do not satisfy the needs of the users and the application. Moreover, fine grained object tracking that is useful for motion based UI design is more challenging, because it requires very accurate tracking to meet the needs of the applications. The existing solutions use extra hardware (*e.g.*, camera or antenna array) to achieve it, or provide

insufficient accuracy.

Our goal is enabling accurate object tracking only relying on existing hardware in smart devices. To achieve it, this dissertation explores the acoustic signal for the purpose of tracking. A unique advantage of it is that the speakers and microphones available in smart device can be used to transmit and receive the acoustic signal. Using the acoustic signals transmitted in the inaudible frequency range, (*i.e.*, 16 KHz to 24 KHz), we continuously track the movement of the object in real-time. This dissertation demonstrates that the acoustic signal based object tracking can provide high tracking accuracy in both 1) mobile device tracking that tracks the position of the mobile device with microphone while the speakers in fixed location are emitting the acoustic signal, and 2) device-free tracking that tracks the location of the moving object while the speakers and the microphones are fixed.

1.2 Mobile Device Tracking

A mouse has been one of the most successful technologies for controlling the graphic user interface due to its ease of use. Its attraction will soon penetrate well beyond just computers. There already have been mice designed for game consoles and smart TVs. A smart TV allows a user to run popular computer programs and smartphone applications. For example, a smart TV user may want to use a Web browser and click on a certain URL or some part of a map using a mouse. A traditional remote controller, which uses buttons for user input, is no longer sufficient to exploit full functionalities offered by

the smart TV. On the other hand, a traditional mouse, which requires a flat and smooth surface to operate, cannot satisfy many new usage scenarios. A user may want to interact with the remote device while on the move. For example, a speaker wants to freely move around and click on different objects in his slide; a smart TV user wants to watch TV in any part of a room; a Google Glass user wants to query about objects while he is touring around. Wouldn't it be nice if a user can simply turn his smartphone or smart watch into a mouse by moving it in the air?

However, there are many challenges to achieve these goals. First of all, in order to enable the air mouse capability using the device, the device movement should be tracked very accurately, within a few centimeters. Existing indoor localization that provides meter-level accuracy cannot achieve this goal. Many smart TV and set-top box manufacturers provide advanced remote controllers [56]. Some of them even provides motion control and gesture recognition using inertial sensors, such as accelerometer and gyroscope [51, 52]. Existing accelerometer is well known for their significant measurement errors, and cannot provide accurate tracking. We further confirm this using our measurement studies. Gyroscopes achieve better accuracy in tracking rotation. However, a user has to learn to how to rotate in order to control the displacement in a 2-D space. This is not intuitive, and is especially hard for moving in a diagonal direction, thereby degrading user experience and speed of control. Accelerometer based approach also fails due to significant noise in the measured acceleration.

Figure 1.1: Acoustic signal based device tracking.

We propose Accurate Air Mouse (AAMouse) that accurately tracks device movement in real time. It enables any mobile device with a microphone, such as a smart phone and smart watch, to serve as a mouse to control an electronic device with speakers. A unique feature of our approach is that it uses existing hardware in mobile and electronic devices. Figure 4.2 shows an example of our system, where a mobile device with a microphone serves as a mouse for a smart TV with two speakers. The smart TV emits inaudible acoustic signals, and the mobile device records and feeds it back to the smart TV, which estimates the device position based on the Doppler shift.

While there are some existing work that leverages the Doppler shift for gesture recognition, tracking is more challenging since gesture recognition only requires matching against one of the training patterns whereas tracking requires accurate positioning of the device. This not only requires accurate estimation of frequency shift, but also translating the frequency shift into a position involves significant additional research issues, such as how to estimate the distance between the speakers, the device's initial device position, and its new position based on the frequency shift.

We first estimate frequency shift and use it to position the device assuming that the distance between the speakers and the device's initial position are both known. Then we develop techniques to quickly calibrate the distance between the speakers using the Doppler shift. To address the device's unknown initial position, we employ particle filter, which generates many particles corresponding to the device's possible positions and filters the particles whose locations are inconsistent with the measured frequency shifts. The device's position is estimated as the centroid of the remaining particles. To further enhance robustness, we transmit signals at multiple frequencies, perform outlier removal, and combine the remaining estimations.

1.3 Device-Free Tracking

Device-free motion tracking enables a new way for users to interact with the world by simply moving their hands. Recently, Virtual Reality (VR) and Augmented Reality (AR) are all taking off, where the availability of easy-to-use user interface is the key to their success. VR and AR are expected to hit $150 billion by 2020. They provide immersed experience, and open the doors to new ways of training, education, meeting, advertising, travel, health care, emergency responses, and scientific experiments. However, the current user interfaces of VR/AR are rather limited: they rely on tapping, swiping, voice recognition, or steering the camera towards the hand to make sure the hand is within the view and line-of-sight of the camera while wearing the headset. Our vision is to develop a device-free user interface (UI) so that a user can freely

move his or her hand to control game console, VR, AR, and smart appliances.

Motivated by this vision, we develop novel device-free tracking systems to accurately track a hands movement. In our system, a device to be controlled (e.g., game console, VR/AR headset, smart appliance) emits audio signals through its speakers. The audio signals are reflected by a users hand as well as other nearby objects and then received by the microphones on the same device. The device uses the reflected audio signal to continuously track the hand in real-time.

We use audio signals for tracking due to its slow propagation speed, which makes it possible to achieve high accuracy. Moreover, it can be generated and received by widely available speakers and microphones and can be processed in software without any extra hardware. It also works in a wide range of scenarios, such as non line-of-sight and different lighting conditions.

We introduce two approaches to achieve the acoustic signal based device-free tracking. The first one is Frequency Modulated Continuous Waves (FMCW) based tracking. Here, the speakers transmit FMCW. Upon receiving the audio signals, the microphones extract distance and velocity with respect to the speakers, and then derives its coordinates based on the estimated distance and velocity.

There are several challenges in achieving this goal. First, it is common to have more than one object around the speakers and they all reflect the signals. How to distinguish which one corresponds to the reflection from the

hand? Second, the frequency shift is caused by both propagation delay and velocity. The receiver can only measure the overall frequency change, and should decompose it to the change caused by the propagation delay and the one caused by movement. To address them, we introduce novel algorithms to filter out the reflections from static objects and separate the frequency shift by Doppler and the delay from observed frequency shift in FMCW.

Next, we introduce Acoustic Phase Tracking (APT) that tracks the movement of the finger using the acoustic signal in close proximity. We observe the phase change of the acoustic signal for tracking. The wavelength of the acoustic signal is very short in inaudible frequency range (*e.g.*, 1.7 cm in for 20 KHz signal). Considering that the phase of the signal rotates 2π when the travel distance changes as much as the wavelength, observing the phase allows us to track the movement in fine resolution.

The challenge is how to distinguish the phase change of the signal reflected from the finger from those of all other signals. In the received audio signal recorded by the microphone, the reflected signal from the finger is merged with many other reflected signals as well as the directly received signal. As a consequence, the phase tracking is affected by the movement of the surrounding objects besides the target movement. We address this by observing the phase change of the channel impulse response (CIR). In CIR, each channel tap value corresponds to the signal path with specific delay. By observing the phase change of certain taps, we can ignore the phase change incurred by the paths that are not related to the target movement. For the

channel estimation, we design our own acoustic communication channel that sends and receives training sequence frames in inaudible frequency band. From the received signal, we estimate the channel, observe the phase change, and eventually track the finger movement. The whole process is implemented in software and can be processed in real-time.

1.4 Summary of Contributions

In summary, the followings are the major contributions of the dissertation:

- We propose a novel device tracking mechanism that enables air mouse capability using existing hardware in mobile and electronic devices. It accurately estimates frequency shift and uses it to position the device when the distance between speakers and initial position of the device are both known. Then we develop techniques to estimate the distance between speakers and the device initial position. To further enhance robustness, we transmit signals at multiple frequencies, perform outlier removal, and combine the remaining estimations. We show AAMouse can track the trajectory of the device movement with a median trajectory error of 1.4 cm. This is comparable to RF-IDraw [63] that requires multiple antenna arrays of 4 antennas. Compared to Tagoram [72] with unknown track, it is 7 times more accurate.

- We propose a device-free hand tracking mechanism that accurately track

the movement of the hand. We estimate the distance and the velocity using a single chirp signal and combine both information to accurately locate the moving hand. Our evaluation shows that it achieves the median tracking error of 1.94 cm using 2 speakers and 2 microphones on the same computer without extra hardware.

- We provide a fine-grained finger tracking that tracks the movement of the finger in close proximity. It tracks the object by observing the phase change cause by the movement from the estimated channel impulse response. From the experiment using a mobile phone with one speaker and two microphones, we show it can accurately track the movement of finger with the median tracking error 1.1 cm.

1.5 Dissertation Outline

Chapter 2 overviews the related work. Chapter 3 shows how to track the mobile device, and Chapter 4 presents device-free tracking using the acoustic signal. Chapter 5 introduces fine-grained finger tracking, and Chapter 6 concludes the dissertation.

Chapter 2

Related Work

This chapter presents related work. Section 2.1 presents the work related to mobile device tracking, and Section 2.2 describes the work in related to device-free tracking.

2.1 Mobile Device Tracking

Localization and device tracking: Localization has received significant research attention over the last few decades. However, most of the previous works achieve coarse grained localization, and do not satisfy our goal of achieving centimeter-level accuracy. There are a few ranging-based localization papers that use acoustic signals to estimate the distance based on time-of-arrival (*e.g.*, [43, 74, 46]). Different from these ranging based works, we focus on continuous trajectory tracking to enable mouse functionalities. [57] uses 6 beacon nodes as transmitters to achieve a median error of 12 cm under the line-of-sight. To achieve such an accuracy, it requires special hardware, dense deployment, and line-of-sight, which may not be feasible in many cases. Another fine grained indoor localizations is ArrayTrack [71], but its median error is still 23 cm. Recently, there have been a few works that track the trajectory of RFID tags

in centimeter-level [62, 63, 72]. They require special RFID readers with many antennas. RF-IDraw [63] uses two RFID readers, each with 4 antennas, and Tagoram [72] uses four readers, each equipped with 4 antennas. In home environment, it is difficult to imagine that people deploy RFID readers to track their devices. We achieve comparable accuracy by exploiting two speakers that are available in most TVs, PCs, and laptops. Moreover, although Tagoram achieves millimeter-level accuracy when the target moves along a known track, in an unknown track their median error is 12 cm, well above our error (*i.e.*, 1.4 cm).

Using inertial sensors: [29] uses the acceleration measured from the accelerometer of the smart phone for localization. Due to significant error from the acceleration measurement, it uses dead reckoning approach that finds the position by estimating the number of steps and the heading direction instead of double integration. Zee [48] uses a similar approach to fill in the gap between the locations estimated using WiFi. Both of them achieve sub-meter level accuracy. Such an approach is suitable for tracking walking, but not hand movement, which is our focus.

Inertial sensors are also used for gesture recognition and device tracking [42, 5]. E-gesture [42] uses gyroscope and accelerometer for gesture recognition, but it is not applicable to device tracking. [5] tracks the device movement and recognizes the alphabet the user writes. Their tracking algorithm is similar to the acceleration based tracking used as a baseline comparison in Section 5.5. Our evaluation result shows that AAMouse achieves much higher accuracy

than the accelerometer based tracking.

Using Doppler for motion and device tracking: WiSee [44] is a novel method for device-free gesture recognition using the Doppler shift of the WiFi RF signal. Its main benefit is that users do not need to carry any device. Its focus is gesture recognition, which distinguishes one gesture from another, instead of estimating the exact position, so it is not sufficient for our purpose. A few other works [16, 12, 9, 60] use the Doppler shift of the audio signal for gesture recognition rather than device tracking. DopLink [9] enables the interaction among devices using the Doppler shift, and Spartacus [60] finds the direction of the device movement using the Doppler shift and pairs it with another device that moves in the same direction. Swadloon [19] exploits the Doppler shift of audio signal for fine-grained indoor localization. It assumes the position of the anchor nodes (*i.e.*, speakers) are known, and its error is around 50 cm, well below our accuracy requirement. [26] proposes acoustic signal based localization, but it uses chirp ranging instead of the Doppler effect. Its localization accuracy is close to 1 m, and its tracking accuracy during device movement has not been evaluated.

Infrastructure based tracking: Microsoft X-box Kinect [1] and Nintendo Wii [2] have been widely successful as gaming controllers. They augment the game reality by using motion tracking and gesture recognition. Kinect recognizes the gesture of the user using the depth sensing camera. Wii uses Infrared (IR) signal to track the movement of the controller. The sensor bar on the top of TV emits IR signal and the IR camera attached to the Wii

controller determines its position relative to the sensor bar by detecting the change in IR blobs [67]. Different from these devices, our approach enables mouse functionalities using the speakers and microphones that are already available in most TV and mobile devices and does not require line-of-sight.

Gyroscope based Air mouse: One of our main applications is to let a user point at a specific target using a mobile device as a mouse. There have been a few types of *Air mice* [6] in the market. These are usually used as remote controllers for laptops and PCs. Advanced remote controllers for smart TVs, such as Roku 3 [51], Samsung Smart Remote [52], and LG magic motion controller [28], provide similar functionality. They all use gyroscopes, which measure the angular velocity to track motion, where a user rotates his wrist to control the pointer position in a 2-D space. Rotating the device yields the change of the angular velocity, which is translated into the pointer movement. Such a movement is not intuitive to users. We performed user study to compare gyroscope based device control with our scheme. As shown in Section 5.5, users traverse 40% more distance to touch a target. When asked to draw simple shapes using the gyroscope, the outputs are rather different from the original shapes.

2.2 Device-Free Tracking

FMCW-based: Our work is inspired by several existing works on device free tracking, in particular, FMCW Radar. FMCW Radar uses large antenna arrays to achieve a narrow beam for tracking. Therefore it requires high-

power and is heavy. WiTrack [4] applies FMCW to WiFi to track a user's location with 10-13 cm error in x and y coordinates and 21 cm in z-dimension. In addition, since off-the-shelf radios do not perform FMCW, it implements on USRP. Different from FMCW Radar and WiTrack, our system uses widely available speakers and microphones and can run on laptops or on smartphones without additional hardware. We achieve high accuracy by using audio signals and extracting not only distance but also velocity from a single FMCW signal. ApneaApp [36] uses FMCW to track heartbeat by looking at periodic patterns. In comparison, continuous tracking is more challenging due to the lack of patterns.

Other device-free tracking: WiDraw [59] estimates angle of arrival (AoA) using CSI, and achieves a median tracking error of 5 cm using 25 WiFi access points (APs). The requirement of such a large number of APs significantly limits the applicability of WiDraw. In comparison, we only require 2 speakers and microphones on one machine to achieve higher accuracy. WiDeo [22] traces human motion based on reflected WiFi signals. It implements on WARP using 4 antennas and phase synchronized RX chains. Its median error is 7 cm. [75] uses 60 GHz radios and achieves cm-level accuracy for static objects with line-of-sight. mTrack [65] achieves higher tracking accuracy by using 60 GHz steerable antennas with narrow beamwidth (3.4^o) and fine-grained phase rotation (1^o). The major advantage of our system is ease of deployment since it requires no extra hardware and only software-based processing. FingerIO [37] uses the cross-correlation of the received signal to track the moving finger.

The transmitter sends OFDM signal with data symbols in inaudible frequency band. The receiver checks the cross-correlation between the transmitted and the received signals, which yields echo-profile that shows peaks at the locations of the reflecting objects. By observing the change of the echo-profiles between two consecutive frames, it detect the change of the moving finger location, and eventually tracks the movement. In Chapter 5, we performed extensive performance comparison with our scheme, we shows our scheme is much more accurate.

Device-free gesture recognition: WiSee [44] is a pioneering work that uses WiFi signals to enable device-free gesture recognition. It can recognize 9 gestures in several different environments. AllSee [23] enables RFID tags to recognize gestures based on existing signals, such as TV transmissions. [8] uses CSI from WiFi to infer which key a user types using significant tracking. [16, 12] use the Doppler shift of the audio signal to detect in-air gestures. In general, gesture recognition performs pattern matching and requires significant training data. In comparison, continuous tracking is more challenging due to the lack of training data or patterns to match against. CARM [64] recognizes the the activity of the user by analyzing CSIs available from WiFi NICs. It infers the movement speed of the user from the power changes incurred by the phase rotation of the signal path reflected by human body. The speed information extracted from the CSI is rough and inaccurate, so it is used only for the pattern matching based activity recognition and cannot be extended to general movement tracking.

Chapter 3

Precise mobile device tracking using acoustic signal

In this chapter, we show how to track the movement of a mobile device using acoustic signals.

3.1 Motivation

Most recent smart phones are equipped with MEMS accelerometer sensors to measure the movement of the device. A natural thought is to use the accelerometer for device tracking. In this section, we show that existing accelerometers do not provide high enough accuracy for tracking or enabling mouse functionalities. We describe fundamental reasons behind the limited accuracy.

Regarding the problem of the device movement tracking, what one can easily imagine is accelerometer based tracking. An accelerometer typically gives 3-axis output that allows us to calculate both the direction and speed of the movement. In theory, we can integrate acceleration to get velocity and integrate velocity to get the distance traveled. However, it is well known that the accelerometer based positioning offers limited accuracy [25, 38]. Here are

a few fundamental issues with using acceleration for device tracking.

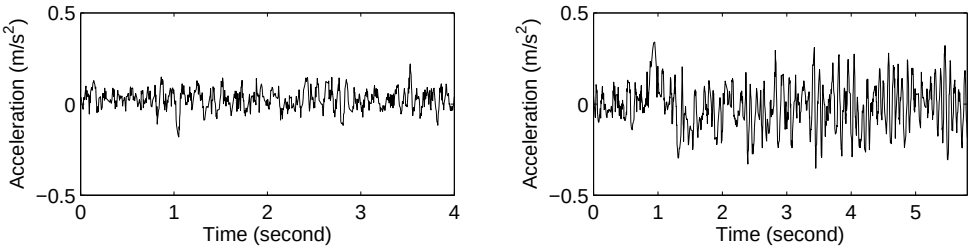
1. The measurement result is easily affected by gravity. While there have been some techniques to remove the effect of gravity, their accuracy is still limited.

2. Computing velocity based on acceleration is hard (*e.g.*, the acceleration is close to zero as the device movement speed approaches a constant speed, and the acceleration is large as the device suddenly stops).

3. Estimating distance traveled based on acceleration requires double integration, and small measurement error can easily get exploded during double integration.

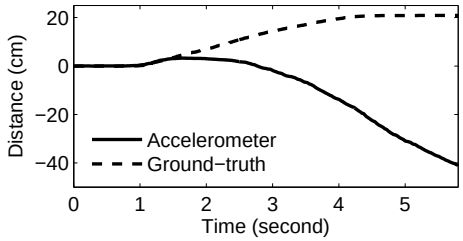
In this section, we conduct measurement studies and confirm that existing acceleration based tracking is error-prone. We will show more user study on accelerometer based tracking in Section 5.5.

To demonstrate these issues, we observe that the acceleration measured by the accelerometer inherently includes gravity, which is approximately 9.8 m/s^2 on earth. When the device lies perfectly flat, gravity only affects the acceleration of Z-axis, so X and Y-axis have zero accelerations. However, while holding the device, one cannot avoid natural hand vibration, which generates gravity induced acceleration in all axes. This acceleration is much larger than that caused by intentional movement (*e.g.*, which is around 0.12 m/s^2 on average in our experiments). This makes the acceleration based tracking unreliable.



(a) Acceleration while the device is not moving

(b) Acceleration while the device is moving



(c) Moving distance tracked by accelerometer

Figure 3.1: Acceleration while device is moving and not moving.

One way to remove gravity from the measured acceleration is to apply a high-pass filter [15]. As the frequency of gravity is low, it is considered that filtering low frequency acceleration is effective in reducing the impact of gravity. Google Android SDK provides a linear accelerometer API, which gives acceleration after a high-pass filter. This is helpful in removing gravity in a coarse scale, but it still does not provide sufficient accuracy to track the device position. This is due to two reasons: (i) filtering always incurs additional delay and (ii) residual acceleration after filtering is still large enough to result in significant positioning errors.

Next we perform the measurement of linear acceleration in Android devices. We use Google NEXUS 4 and Samsung Galaxy 3. Their accelerometers

come from InvenSense [20], which is one of the few accelerometer manufacturers that provides accelerometers for latest iPhones and many android phones, since InvenSense MEMS sensors are considered the most accurate.

Our experiments have 3 users. Different users do not give distinguishable differences in the results. The sampling rate of the accelerometer was set to the highest (*i.e.*, 200Hz). To get the ground-truth, we attach a camera on the ceiling, and run an object tracking program to track the mobile phone movement precisely.

Figure 3.1(a) shows a snapshot of the acceleration while a user is holding the device. We only plot Y-axis acceleration for ease of viewing. While a user is holding the device, the average magnitude is 0.05 m/s^2 across all users and devices. This can be regarded as measurement noise because it is generated while the device is not moving.

Figure 3.1(b) shows the acceleration while the user is moving the device in one direction. In this experiment, users are asked to mimic the speed of controlling a mouse. The device starts to move at 1 second, and stops at 4.3 second. Comparing Figure 3.1(a) and (b), it is hard to tell exactly when the device starts moving and when it stops. At the first glance, we might observe the value difference while the device is moving and not is negligible. At the beginning of the movement, it gives high enough acceleration to calculate the initial velocity, but as the device velocity approaches close to a constant, the acceleration decreases, which makes the position tracking unreliable. In addition, even after the movement has stopped, the acceleration remains high

due to deacceleration and filtering noise.

Figure 3.1(c) shows the distance calculated by double integration of the acceleration. We can observe the tracking error is significant, and the direction is completely off. While this is just one run, it is common to see such a significant error in other runs. The average acceleration while the device is moving is 0.12 m/s^2, which is not very different from the measurement noise. Using double integration to get the distance causes small measurement error to get accumulated quickly. Figure 3.1(c) shows that the distance increases even after the device has stopped due to the accumulated error. This shows that the existing accelerometers do not provide high enough accuracy to track small-scale movements.

Therefore, we conclude that the MEMS accelerometers in current smart phones do not give accurate acceleration to track small scale movements.

3.2 Our Approach

This section presents our approach to device tracking.

3.2.1 Doppler Effect Based Tracking

The Doppler effect is a well known phenomenon where the frequency of a signal changes as a sender or receiver moves [34]. The frequency increases as they move closer, and decreases as they move far away. The amount of the frequency change is called Doppler shift, which is determined by the speed of the sender or receiver. Without loss of generality, we consider that only

the receiver moves while the sender remains static. Let F denote the original frequency of the signal, F^s denote the amount of frequency shift, and v and c denote the receiver's speed towards the sender and the propagation speed of wave, respectively. They have the following relationship:

$$v = \frac{F^s}{F} c. \tag{3.2.1}$$

So if we know F and c and can measure F^s, we can use the above relationship to estimate the speed of movement. Compared to the acceleration that requires double integration to get the distance, the Doppler shift allows us to get distance using a single integration, which is more reliable.

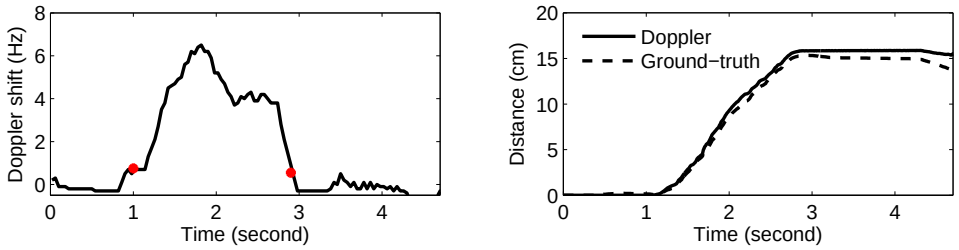
The Doppler effect is observed in any wave, including RF and acoustic signal. We use acoustic signal to achieve high accuracy due to its (i) narrower bandwidth and (ii) slower propagation speed. Its narrower bandwidth makes it easy to detect 1 Hz frequency shift than that in RF signals (*i.e.*, 44.1 KHz in acoustic signals versus 20 MHz in WiFi). In order to improve the accuracy of detecting frequency shift in WiFi signals, WiSee [44] proposes to reduce the bandwidth of the received signal by first decoding the data to get the channel estimate and then re-encoding repeated data using the estimated channel. Even with significant computational overhead, WiSee detects only 2 Hz frequency shift. In comparison, we can easily detect 1 Hz frequency shift in real-time using acoustic signal. Even assuming that we can detect 1 Hz frequency shift in both WiFi and acoustic signals, the accuracy in speed estimation is still higher in acoustic signal due to its slower speed. The acoustic signal travels at 346. m/s in dry air at 26 °C. If we use sound frequency of 17 KHz, the

speed resolution is $\frac{1\times346.6}{17000} \approx 0.02m/s = 2cm/s$. In comparison, when the RF center frequency is 2.4 GHz, the resolution is $\frac{1\times3\times10^8}{2.4\times10^9} = 0.125m/s = 12.5cm/s$, which is around 6 times as large. This implies that for the same movement, the Doppler shift of the acoustic signal is $6\times$ that of RF signal, which allows us to more accurately measure the movement speed.

Moreover, it incurs less overhead to measure the Doppler shift. To observe the signal in 1 Hz level in the frequency domain, one needs to perform the FFT with respect to the bandwidth of the signal [40]. While the bandwidth of the acoustic signal is 44.1 KHz, WiFi signal bandwidth is 20 MHz. In Section 4.2, we describe how we observe the Doppler shift in 1 Hz level in real-time. WiSee reduces the Doppler resolution to 2 Hz to relieve the computational overhead, which further decreases the resolution of the speed.

In addition, acoustic signal can be easily generated and received using speakers and microphones, which are widely available on TVs, Google Glasses, smartphones, and smart watches. To avoid disturbance to other people, we can generate inaudible acoustic signals. While in theory some people may hear up to 20 KHz, we find sound above 17 KHz is typically inaudible. We can easily emit inaudible sound using any device that can play audio files with a sampling rate of at least 44.1 KHz.

We perform a simple experiment to see how accurately we can track the device movement using the Doppler shift. Using MATLAB, we generate a 17 KHz sinusoid audio file that takes 1 Hz in the frequency domain, and play it using a normal PC speaker, and record it using a microphone on Google



(a) Doppler shift (b) Moving distance

Figure 3.2: The Doppler shift and the moving distance estimation.

NEXUS 4. We measure the Doppler shift while the device is moving towards the speaker. The details on how to accurately calculate the Doppler shift will be explained in Section 3.2.2. Figure 3.2 shows the Doppler shift and the moving distance over time estimated by Equation 3.2.1. The device starts moving at 1 second and stops at 2.8 second. Unlike acceleration measurement in Figure 3.1, it is easy to tell the start and stop time of the device movement (*e.g.*, the Doppler shift is well above 1 Hz during movement and well below 1 Hz when it stops). Moreover, since we get speed from the Doppler shift and can calculate the distance traveled using a single integration rather than double integrations in accelerometer, the accuracy improves significantly. As shown in Figure 3.2(b), the maximum tracking error is only 0.7 cm.

Based on the above concept, we develop the following system, as illustrated in Figure 4.2, where a sender with two speakers sends inaudible sound pulses to a mobile device to be tracked. The mobile device can be any device with a microphone, such as a smart phone and smart watch. Here we consider a smart TV as a source of the signal but it can be any device with two speakers such as laptop and PC. Also, the device to be tracked can be any mobile device

with a microphone, such as smart phone and smart watch. The audio signal sender also serves as a main processor that calculates Doppler shift and tracks the device. The basic scenario is as follows. The TV emits narrow sound wave in inaudible frequency range.

To distinguish which speaker generates the signal, the two speakers emit different frequencies. The device initiates tracking using a simple gesture or tapping the screen, and starts recording the audio signal from the microphone. The mobile device can either locally process the received signal to compute its location, or send the received audio file via a wireless interface (*e.g.*, WiFi or bluetooth) back to the sender for it to process the data and track the device. The audio signal is simply a sequence of pulse-coded modulation (PCM) bits, which is typically 16 bits per sample. Assuming 44.1 KHz sampling rate, the amount of the audio data per second is 705.6 Kb, which is lower than the bit-rate of classic Bluetooth (*i.e.*, 2.1 Mbps) [18]. Depending on the application, it can be translated into the cursor position or used to track the trajectory of the user's movement.

There are several important research issues we should address to realize the above system:

- How to estimate the Doppler shift?
- How to estimate the position based on the Doppler shift?
- How to improve robustness?

- How to determine the distance between speakers?
- How to determine the mobile device's initial position?
- How to extend this approach to control devices without multiple speakers?

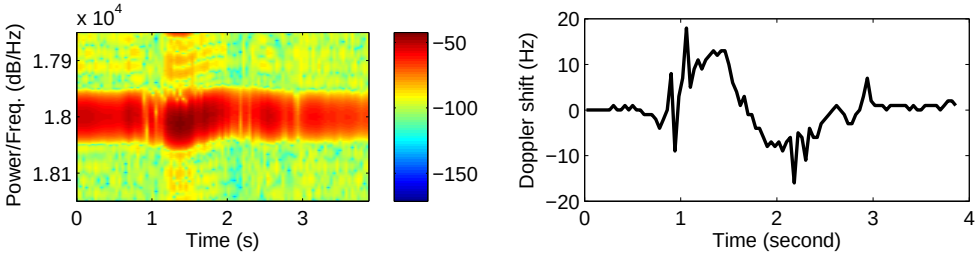
Below we address each of these challenges in turn. Note that we focus on tracking in a 2-D space for mouse applications. Other applications may require tracking in a 3-D space. Our approach supports 3-D tracking when there are 3 or more anchor points (*e.g.*, 3 speakers).

3.2.2 Estimating the Doppler Shift

To record sound in inaudible range (*i.e.*, between 17KHz and 22KHz) without aliasing, we should use the audio sampling rate at least 44 KHz [40]. We use 44.1 KHz sampling rate since most android devices support it by default. To achieve high tracking accuracy, we aim to estimate frequency shift with a resolution of 1 Hz. This implies we need to perform 44100-point FFT to analyze the signal in the frequency domain in 1 Hz-level. This poses a challenge: a long FFT does not allow us to continuously track a device in real time. For 44100-point FFT, we need to store 44100 samples, which takes one second. During that time, the device's position might already have changed several times, which significantly degrades the reliability and responsiveness of tracking.

To address this issue, we use Short-term Fourier Transform (STFT), which is used to analyze the change of spectrum over time [66]. It uses fewer data samples than that required by FFT. The missing values of the input is filled with zeros. Then the FFT output has desired resolution. However, this alone may cause aliasing due to under-sampling. To minimize the distortion, windowing is applied in time domain and each window contains all the audio samples during the current sampling interval. We use Hamming window for that purpose [21].

In our design, we set input length to 44100 and use 1764 audio samples (*i.e.*, the total number of audio samples in 40 ms) as the input, which gives FFT output with 1 Hz resolution every 40 ms. From it, we measure the Doppler shift by finding the peak frequency (*i.e.*, the frequency with the highest value) and subtracting it from the original signal frequency. The complexity is determined by the width of spectrum to be scanned in order to detect the peak frequency. We set it to 100 Hz assuming that the maximum possible Doppler shift is 50 Hz, which corresponds to 1 m/s. According to our experiment, when a person moves a mobile device with a hand, its speed does not exceed 1 m/s. Figure 3.3(a) and (b) show an example of the received audio signal in the frequency domain and the estimated Doppler shift, respectively, while the device is moving around a circle.



(a) Spectrogram (b) Doppler shift

Figure 3.3: Doppler analysis of the received signal.

3.2.3 Tracking the Position

Next we use the estimated frequency shift to derive the position of the mobile device. In this section, we assume the distance between the speakers and the initial position of the mobile device are both known. We will relax these assumptions in Section 3.2.5 and 3.2.6. With the measured Doppler shift, we can calculate the device movement speed and the moving distance given the amount of time.

We estimate the frequency shift from the two speakers to get the distance change from the speakers. More specifically, let D denote the distance between the speakers. We construct a virtual two-dimensional coordinate where the origin is the left speaker and the X-axis is aligned with the line between speakers. In this coordinate, the left and right speakers are located at $(0, 0)$ and $(D, 0)$, respectively. Let (x_0, y_0) denote the device's initial position in this coordinate. The distances from the device to the speakers 1 and 2 are denoted by $D_{0,1}$ and $D_{0,2}$, respectively , which is

$$D_{0,1} = \sqrt{x_0^2 + y_0^2},$$
$$D_{0,2} = \sqrt{(D - x_0)^2 + y_0^2}.$$

Let t_s be the sampling interval in which we estimate the frequency shift. Our evaluation uses 40 ms, which means the cursor's position is updated every 40 ms, which corresponds to popular video frame rates of 24-25 frames per second [14]. After t_s, we can get the new distance from the two speakers $D_{1,1}$ and $D_{1,2}$ using the Doppler shift. From the measured Doppler shift and Equation 3.2.1, we get:

$$D_{1,1} = D_{0,1} + \left(\frac{F^s_{1,1}}{F_1}c\right) t_s,$$
$$D_{1,2} = D_{0,2} + \left(\frac{F^s_{1,2}}{F_2}c\right) t_s,$$

where F_k and $F^s_{i,k}$ are the sound frequency and Doppler shift from speaker k during the i-th sampling interval, respectively.

Given the updated distance from the speakers, the remaining question is how to get the new position. As illustrated in Figure 3.4, the new position should be the intersection of the two circles whose center points are $(0, 0)$ and $(D, 0)$, and radii are $D_{1,1}$ and $D_{1,2}$, respectively. We can calculate the intersection of the two circles efficiently as follows [3]:

$$\theta_1 = cos^{-1}\left(\frac{D_{1,1}^2 + D^2 - D_{1,2}^2}{2DD_{1,1}}\right),$$
$$(x^1, y^1) = (D_{1,1}\cos(\theta_1), D_{1,1}sin(\theta_1)),$$
$$(x^2, y^2) = (D_{1,1}cos(-\theta_1), D_{1,1}sin(-\theta_1)),$$

where (x^1, y^1) and (x^2, y^2) are two intersection points of the circles. Note that if $D_{1,1} + D_{1,2} < D$, there is no intersection between the two circles. If $D_{1,1} + D_{1,2} = D$, there is one intersection. In the other cases, there are two intersection points. In the last case, we choose the point closer to (x_0, y_0) as the next position, denoted as (x_1, y_1), since the movement is continuous and our sampling interval is small.

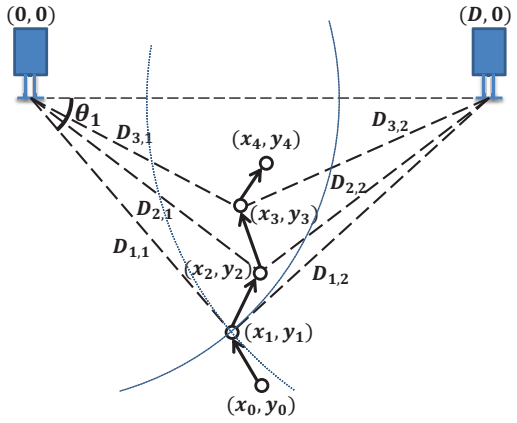
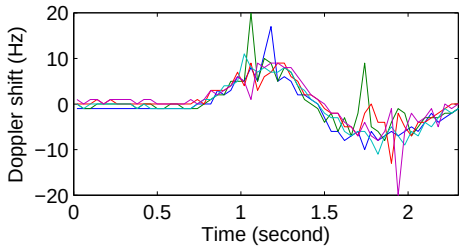
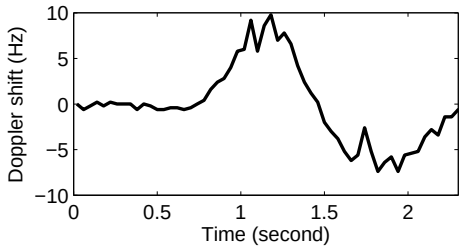


Figure 3.4: Tracking the position based on the Doppler shift.

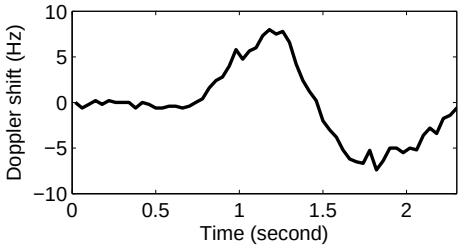
In the next Doppler sampling interval, we measure $F_{2,1}^s$ and $F_{2,2}^s$, calculate $D_{2,1}$ and $D_{2,2}$, and derive (x_2, y_2) from it. This process is repeated until the device stops moving. To minimize the impact of errors in the frequency shift



(a) Doppler shift measured from 5 tones



(b) Doppler shift after MRC without outlier removal



(c) Doppler shift after MRC with outlier removal

Figure 3.5: Improving the accuracy of the Doppler estimation.

estimation, we filter out the frequency shift below 1 Hz and use the remaining frequency shifts to estimate the speeds and distance.

We detect the start and end of device movement using simple thresholding. If the absolute value of the measured Doppler shift is larger than 1 Hz for more than 5 consecutive intervals, it starts tracking the device. Similarly, we regard that the device has stopped if the Doppler shift is smaller than 1 Hz during 5 consecutive intervals.

3.2.4 Improving the Accuracy

To achieve high accuracy in device tracking, it is crucial to accurately estimate the Doppler shift. However, measuring it from a single sound wave

may not be reliable. The accuracy of estimating the Doppler shift in part depends on SNR of the received signal. Due to frequency selective fading, SNR varies across frequencies. Our idea to improve the accuracy is using larger spectrum for Doppler measurement. In particular, we exploit the diversity idea used in wireless communication. If we send 1-Hz sound tones at different center frequencies, we can use all of them to measure the Doppler shift.

In order to leverage multiple frequencies, the first question is which center frequencies should be used. If the different center frequencies are too close, they will interfere with each other especially under movement. As mentioned earlier, the hand movement speed for mouse applications is typically within 1 m/s, which corresponds to 50 Hz Doppler shift. To be conservative, we set adjacent sound tones to be 200 Hz apart. Based on our evaluation result in Section 3.4.1, we allocate 10 sound tones for each speaker.

The next question is how to take advantage of the measurements at multiple frequencies to improve the accuracy. One approach is to apply Maximal Ratio Combining (MRC) technique used in the receiver antenna diversity, which averages the received signal weighted by the inverse of the noise variance. It is known to be optimal when the noise follows a Gaussian distribution [69]. However, we find some frequencies may incur significantly higher noise than others, and it is important to remove such outliers before combining them using a weighted average. In our system, the Doppler sampling interval is 40 ms. 10 Hz difference from the previous measurement implies the velocity has changed 0.2 m/s during 40 ms, which translates into an acceleration

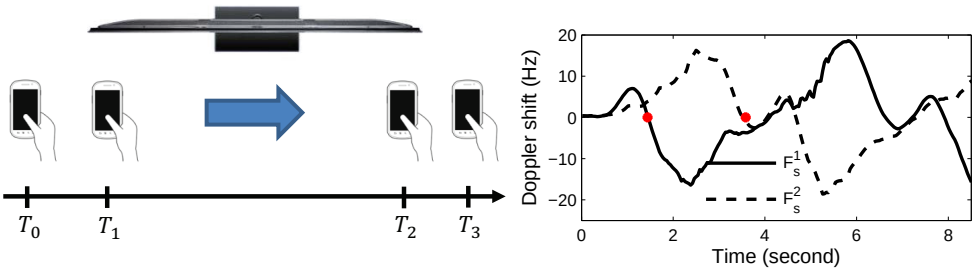
of 5 m/s^2. Such a large acceleration is unlikely to be caused by the device movement. So whenever the change in frequency shifts during two consecutive sampling intervals (*i.e.*, $|F^s_{i+1,k} - F^s_{i,k}|$) is larger than 10 Hz, we consider it as an error and remove it before performing MRC. In an exceptional case where all measurement differences exceed 10 Hz, we select the one closest to the previous measurement. After MRC, the Kalman filtering is applied to smooth the estimation. We set the process noise covariance Q and measurement noise covariance R in the Kalman filter both to 0.00001.

Figure 3.5 shows the raw Doppler shift measurements and the result after MRC with and without outlier removal. It shows that the Doppler estimation after outlier removal yields more smooth output and likely contains smaller errors. In Section 3.4.1, we show that the outlier removal improves the accuracy by 26% when 5 sound tones are used.

3.2.5 Finding the Distance between the Speakers

So far, we assume the distance between the speakers is known a priori. In practice, this information may not be available in advance. One solution is to ask the user to measure the distance between the speakers using a ruler and report it. This is troublesome. Moreover, sometimes users do not know the exact location of the speakers. Therefore, it is desirable to provide a simple yet effective calibration mechanism that measures the speaker distance whenever the speakers' positions change.

We propose a Doppler based calibration method. It only takes a few



(a) Calibration process.

(b) Doppler shift while scanning the TV (round trip).

Figure 3.6: Illustration of the calibration process.

seconds for a user to conduct calibration. As shown in Figure 3.6(a), during the calibration, the TV emits inaudible sound and the device records it using its microphone. The user scans the TV with his hand holding the device. The user starts from the left end of the TV, and move towards the right end of the TV in a straight line. The user stops after it moves beyond the right end, and comes back to the left end. The user can repeat this procedure a few times to improve the accuracy.

Figure 3.6(b) shows the change of the Doppler shift while a user is performing the calibration. We can easily detect the time when the device moves past the left and right speakers, and measure the distance by calculating the movement speed based on the Doppler shift. As mentioned in Section 3.2.1, the Doppler shift is positive as the receiver moves towards the sender. When the device is at the left side of both speakers, both F_1^s and F_2^s are positive as it moves towards the right. As it passes the left speaker at 1.48 second (as shown in Figure 3.6), F_1^s changes to negative while F_2^s stays positive. Similarly, as the device passes the right speaker, F_2^s changes from positive to negative. By

finding these points, we find the amount of time user spends moving between the two speakers, and measure the distance using the Doppler shift. To improve the accuracy, we obtain one distance estimate in each direction, and average the estimated distances in both directions.

One question is how many repetitions are required to achieve reasonable accuracy. It depends on the distance error and its impact on device tracking. To better understand the impact, we inject error to the distance estimation. As shown in Section 5.5, when users repeat the calibration three times (*i.e.*, moving the mobile device back and forth for three times), the 95 percentile error is 5 cm. The experiment also shows the impact of 5 cm speaker distance error on device tracking is negligible. Therefore, three repetitions is generally sufficient.

3.2.6 Finding the Initial Device Position

Next we consider how to handle the issue that the device's initial position is unknown. To address this, we use particle filter. Particle filter has been successfully used in localization to address the uncertainty of the location [48, 13]. We use it in the following way. Initially, we generate many particles uniformly distributed in an area where each particle corresponds to a possible initial position of the device. In the next Doppler sampling interval, it determines the device movement from the current particles. If the device movement is not feasible, the particle is filtered out. In Section 5.3.4, we mentioned that the position of the device is determined by finding the intersection

of the two circles. If $D_1 + D_2 \geq D$, we can find one or more intersections; otherwise, there is no intersection. In this case, we regard the current particle is infeasible and filters it out. The device movement is determined by averaging the movement of the all remaining particles.

More specifically, let $\mathbf{P}$ be the set of particles, which is initialized as $\mathbf{P} = \{(x_o^1, y_o^1), ..., (x_o^N, y_o^N)\}$, where (x_o^k, y_o^k) is the initial position of the k^{th} particle and N is the number of particles. During a new Doppler sampling interval, the particles that give infeasible movement are filtered out from $\mathbf{P}$. After the i^{th} movement, the position at the $(i+1)^{th}$ sample is tracked by averaging the difference between the $(i+1)^{th}$ and i^{th} particle positions. That is,

$$(x_{i+1}, y_{i+1}) = (x_i + \frac{\sum_{k \in \mathbf{P}} (x_{i+1}^k - x_i^k)}{|\mathbf{P}|}, y_i + \frac{\sum_{k \in \mathbf{P}} (y_{i+1}^k - y_i^k)}{|\mathbf{P}|}),$$

where $|\mathbf{P}|$ is the number of remaining particles in $\mathbf{P}$.

The question remains how many particles to allocate. There is a trade off between complexity and accuracy. Increasing the number of particles is likely to increase the accuracy of initial position estimation. We use 625 particles to balance the trade off. 625 particles take 3.63 ms to process, well below 40 ms sampling interval. Refer to Section 3.3 for further details of this choice.

In Section 3.4.1, we evaluate the impact of the initial position error on the tracking accuracy. It shows the tracking accuracy is not sensitive to the initial position error. Even with 20 cm error, the median tracking error increases by only 0.2 *cm*. Note that we determine the initial position not

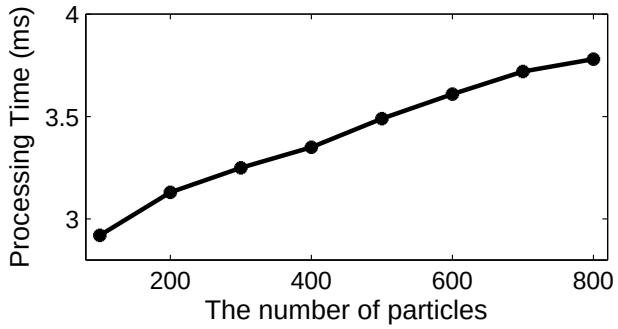


Figure 3.7: Processing time with a varying number of particles.

because we want to know the exact initial location of the device, but because we need it to track the next position. Regardless of the physical position of the device, the cursor always starts from the center of the screen. So the initial position error does not directly translate into the tracking error.

3.3 Implementation

We implement our system and validate the feasibility of the real-time device tracking. The mobile program is implemented on Android, which collects the inertial sensor data or audio signal and sends it to the tracking processor through Android debug bridge. Tracking is implemented in JAVA, which tracks the position of the device using the audio signal in real-time, as described in Section 4.2. For comparison, we also implement other tracking methods based on an accelerometer, gyroscope, and camera.

AAMouse: Unless otherwise specified, we use the Doppler based tracking. Since tracking requires non-negligible amount of computation, all tracking algorithms are handled by the tracking processor while the mobile device is

only responsible for delivering the recorded audio samples to the processor. To support real-time processing, we observe the main processing overhead is FFT. During each Doppler sampling interval (*i.e.*, 40ms), we 1) perform 44100-point FFT, 2) find peak frequency in each band, 3) combine them after removing outliers, and 4) calculate the position for each particle. Except FFT, the complexity is $O(WN + P)$, where W is the width of spectrum to scan for each tone, N is the number of tones, and P is the number of particles. To minimize the computational overhead, we set W and N to 100 and 10, respectively. To determine how many particles to use, we measured the processing time of tracking that is performed every Doppler sampling interval (*i.e.*, 40 ms) while varying the number of particles from 100 to 800. We used a PC with Intel i7 2GHz processor and 4GB memory as our testbed. As shown in Figure 3.7, there is a constant overhead of 2.8 ms on average, which does not depend on the number of particles. The additional overhead incurred by particle filtering is relatively small, and linearly increases with P. Therefore, we allocate $25 \times 25 = 625$ particles every 20 cm to cover $5m \times 5m$ space. In this case, it takes 3.63 ms to process the data, well below 40 ms sampling interval. The computation overhead is reduced further because the particles are filtered out during the tracking.

We also evaluate the Doppler and phase based tracking for an object that has only one speaker. We use USRP nodes as a sender and receiver to get the phase information, and attach the USRP receiver antenna to the smartphone with a microphone so that the WiFi and audio signals are received

at the same location. We implement a UHD program that records the phase of the received RF signal at USRP and feeds it to the tracking program, which tracks the device location using the phase change of the RF signal as well as Doppler shift of the acoustic signal. While we use USRP to generate RF signals, one can also use commodity WiFi cards to get the phase information and remove CFO [24, 25].

Accelerometer based tracking: For comparison, we also implement accelerometer based tracking, which uses double integration of acceleration to estimate distance. We follow the instruction in [53]. One of the difficulties in accelerometer based positioning is to determine the threshold used for detecting device movement. It is difficult to find a proper threshold that achieves both low false positive and low false negative. So we set a low threshold (*i.e.*, 0.07 m/s^2) to avoid treating device movement as *stop* by mistake.

Gyroscope based tracking: This mimics the gyroscope based tracking used in commercial air mouse products [6] and smart TV motion controllers [51, 52]. The gyroscope measures the angular velocity, and translates it into a 2-D displacement of a cursor.

Camera based tracking: The goal of camera based tracking is to get the ground-truth of the device movement. Using a camera attached to the ceiling of our laboratory, we track the position of the device very precisely. We implement a simple vision based object tracking program using OpenCV [10]. To improve the accuracy, we attach a label with a distinct color (blue in our

setup) to the device and implement a program to track that color. Camera based tracking is suitable for providing the ground truth in our setting, but is not a practical solution in general, since it requires not only lines of sight to the object at all time but also computational intensive and error-prone object recognition and tracking (especially for a general object and background).

3.4 Evaluation

Experimental setup: We evaluate the tracking performance of AAMouse and compare its usability with the other tracking methods. We use a DELL XPS desktop with Intel i7 CPU and 4GB memory as a main processor, and use Google NEXUS 4 as our main mobile device for user study. The audio source is Logitech S120 2.0 speaker. The volume of the speaker is set to 30 out of 100 to make sure it works in a normal volume range. By default, the distance between the two speakers is 0.9 m, and the distance between the speakers and the mobile device is around 2 m. We also vary these values to understand their impacts.

For our experiment, we implement a program that displays a pointer and the trajectory of the device movement in real-time so that users can see how well the pointer moves as they intend. To emit sound tones in inaudible frequency range, we play a wave format audio file with specific frequency tones that we generate using MATLAB. The left and right speakers use 17 - 19 KHz and 19 - 21 KHz spectrum, respectively. Each speaker emits 10 1-Hz sound tones with 200 Hz spacing in between. The best 5 tones are selected to estimate

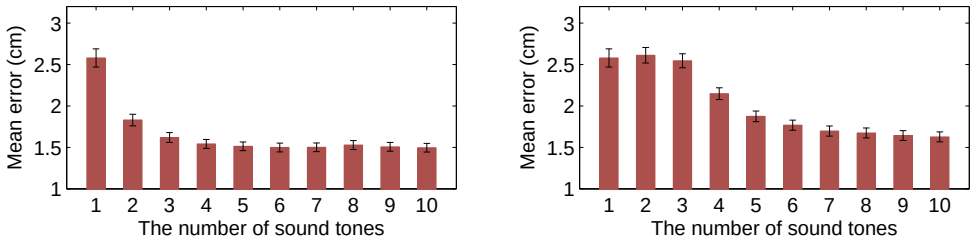
the Doppler shift as described in Section 3.4.1.

Evaluation methodology: We conduct user study with 8 students (5 male and 3 female students). They use all four schemes: AAMouse, accelerometer, gyroscope, and camera based tracking. Each user has 10-minute training for each scheme. When using AAMouse, the users are asked to hold the device in such a way to avoid their hands blocking the microphone. When evaluating the accelerometer and gyroscope based approaches, the users hold the device so that the phone is parallel to the line defined by the two speaker so that the coordinates of the accelerometer or gyroscope are consistent with the coordinates defined by the speakers. To quantify the accuracy and usability, we perform two kinds of experiments. The first experiment is to validate if the mobile device can be used as a pointing device like a mouse. We show a target on the screen, and ask the user to touch it. Each user repeats for 40 times. In the second experiment, we show simple shapes, such as a circle, triangle, diamond, and heart on the screen, and ask the users to use a mobile device to trace the shapes on the screen as closely as possible so that we can quantify how accurately they can control the pointer. Each user draws a shape for 5 times. The overall experiment lasts about one hour for each user excluding the training time. The average distance of the trajectory in the pointing and drawing experiments are 21.1 cm and 65.4 cm, respectively. The average tracking times for the pointing and the drawing are 3.2 and 8.4 seconds, respectively. The tracking and visualization are both done online in real-time. Meanwhile, we also store the complete traces to compute various

performance metrics offline. The error is measured by comparing the actual pointer trajectory with that from the camera based tracking. We also compare the distance traveled in order to touch a target, which reflects the amount of user effort.

3.4.1 Micro benchmark

Before presenting user study results, we first show a few micro benchmarks.



(a) With outlier removal. (b) Without outlier removal.

Figure 3.8: Trajectory error with different numbers of sound tones.

Trajectory Accuracy: To quantify the accuracy of trajectory tracking, we compare the trajectory a user traverses when trying to touch a target or draw a shape using AAMouse with the trajectories tracked by the camera, which serves as the ground truth. We sample the trajectory tracked by AAMouse at the camera sampling rate (*i.e.*, every 50 ms). For each sampled point, we compute the distance to the closest point on the trajectory tracked by the camera. We use the average distance across all sampled points as the error metric.

Figure 3.8(a) shows the mean trajectory error as we vary the number of sound tones used to estimate the Doppler shift, where the error bars represent 90% confidence interval. We observe that the mean trajectory error decreases from 2.7 cm to 1.9 cm by increasing the number of sound tones from 1 to 2. The error decreases further as we use more sound tones, but the improvement is marginal. Therefore, we decide to use 5 tones in the remaining evaluation to balance the tradeoff between accuracy and computation overhead.

Figure 3.8(b) is the mean trajectory error without the outlier removal introduced in Section 3.2.4. The results show that the outlier removal is crucial. If outliers are not removed, the benefit of using more tones is smaller and it requires more sound tones to achieve the same accuracy. This is because we need many samples to mitigate the significant error introduced by the outliers. With 5 sound tones, the mean trajectory error with outliers is 26% higher than that without outliers; even with 10 sound tones, the mean trajectory error with outliers is still around 6% higher.

Speaker distance error: In this experiment, a user moves the mobile device across the sender's left and right speakers to estimate the distance between its two speakers, as described in Section 3.2.5. The actual distance between the speakers is 0.9 m.

Figure 3.9 shows the CDF of distance error as we vary the number of scannings, where a scan means that a user moves the device from the left end to the right end, passing both speakers, and then comes back (*i.e.*, one round-trip). As we would expect, increasing the number of scannings reduces

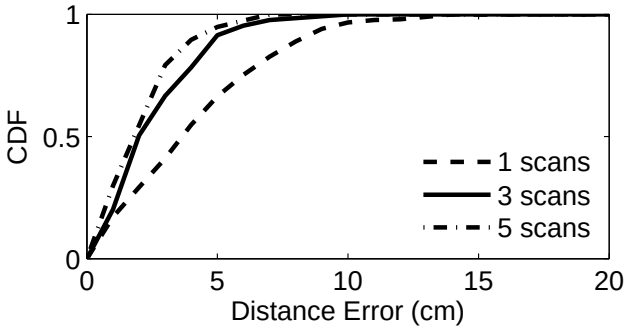


Figure 3.9: CDF of the speaker distance.

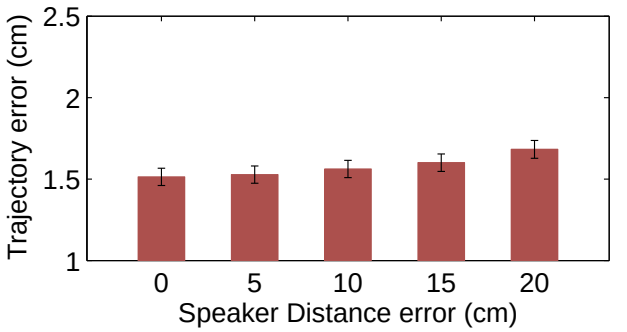


Figure 3.10: Trajectory error with various speaker distance.

the measurement error. When the user scans three times, 95% and 99% of the experiments have measurement errors within 5 and 10 cm, respectively. Further increasing the number of scannings yields marginal improvement in the accuracy.

Next we examine how the error in the speaker distance estimation affects the tracking accuracy. We inject varying amount of error to the speak distance estimation. As shown in Figure 3.10, the tracking error increases slightly with the increasing error in the speaker distance estimation. The speaker distance estimation error of 10 cm increases the mean error by 0.05

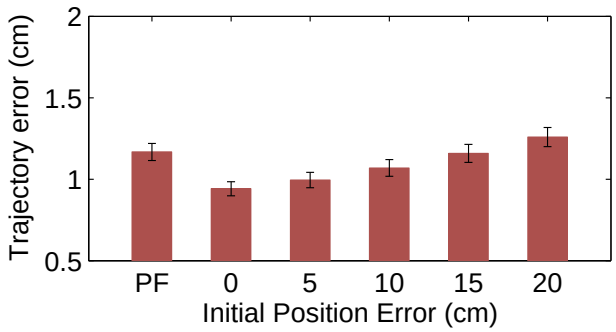


Figure 3.11: Trajectory error as the error of the device initial position varies.

cm (*i.e.*, from 1.51 to 1.56 cm). In Section 5.3.4, we explain that the position is tracked by finding the intersections of two circles whose center points are the speaker locations. Erroneous estimation of speaker positions degrades tracking accuracy, but the degradation is not significant, as shown in Figure 3.10.

Impact of initial position error: Finally, we show the impact of the initial position estimation error on the tracking accuracy. As it is difficult to know the initial position of the device in advance, we use particle filter as described in Section 3.2.6. For this experiment, the users start moving the device from a fixed initial position and perform the same experiments of pointing and drawing. We compare the tracking error when (i) the initial position is known, (ii) the initial position is known with a fixed amount of error, and (iii) using particle filter to estimate the initial position. Note that this is the only experiment where users start from the fixed initial position, while in all the other experiments users all start from an arbitrary initial position, which is more realistic.

Figure 3.11 presents the mean tracking error where *PF* is our particle filter approach. If the initial position is known, the accuracy can be further improved, but this is often infeasible. With particle filter, we can limit the impact of the initial position error. The result shows that the tracking error with particle filter is quite small: 1.4 cm, which is slightly larger than the case when there is 15 cm error in the initial position estimation, but smaller than that of 20 cm error in the initial position estimation.

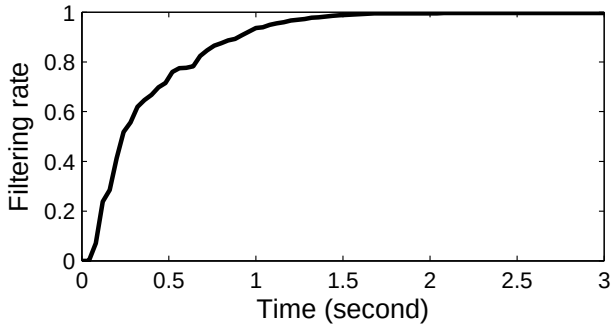


Figure 3.12: Particle filter convergence time.

Particle filter convergence: Figure 3.12 shows the particle filter convergence. Using the user experiment data, we plot the average fraction of the particles that are filtered over time. The result shows that after one second since the tracking starts, 93% of particles are filtered out. This demonstrates that particle filter converges fast.

3.4.2 AAMouse evaluation

Tracking accuracy: Next we compare the accuracy of AAMouse with the accelerometer based tracking. Their errors are measured in the same way as

described in Section 3.4.1. Figure 3.13 shows the cumulative distribution of the trajectory errors for AAMouse and the accelerometer based tracking. The median error for AAMouse is 1.4 cm while that for the accelerometer is 17.9 cm. The 90th percentile errors of AAMouse and accelerometer are 3.4 cm and 37.7 cm, respectively. AAMouse has 10 times lower error than accelerometer in terms of both the median and 90th percentile errors.

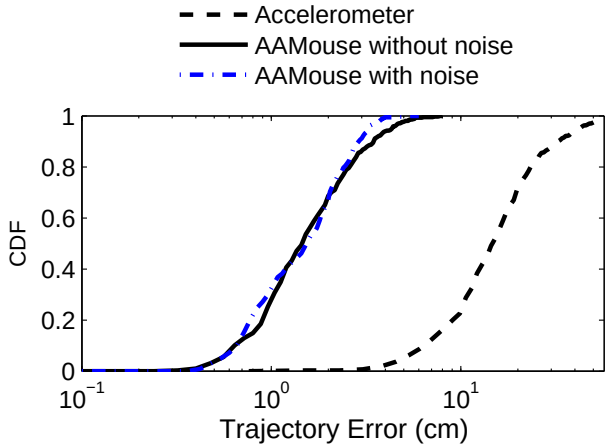


Figure 3.13: CDF of trajectory error.

Tracking accuracy under background noise: In order to evaluate the robustness of AAMouse against background noise, we perform additional experiments on AAMouse while the speakers generate both music and inaudible sound tones for tracking. This emulates the scenarios where smart TVs play background sound while sending inaudible acoustic signals for device tracking. According to [27], electronic music tend to generate noise at higher frequencies than other music genres. Therefore we use a few music clips in YouTube and select "Top EDM 2014 Music Playlist Tracklist" [35] as the background noise.

As shown in Figure 3.13, the accuracy remains the same. This is because the frequency of human voice and many music genres, such as Jazz, Pop and Classic, do not exceed 10 KHz. Even for some music with clangs and artificial sounds, such as rock and electronic music, their frequency hardly exceeds 15 KHz. Therefore they do not affect AAMouse, which uses higher than 17 KHz frequency band. This is also consistent with the findings reported in Spartacus [60], which uses inaudible sound for device pairing.

Target pointing evaluation: To evaluate the usability of AAMouse as a pointing device, we measure the distance that the pointer travels in order to touch a specific target. The main purpose of this experiment is to evaluate how easily one can control the pointer. If the device tracking is inaccurate, the pointer movement will be erroneous, which will increase the distance traveled. Similarly, if the tracking mechanism is not intuitive, the pointer might not move as the user intends, which will also increase the distance. We use $R = D_a/D_s$ to quantify the accuracy, where D_a and D_s are the actual distance traveled and the shortest path distance, respectively. When the metric is 1, it means that the movement follows the shortest path and has no tracking error. A larger value indicates a higher tracking error or harder to use.

We compare AAMouse, gyroscope, and camera-based tracking, which all succeed in touching the target every time since the users are required to change the pointer position until the target is reached. We do not compare with the accelerometer based scheme since its error is too large: despite significant time and effort users spend, they still fail to touch the target in over 90% cases.

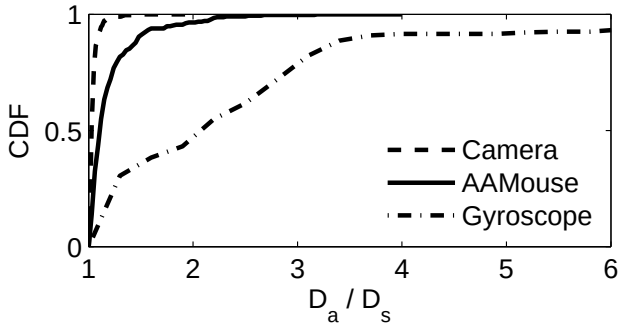
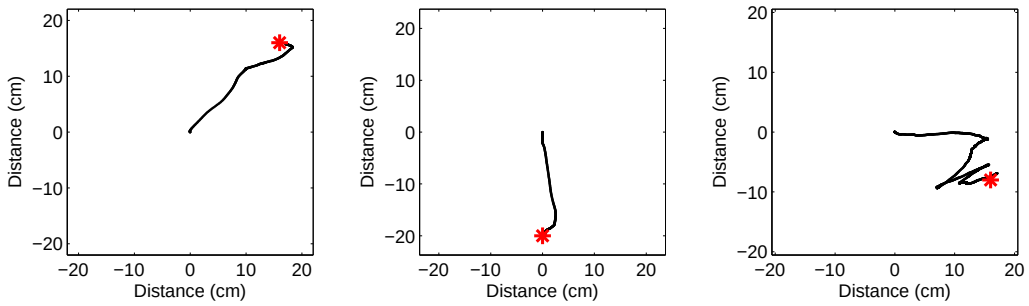


Figure 3.14: CDF of unnecessary movement to reach the target.

Figure 3.14 shows the CDF of $R = D_a/D_s$ for the three schemes. In terms of the median error, camera based tracking and AAMouse yields almost the same performance (*i.e.*, 1.03 versus 1.05), but Gyroscope based tracking yields 1.4, considerably higher than the other two. The performance gap grows even larger in worse cases. The 80th percentile R for the camera, AAMouse and gyroscope based tracking are 1.06, 1.17 and 2.98, respectively. In the case of AAMouse, the pointer travel distance increases mostly due to the tracking error. When the users try to touch the target, they move the device towards the direction of the target. If there is tracking error, the pointer deviates from the direction intended by the user, which extends the moving distance. As it is shown in Figure 3.13 and 3.14, the tracking accuracy of AAMouse is acceptable, so it does not significantly increase R. On the other hand, the moving distance of the gyroscope based tracking increases mainly due to unintuitive use rather than tracking error. According to our observation, the gyroscope itself is quite accurate in measuring the amount of rotation. However, it is not intuitive to users how much they have to rotate their wrists in order to move the pointers in an intended direction, which makes them fail to reach the target. In particular,



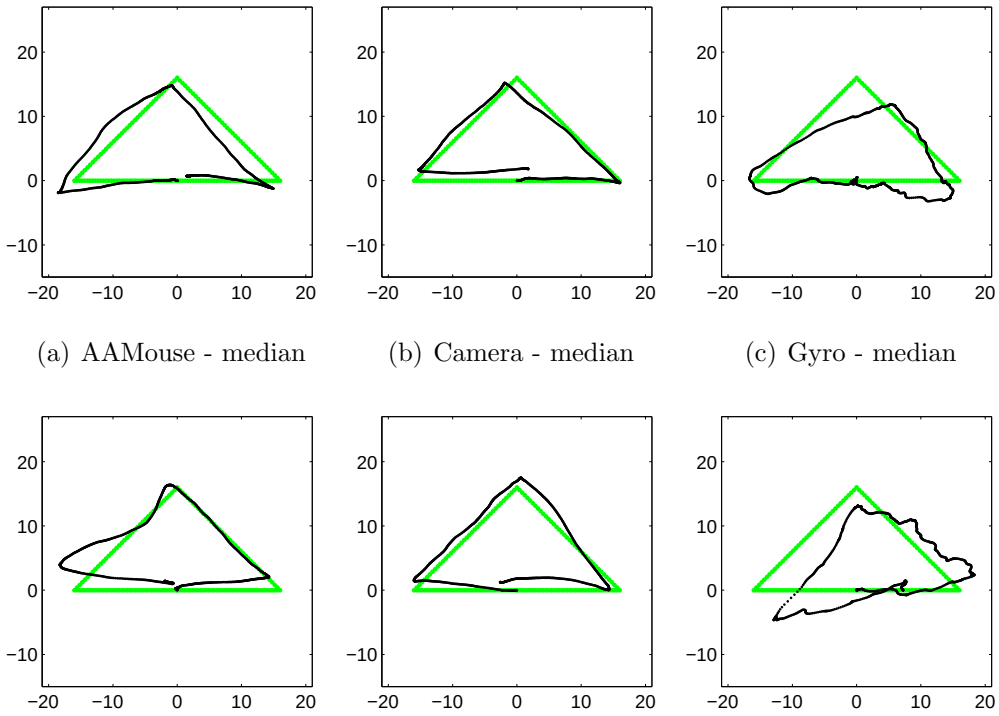
(a) AAMouse. (b) Camera based tracking. (c) Gyroscope based tracking.

Figure 3.15: Sample trajectories.

users have trouble moving a pointer in a diagonal direction, as it requires them to rotate their wrist in two axes simultaneously. The users have to make several attempts before finally touching the pointer, which increases R.

Figure 3.15 shows example trajectories when the user is trying to reach the target under the 80th percentile R. As we can see, AAMouse and camera users spend similar amount of effort to reach the target, but the gyroscope user spends considerable more efforts to reach the target.

Drawing evaluation: Another way of evaluating the usability is to ask a user to draw simple shapes: a triangle, diamond, circle, and heart. The user follows the boundaries of the shapes shown on the screen using the pointer controlled by AAMouse, gyroscope, or camera. Due to the tracking error and imprecise user control, the user cannot draw perfect shapes. We measure the quality of the drawings by calculating the distance between the drawn figure and the original shape. For each point in the figure, we calculate its distance to



(a) AAMouse - median (b) Camera - median (c) Gyro - median

(d) AAMouse - 80% (e) Camera - 80% (f) Gyro - 80%

Figure 3.16: Triangles drawn by the compared schemes.

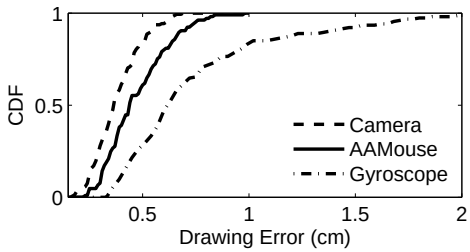
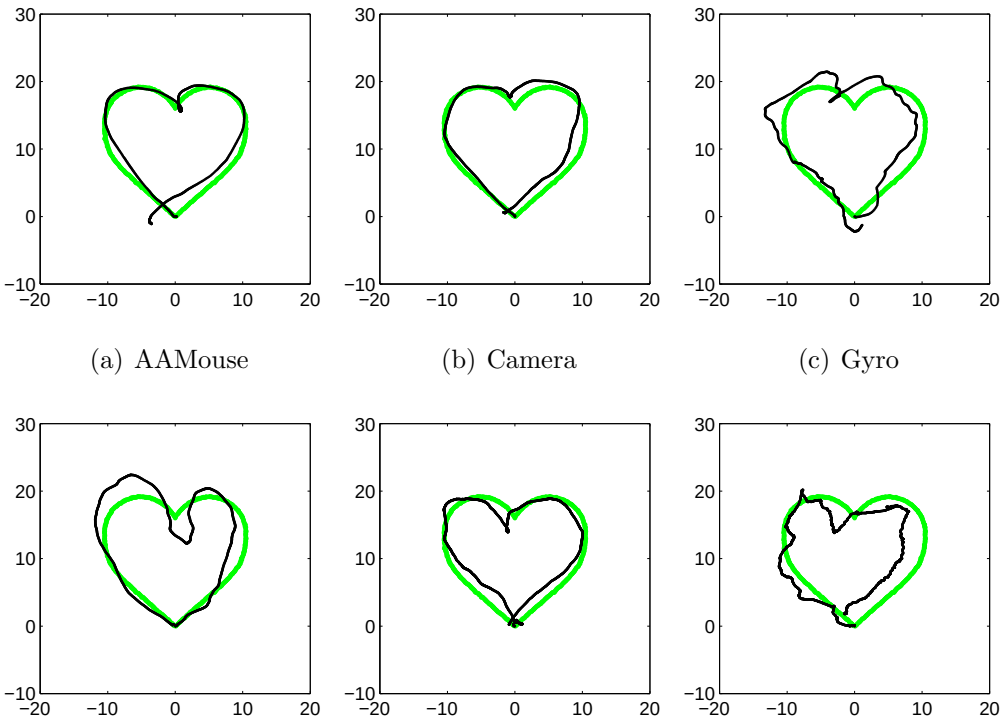


Figure 3.17: CDF of drawing error.

the closest point in the original shape, and average across all points. While this does not perfectly capture the quality of the drawing, it provides reasonable distinction between well-drawn and poorly-drawn figures.

Figure 3.17 shows the CDF of the drawing error. The median drawing errors for camera, AAMouse and gyroscope based tracking are 0.39, 0.47, and 0.61 cm, respectively, and the 80th percentile errors are 0.51, 0.63, and 0.99 cm, respectively. Figures 3.16 and 3.18 show the sample drawings with the corresponding errors. In the interest of space, we omit the results of circle and diamond. The shapes drawn by AAMouse and camera are similar, and do not significantly deviate from the original shapes, whereas the figures from the gyroscope have visibly larger error. In case of larger drawing error shown in Figures 3.16 and 3.18 (d),(e), and (f), the quality difference is even more significant. With the camera, the user is able to draw close to the original shape. The figure from AAMouse is less accurate, but still follows the original shape. On the other hand, the gyroscope based figure contains significant errors, and hardly resembles the original shape.

Impact of the speaker distance: Next we evaluate the impact of the



(a) AAMouse (b) Camera (c) Gyro

(d) AAMouse (e) Camera (f) Gyro

Figure 3.18: Hearts drawn by the compared schemes.

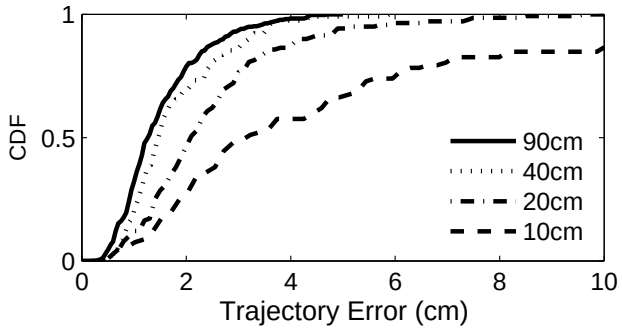


Figure 3.19: CDF of trajectory error with various speaker distances.

speaker distance on the accuracy of AAMouse. In Section 5.3.4, we explain that we track the position by finding the intersection of two circles whose radii are the distances from the speakers. As the distance between the speakers gets closer, the overlap between the two circles increases, which causes a small distance error to result in a larger tracking error. Figure 3.19 shows the CDF of AAMouse trajectory error as we vary the distance between speakers from 10, 20, 40, to 90 cm while keeping the distance between the speakers and mobile device to be around 2 m. Reducing the spacing to 40 cm slightly decreases the accuracy. Reducing the spacing to 20 cm increases the median error from 1.4 cm to 2 cm, which is still acceptable. However, the error under 10 cm spacing is significant. These results suggest when the mobile device is around 2 m from the speakers, AAMouse requires at least 20 cm separation between the two speakers to achieve high accuracy. All smart TVs and most laptops have more than 20 cm distance between speakers, and can support AAMouse well. For example, even a small 13-inch laptop like Apple 13-inch MacBook Pro is 32 cm wide, and its two speakers are at the left and right ends and separated

by around 32 cm [31].

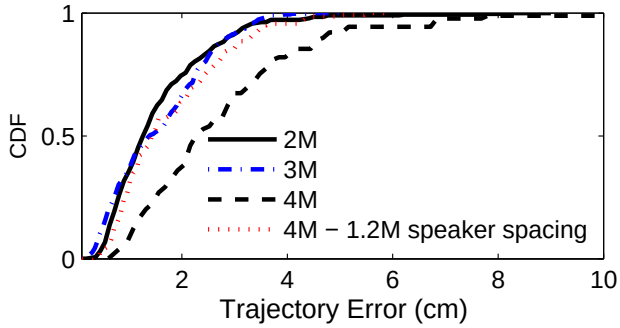


Figure 3.20: Trajectory error with various speaker and microphone distances.

Range experiment: Figure 3.20 shows CDF of the trajectory as we vary the distances between the speaker and the microphone. It shows the accuracy degradation is negligible when we increase the distance from 2 m to 3 m. When the distance increases to 4 m, the degradation increases. The degradation is due to weaker sound signals, and more importantly, reduced relative distance between the speakers versus the distance from the speaker to the mobile device. If the distance from the speaker to the microphone increases while the speaker spacing remains the same, the circles from two speakers have larger overlap, which produces a higher tracking error as we see in Figure 3.19. To confirm this, we extend the speaker spacing from 90 cm to 120 cm. The result shows that increasing the speakers' distance avoids accuracy degradation, and the error is similar to that in 2 m case. As bigger TVs are getting more popular, AAMouse can achieve high accuracy beyond 4 m range. For example, TVs bigger than 60-inch are widely available in market. The distance between the two speakers on such TVs is larger than 130 cm, and can easily support

accurate tracking beyond 4 m.

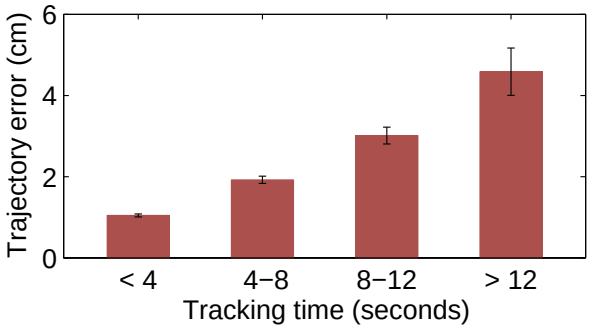
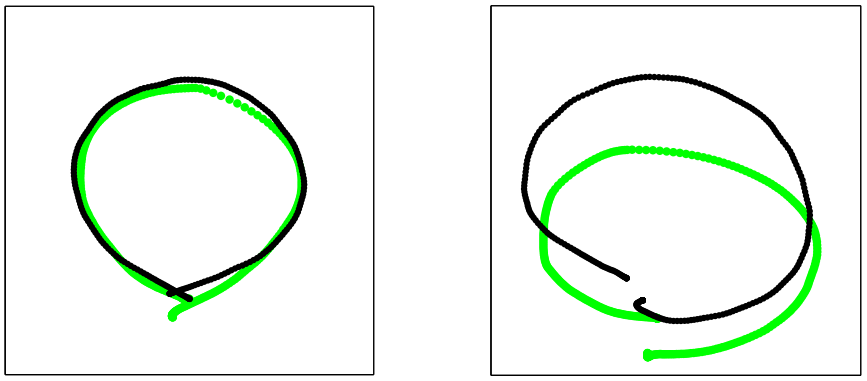


Figure 3.21: Trajectory error with different tracking time.

Impact of the tracking error accumulation: AAMouse tracks the device by estimating the movement from the previous position based on the velocity derived from Doppler shift. The estimation error of the previous position will increase the estimation error of the next position. Therefore, the error tends to increase over time. To evaluate the impact of tracking time on the accuracy, we classify the traces according to the tracking time. In our user study data, the tracking time of each individual experiment depends on the user and the tracking goal (*i.e.*, pointing or drawing), and the minimum, maximum, and mean tracking times are 1.8, 13.5, and 4.9 seconds, respectively. Figure 3.21 shows the mean tracking error for four different ranges of tracking time. As we can see, longer tracking time leads to larger tracking error. When the tracking time is between 8 and 12 seconds, the mean error is 2.9 cm; but when tracking time is longer than 12 seconds, the error increases to 4.5 cm. This suggests that AAMouse is most effective for short-term tracking (*e.g.*, less than 10 seconds). To support short-term tracking, a user can initiate tracking using

a simple gesture or tapping the screen.

Even for longer-term tracking, users are generally not sensitive to the position error slowly accumulated over time. This is especially so for drawing applications, where users are mainly interested in whether the pointer is moving in the way as they intend, and the error in absolute device position is not obvious to the user. To demonstrate it, we ask a user to repeatedly draw circles using AAMouse and track it over 25 seconds. Figure 3.22 shows the circles drawn by AAMouse (black) as well as the ground-truth device movement (green) at the beginning of the tracking and after 20 seconds, where the trajectory errors are 1.2 and 7.9 cm, respectively. As shown in Figure 3.22 (b), after 20 seconds, the absolute position error increases due to error accumulation, but the shape is still preserved.



(a) In the beginning. (b) After 20 seconds.

Figure 3.22: Circles drawn during 20 seconds.

From our experience, we find the user is mainly concerned about in-

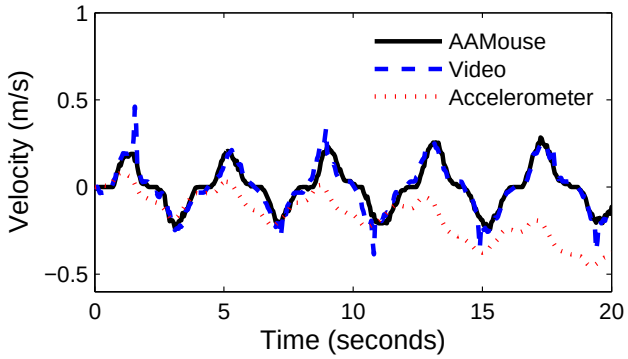


Figure 3.23: Velocity while moving the device backward and forward.

stantaneous tracking error because it moves the pointer differently from what the user intends. This can happen in accelerometer based tracking due to double integration of the acceleration, which causes speed estimation error. In comparison, AAMouse estimates the velocity using the measured Doppler shift every time, so the error of speed estimation does not accumulate over time. To confirm the intuition, we perform the following experiment, where we repeatedly move the mobile device back and forth, and measure the velocity using AAMouse, camera, and accelerometer. As shown in Figure 3.23, the error is accumulated with the accelerometer: the difference between accelerometer and video based velocity estimation increases over time. In comparison, the difference between the video and Doppler based velocity estimation is small and remains constant over time.

Impact of vertical movements: So far, we focus on tracking in a 2-D space by using either 2 speakers or one speaker and one wireless device as anchors. Using more anchors allows us to track in a 3-D space. If we are

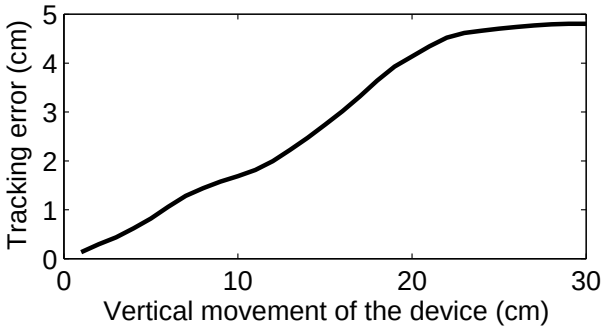


Figure 3.24: Tracking error as the device is moving vertically.

limited to two anchors, we can track the device movement in the horizontal plane (assuming the speakers are placed horizontally next to each other); and vertical movement cannot be tracked. To understand the impact how vertical movements affect the tracking accuracy, we vertically move the device while fixing its position in the horizontal plane. Since the device does not have horizontal movement, any movement tracked is an error. Figure 3.24 shows the tracking error when we vertically move the device up to 30 cm. We repeat 30 times, and get the average tracking distance of AAMouse. Like the other experiments, the distance between 2 speakers is 0.9 m and the distance between the speaker and the device is approximately 2 m. The result shows that the tracking error generated by 30 cm vertical movement is less than 5 cm. With the Doppler shift measurement, AAMouse tracks the device by observing the change in the distance between the two speakers and the device. Given the distance between the speaker and the device, the distance change by the vertical movement is much smaller than that by the horizontal movement. Moreover, as a mouse application, a user is typically aware that vertical movement may

incur tracking error and avoids large vertical movement. If they limit their vertical movement to 10 cm, the tracking error reduces to 1.7 cm. Therefore, the error introduced by vertical movement is likely to be small.

Chapter 4

Acoustic signal based Device-free hand tracking

In this chapter, we develop a novel tracking scheme that can track the movement of the hand while the user is not holding the device.

4.1 Background

Our work is built on the existing Frequency Modulated Continuous Wave (FMCW). Below we give a brief introduction to FMCW, which is a widely used technique in radar [47] to measure the distance to an object.

One way to estimate the distance d is to directly measure the propagation delay τ and use their relationship $d = \tau \cdot v$, where v is the propagation speed of the signal. However, this is challenging because we need large bandwidth in order to send a sharp pulse signal with good time resolution. Instead, FMCW indirectly estimates the propagation delay based on the frequency shift of the chirp signal as follows.

The red curve in Figure 4.1 shows a transmission chirp, whose frequency linearly increases over time. Let f_c, B, and T denote the carrier frequency, bandwidth, and duration of the chirp, respectively. The frequency of the signal

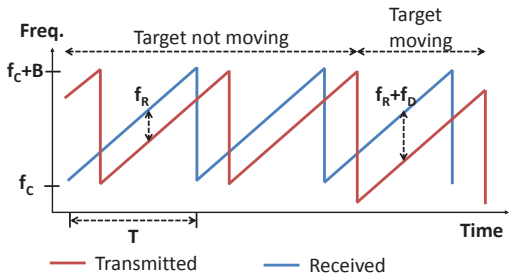


Figure 4.1: Chirp signal in FMCW.

at time t is given by

$$f(t) = f_c + \frac{Bt}{T}.$$

The phase of the signal is calculated by integrating $f(t)$ over time, which is:

$$u(t) = 2\pi(f_c t + B\frac{t^2}{2T}).$$

As a result, the transmitted chirp signal can be represented as $v_{tx}(t) = \cos(u(t))$, where its magnitude is assumed to be one for simplicity.

Consider that the transmitter and receiver are co-located. The transmitted signal is reflected by the target and received with the delay τ_d. Therefore, the received signal at time t is $v_{rx}(t) = v_{tx}(t - \tau_d)$, which is:

$$v_{rx}(t) = \cos(2\pi f_c(t - \tau_d) + \frac{\pi B(t - \tau_d)^2}{T}),$$

where again the magnitude change is ignored. Let R, V_c, V denote the distance between the transceiver and target, the propagation speed, and the target's velocity. Then, the reflected signal is delayed by τ_d:

$$\tau_d = \frac{2(R + Vt)}{V_c}. \tag{4.1.1}$$

The receiver multiplies the transmitted signal with the received signal in the time domain, which is $v_m(t) = v_{tx}(t)v_{rx}(t)$. The mixed signal $v_m(t)$ is called Intermediate Frequency (IF) signal. By using $\cos A \cos B = (\cos(A - B) + \cos(A + B))/2$ and filtering out the high frequency $\cos(A+B)$ component, which has the frequency of the order $2f_c$, $v_m(t)$ becomes:

$$v_m(t) = \cos(2\pi f_c \tau_d + 2\pi t \frac{B\tau_d}{T} + \frac{2\pi B \tau_d^2}{T}).$$

By plugging equation 4.1.1 into the above equation, $v_m(t)$ becomes:

$$v_m(t) = \cos(4\pi f_c \frac{R}{V_c} + 2\pi(\frac{2f_c V}{V_c} + \frac{2RB}{V_c T})t + \frac{2\pi(R + Vt)^2}{Tc^2}). \tag{4.1.2}$$

We ignore the constant phase terms and quadratic term since the former does not change the frequency and the latter is too small to matter. Then we see the frequency of the IF signal f_{IF} is approximated as:

$$f_{IF} = f_R + f_V \tag{4.1.3}$$

$$f_R = \frac{2RB}{V_c T}. \tag{4.1.4}$$

$$f_V = \frac{2f_c V}{V_c} \tag{4.1.5}$$

As we can see, the frequency shift includes (i) the frequency shift that is proportional to the distance to the target f_R, and (ii) the Doppler shift f_V due to the movement of the target. The former depends on the distance and the latter depends on the velocity. These effects also are shown in Figure 4.1, which compares the frequency change with and without movement and shows an additional shift coming from the Doppler shift when the target moves.

4.2 Our Approach

Figure 4.2: Illustration of the device-free hand tracking.

In this section, we present our approach to achieve device-free tracking based on the FMCW. Figure 4.2 illustrates our system. Two speakers and two microphones are used to emit and receive audio signal and track the hand movement. A pair of speaker and microphone is co-located, and serves as an anchor point. Each speaker transmits chirp signals in inaudible and non-overlapping spectrum band with a guard band in between, and the microphone collects the signals from the corresponding spectrum, mixes it with the transmitted signal, uses the peak frequencies to estimate the distance and velocity, which are in turn used to track the hand movement.

As shown in Figure 4.3, we first detects the start of the chirp signal using the cross correlation. Since the chirp signal is periodic, we only need to detect once and can continuously fetch audio samples from the next sampling intervals (100 ms). We also estimate the distance between the speakers and the initial hand's position. Then we continuously fetch audio signals, and perform FFT to detect the peak frequencies. We observe the mixed signal in FMCW has fundamental frequency determined by the parameters of the

chirp sequence. We leverage this property to filter out the reflection from the static objects and detect the reflection caused by the moving hand. Next, we estimate the Doppler shift and use the estimated Doppler shift to select the appropriate FMCW peak for distance estimation. We then incorporate both distance and velocity estimation to continuously track the hand. Below we describe each step in details.

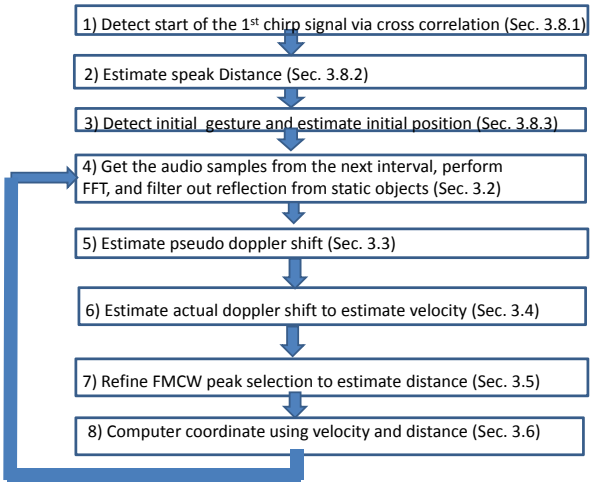


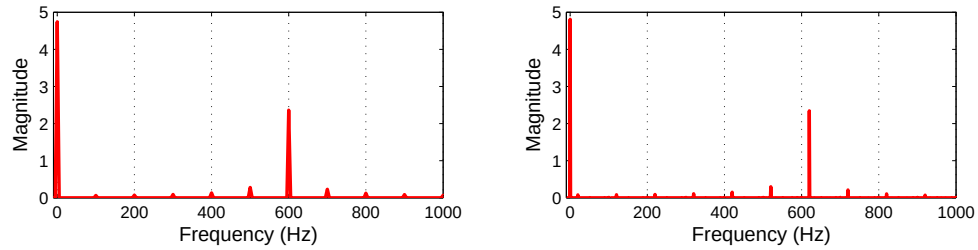
Figure 4.3: Flow chart.

For simplicity, in the remainder of this section, we assume that the chirp signal duration is 0.01 s and the bandwidth is 2KHz, but our scheme can be applied to any duration and any bandwidth.

4.2.1 Fundamental Frequencies in FMCW

Consider a chirp signal (shown in Figure 4.1) propagates over the medium and arrives at the receiver after a delay t_d. As shown in Equation 4.1.3, without movement the range can be estimated based on frequency shift as fol-

low: $R = \frac{f_p cT}{B}$. The transmitted chirp signal has fundamental frequencies that are all multiples of the frequency $\frac{1}{T}$ [41]. In other words, in the frequency domain, it has spectral points with an interval of $\frac{1}{T}$ Hz. For example, when the chirp interval is 0.01 s, it has spectral points every 100 Hz. The received signal $v_r(t)$ is a simple time shifted version of the transmitted signal $v_t(t)$, and has the same period. $v_m(t) = v_r(t)v_t(t)$ has the same period since the periodicity is preserved when two periodic signals with the same period are multiplied [41]. Therefore, the frequency shifts in FMCW exhibit peaks at discrete frequencies, all of which are multiples of $\frac{1}{T}$ Hz without any movement.



(a) Range = 0.5m, Doppler shift = 0 Hz (b) Range = 0.5m, Doppler shift = 20 Hz

Figure 4.4: FMCW signal in simulation.

Figure 4.4 (a) shows the simulation result of FMCW range detection when the transmitter and receiver are both static and 0.5 m apart. The carrier frequency, bandwidth, and the duration of the chirp signal are 17 KHz, 2 KHz, and 0.01 s, respectively. Figure 4.4 (a) is the FFT result of $v_m(t)$. $v_m(t)$ has power every 100 Hz. There are two major peaks at 0 Hz and 600 Hz. The peak at 0 Hz is caused by the self-interference that the transmitted signal is directly received without reflection. We can easily filter out the self-interference by ignoring the peak around 0Hz. Then another peak at 600 Hz is

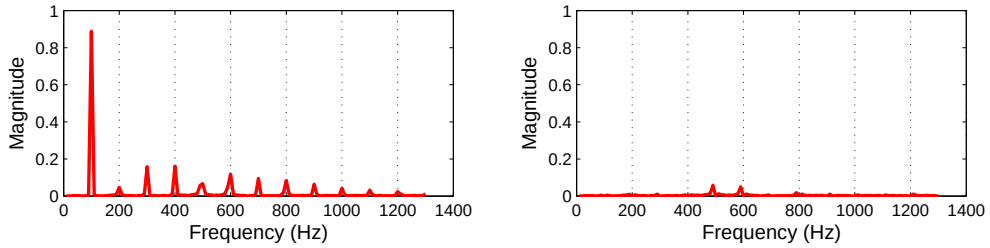
detected and considered to be due to the reflection from the target. Using the Equation 4.1.4, we can find the total distance is 1.02 m, and the distance from the transceiver to the target is half of that: 0.51 m since the total distance includes both forward and reflected paths.

Our initial experiments show that the above FMCW approach yields rough estimation of the hand position but the error is significant: 10 cm, which is too large for many applications. There are several factors that contribute to the large error. First, there are multiple peaks, some of which are due to reflection from other objects. Second, even ignoring the reflection from static objects, the signals can still be reflected by multiple body parts that move together with the hand. These body parts have different distances from the speakers. Moreover, even considering reflection from the same point, the reflected signal may still traverse different paths to reach the microphone. These effects together contribute to multiple FMCW peaks. It is important to select the peak that comes through the direct path from the hand. Third, the frequency change in FMCW partly comes from the propagation delay and partly comes from the Doppler shift. This is shown in Figure 4.4 (b), which plots the simulation result when the range is 0.5 m and the Doppler shift is 20 Hz. In this case, we observe a combined frequency shift at 620 Hz. It is important to decouple the overall shift to the distance-based shift and Doppler shift to achieve high accuracy.

4.2.2 Ignoring Reflection from Static Objects

Reflected signals are challenging to use for tracking because there are possibly many objects around the target that reflect the signal. There are multiple peaks in the FMCW. We should extract the peaks caused by the hand movement.

It is reasonable to assume only the hand is moving and the other objects are static. Based on this assumption, we can ignore all peaks at the spectral points (*i.e.*, integer multiples of the fundamental frequency) since they do not have Doppler shift. Figure 4.5 shows the FMCW signals with and without the peaks in the spectral points while the hand is moving. Ignoring the spectral points, we can clearly observe the shift caused by the moving hand.



(a) Before ignoring the spectral points. (b) After ignoring the spectral points.

Figure 4.5: Before and after removing the peaks at the spectral points.

4.2.3 Estimate Pseudo Doppler Shift

We introduce a notion of the pseudo Doppler shift. It is defined as the difference between the current peak frequency versus the closest spectral point below the current frequency. If the fundamental frequency is 100 Hz and the peak frequency is 620 Hz, then the pseudo Doppler shift is 20 Hz. Similarly,

a peak frequency at 660 Hz has the pseudo Doppler shift of 60 Hz. In this case, the actual Doppler shift may be 60 Hz or -40 Hz. We will describe in Section 4.2.4 how to translate the pseudo Doppler shift to the actual Doppler shift. Note that we define the pseudo Doppler shift as the difference with respect to the spectral point *below* the current frequency for consistency. For example, suppose two peaks at 660 Hz and 540 Hz. To combine the two estimates, we should combine 60 Hz with 40 Hz, instead of -40 Hz with 40 Hz.

As mentioned earlier, there exist multiple peaks even after filtering out the peaks caused by reflection from static objects, because the signal is reflected by different body parts that move together with the hand and they may have difference distances from the speaker. Interestingly, there is a benefit of getting multiple peaks – if some or all of the peaks correspond to the body parts with similar velocities, we essentially get multiple estimates of the Doppler shift and can combine these multiple estimates to enhance the accuracy using Maximum Ratio Combining (MRC) [7].

Specifically, we use the highest peak and multiple peaks around the highest peak to estimate the pseudo Doppler shift as follow.

1. We divide the whole spectrum into multiple spectrum slots that span from $[\frac{n}{T} - \frac{1}{2T}]$ Hz to $[\frac{n}{T} + \frac{1}{2T}]$ Hz, where n is an integer smaller than $B \times T$. For example, when T is 0.01 s, the n-th slot spans $(n \times 100 - 50, n \times 100 + 50)$.

2. We search for the slot with the maximum peak, and denote it as k. We

compute pseudo Doppler shift in the slot k and its 4 nearby slots.

3. We combine the pseudo Doppler shifts in these 5 slots using the following weighted average:

$$f_D = \frac{\sum_{i=k-2}^{i=k+2} \sigma_i f_{d,i}}{\sum_{i=k-2}^{i=k+2} \sigma_i},$$

where σ_i and $f_{d,i}$ denote the peak magnitude and pseudo Doppler shift during the i-th slot, respectively. The highest peak and its nearby peaks are used for the Doppler estimation of the hand, because his/her hand is the closest moving object to the speakers when a user is facing the device to be controlled (a common usage scenario).

As an example, Figure 4.6 shows a snapshot of the received FMCW signal while the hand is moving towards the microphone. The highest peak is in slot 6. We combine the peaks from the slots 4 to 8 to get the final estimate of the pseudo Doppler shift.

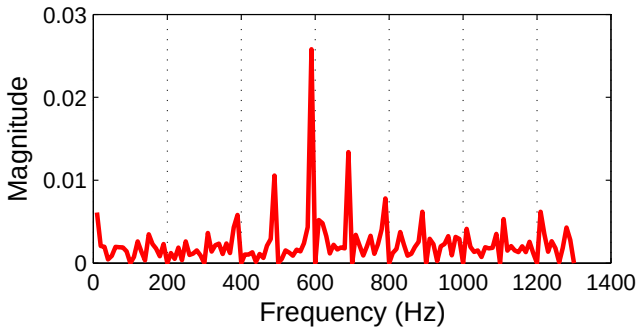


Figure 4.6: FMCW signal while the hand is moving.

4.2.4 Translating Pseudo Doppler Shift to Actual Doppler Shift

As mentioned above, the frequency shift comes partly from the propagation delay and partly from the Doppler shift. To decompose the shift, we observe that the shift caused by the propagation delay should be multiples of the fundamental frequency and the difference between the spectral point and the frequency of the current peak is caused by the Doppler shift. We find the hand movement is typically within 1 m/s, and its corresponding frequency shift should be within 100 Hz. Given the overall frequency change, there are only two choices: reminder(currFreq, FF) or 100-reminder(currFreq,FF). In the previous example, the Doppler shift in the 660 Hz frequency difference can be either reminder(660,100) = 60 Hz or reminder(660,100)-100= -40 Hz.

One approach is to select the Doppler shift with the *smallest magnitude*. For example, one selects -40 Hz instead of 60 Hz when the peak frequency is at 660 Hz. However, this selection may be off since occasionally the hand can generate more than 50 Hz Doppler shift. Figure 4.7 shows an example. At 1.1 second, the actual Doppler shift is 59 Hz, but this scheme (marked in blue) estimates -41 Hz, which results in significant tracking error.

To enhance the accuracy, we use a simple search based on the previous movement (*search*). It tries each of the two choices every slot and picks the combination that minimizes the velocity change over all the slots. We can vary the window size (# slots used to compute the velocity change) to run the exhaustive search. Our results show that a window size of 1, which essentially picks the current Doppler shift that is closest to the previous Doppler shift, is

already good enough. This is because the Doppler shift of the hand movement during the first 0.1-second slot (starting from no movement) is likely to be small: well below 50 Hz and easy to select from the two candidates. Once the Doppler shift of the previous slot is correct, the next one will be correct in the greedy algorithm (assuming minimizing the velocity change leads to the correct selection). In Figure 4.7, our search (marked in red) selects the correct Doppler shift.

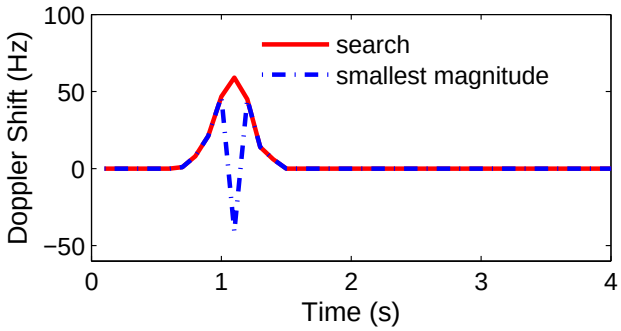


Figure 4.7: The Doppler shift estimation.

Therefore, we use the MRC output from Section 4.2.3 to estimate the pseudo Doppler shift and apply the search to translate the pseudo Doppler shift to the actual Doppler shift.

4.2.5 Refine Range estimation

Due to the multipath, the highest peak may not come through the direct path from the hand. We can use the above Doppler estimate to refine the range estimation. In particular, instead of using the frequency of the highest peak to estimate the range, we find a peak whose distance change is most consistent with the one estimated by the Doppler shift. This is based

on our observation that Doppler estimation in a short interval is more reliable than the range estimation.

Specifically, let v denote the velocity from the current Doppler estimation and t_s denote the sampling interval. Then $v \cdot t_s$ is the distance change during the current interval estimated using the velocity. Meanwhile we can estimate the distance change from the FMCW. For each of the five peaks near the highest peak, we compute the distance change from the previous position (*i.e.*, $|d^i(t) - d(t-1)|$)), where $d(t-1)$ is the previous distance and $d^i(t)$ is the current distance derived using the i-th FMCW peak according to Equation 4.1.3. We select i that minimizes $||d^i(t), d(i-1)| - v \cdot t_s|$ and use $d^i(t)$ as the range in the current interval.

4.2.6 Track Using Estimated Range and Velocity

Given the initial position of the hand that will be explained in Section 4.2.8.3, we can continuously track the hand's movement using the range and the Doppler shift together. One might wonder why we do not track it solely based on the range. This is because using range alone does not provide high accuracy and exploiting the Doppler shift improves the tracking accuracy.

Let $R_k[t]$ and $f_{D,k}[t]$ denote the distance and Doppler shift from the anchor point k at time t, respectively. In addition, we get distance measurement using the velocity from the Doppler shift and previous position (which is derived from the initial position). The Doppler based distance $R_{D,k}[t]$ is as

follow:

$$R_{D,k}[t] = D_k[t-1] + v \cdot t_s = D_k[t-1] + (\frac{f_{D,k}[t]}{f_k} V_c) t_s,$$

where $D_k[t-1]$ is the distance from the k-th speaker at the time slot $t-1$, f_k is the carrier frequency of the speaker, V_c is the propagation speed of the audio signal, and t_s is the FMCW sampling interval.

Using two independent distance measurements $R_k[t]$ and $R_{D,k}[t]$, we find the current position (x, y) by solving the following optimization problem:

$$\min_{x,y} : \alpha \sum_{k=1}^{2} (\sqrt{(x_k - x)^2 + (y_k - y)^2} - R_k[t])^2 + \beta \sum_{k=1}^{2} (\sqrt{(x_k - x)^2 + (y_k - y)^2} - R_{D,k}[t])^2,$$

where (x_k, y_k) is the position of the k-th speaker, and α and β are the constant weighting factors determined by the reliability of the range and Doppler shift estimation results, respectively. In our evaluation, we set $\alpha = 0.5$ and $\beta = 1$ since we find the Doppler shift is more accurate than FMCW.

4.2.7 Exploiting Crossing Paths

When there are two speaker-microphone pairs, there are altogether four paths: $R-H-R$, $L-H-L$, $R-H-L$, and $L-H-R$, where H denotes the hand, R and L are the right speaker/microphone (which are co-located), respectively. $R-H-R$ means the path from the right speaker to the hand to the right microphone. Using only $R-H-R$ and $L-H-L$ already gives us the distance between the hand and the two speakers, which allows us to

track the hand in a 2D space. We can further enhance the accuracy by using the additional two paths $R-H-L$ and $L-H-R$. One way to combine all four paths is to use least square, where we have two unknowns $L-H$ and $R-H$ ($H-R$ and $R-H$ are the same) and four constraints. However, we find these estimates have different accuracy, and simple least square does not help much.

In our setup, the left and the right speakers uses 16 - 18 KHz and 19 - 21 KHz bands, respectively. The quality of the frequency closer to the cut-off frequency (24 KHz) is worse due to the frequency response of the speaker and the microphone [11]. As a result, the estimation from the left speaker is more reliable. Therefore, we combine these estimates in the following way.

Let f_D^R and f_D^L denote the estimated Doppler shift from the left and the right speakers, respectively. f_D^L is more reliable than f_D^R because its frequency is farther from the cut-off frequency. To improve the accuracy of f_D^R, we get an additional estimate $f_D^{L,R} = f_D^L + f_D^R$, which gives $f_D^{R'} = f_D^{R,L} - f_D^L$. We then take an average of f_D^R and $f_D^{R'}$ for the right speaker Doppler estimation. Similarly, we apply the same method to the range estimation from the right speaker. Our evaluation in Section 4.4.2 shows this helps improve the accuracy.

Note that we have also tried other ways of combining measurements from these four paths, including using $L-H-R$ to estimate $H-R$ distance by canceling out $1/2L-H-L$ and using $L-H-R$, $R-H-R$ and $R-H-L$ together to estimate $H-R$. We find the combination mentioned above yields the best performance due to the more accurate estimation using the left

speaker.

4.2.8 Practical Issues

In this section, we describe three major practical issues: (i) when to start performing FMCW, (ii) how to estimate the distance between the speakers, and (iii) how to estimate the initial hand position.

4.2.8.1 Synchronization

In order to mix the transmitted and the received signal, the tracking program should know the exact start time of the chirp signal transmission. This is challenging even if the audio signal transmission and the reception are implemented in the same program and running on the same machine, due to a random delay between the call of the audio play function and the play time of the audio file. To avoid the uncertainty, we detect the beginning of the chirp signal using cross-correlation between the received audio signal and the original chirp signal. As the speaker and the microphone are co-located, we can consider that the signal is received as soon as it is transmitted. By finding the peak in the cross-correlation output, we can detect the start of the chirp sequence in the received signal, which is also the start of the transmitted chirp signal. Figure 4.8 shows one snapshot of the cross-correlation result. It shows the peak every 480 samples, which corresponds to the chirp duration.

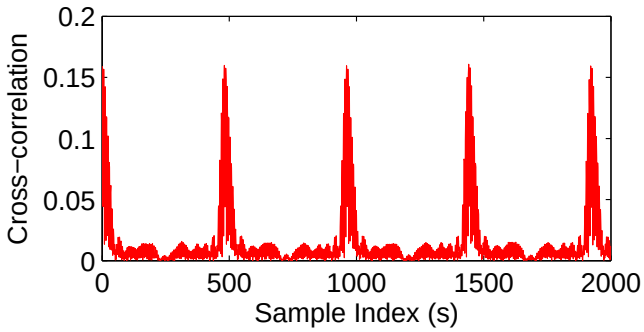


Figure 4.8: The cross-correlation result.

4.2.8.2 Finding the Distance between Speakers

Our tracking needs to know the distance between the two speakers in advance. One can either measure the distance between the speakers using a rule, or use a smartphone based calibration introduced in [73]. Below we introduce an even simpler procedure without any user intervention if the microphones are co-located with the speakers. Suppose we have synchronized the transmitted and received signals using the above procedure. We can measure the distance between from the left speaker to the right microphone using the FMCW signal transmitted from the left speaker based on Equation (4.1.4). Likewise, we can measure the same distance from the right speaker to the left microphone. To improve the accuracy, we average the distance measured from both directions. We can further measure the distances for a longer duration and the get the average, since the speakers are static. Section 4.4.1 presents the performance evaluation result on the speaker distance estimation.

4.2.8.3 Finding Initial Position

A user initializes tracking by performing an initial gesture. Our evaluation uses grabbing gesture (*i.e.*, closing and opening a hand twice) as the initial gesture. We leverage the existing work on gesture recognition to detect the initial gesture (*e.g.*, [17, 39]). So below we focus on determining the initial hand's position while it is performing the initial gesture.

When the hand is performing the initial gesture, we perform FFT to determine the peak frequencies in the mixed signals. Then we filter out the frequencies at the spectral points to remove reflection from the static objects as before. Next, we compute the difference between each of the remaining peaks and their closest spectral points (integer multiples of the fundamental frequency). We estimate the distance using Equation 4.1.4. This procedure works because the initial gesture has slow speed, well below 50 Hz.

When the number of speakers and the number of dimensions are both equal to 2, the hand's position is estimated as the intersection of all the circles centered at each of the speakers with the radius derived from $R = \frac{\Delta f V_c T}{2B}$. Similarly, if both the number of speakers and dimensions are 3, the solution is the intersection of the spheres centered at these speakers with the corresponding radii. If we have more speakers than the number of dimensions (*i.e.*, we have more constraints than the unknowns), we can localize the hand by solving the following simple optimization:

$$min : \sum_i |dist(P - A_i) - dist_i|^2,$$

where P is the hand's coordinate, A_i is the i-th speaker's coordinate, and $dist_i$ is the distance between the ith speaker and the hand estimated using FMCW. Essentially, this objective minimizes the total error with respect to all the FMCW distance measurements from the speakers.

4.2.9 Existing FMCW Approaches

FMCW is a widely used technique in radar detection and there are existing approaches to track the moving target using FMCW [58, 68, 30, 33]. Moving Target Indication (MTI) [58] and Moving Target Doppler (MTD) [68] are the most widely known approaches to track a moving target. These techniques target RF signal, which significantly different from the acoustic signal. RF signal propagates approximately
1,000,000 times faster than the audio signal. It is less susceptible to the multipath fading, but requires large bandwidth (*e.g.*, more than 1 GHz) to detect the target in fine resolution. It also requires high sampling rate, which makes it challenging to implement FMCW tracking on RF signals completely in software. Because both MTI and MTD are designed for RF based radar, they are not suitable for acoustic signal based tracking. To the best of our knowledge, our scheme is the first device-free tracking system based on FMCW using acoustic signals.

MTI tracks the moving target by comparing the difference between the two consecutive chirp signals [58]. Let $c_1(t)$ and $c_2(t)$ be the two chirp signals received in a row, respectively. $d(t)$ is the difference between the two chirps

(*i.e.*, $d(t) = c_2(t) - c_1(t)$). If there is no moving target, $c_1(t)$ and $c_2(t)$ are the same in the ideal case. Then $d(t)$ is be zero. If the target is moving, the signal change will be captured in $d(t)$. MTI performs FMCW processing to $d(t)$ and finds the peak that corresponds the distance of the moving target. Instead of finding both velocity and distance of the moving target, it only estimates the distance, which is not accurate in acoustic signal based tracking.

MTD is closer to our approach. Given the series of the received chirp signals, it performs 2D-FFT and separately estimates the Doppler and the range [68]. Suppose that the received signals are N consecutive chirp sequences whose length are k. It reorganize the received signal to $N \times k$ dimensional matrix, performs 2D-FFT, and finds the peak position in the FFT output. Let $S^{i,j}$ denote the output of the 2D-FFT in the ith row and jth column whose magnitude is the maximum. Then i and j are the frequency indexes by the range and the Doppler shift, respectively. Scaling is required depending on the resolution of the FFT. The idea behind the Doppler-range separation is that the phase change of the signal associated to the target Doppler shift is proportional to the chirp number. Observing one chirp does not reveal it, but performing 2D-FFT on a series of the chirp signals can detect it [30].

We apply MTI and MTD to the acoustic signal, but find they are inaccurate to track the hand movement. MTI hardly follows the movement, and MTD performs better but still significantly under-performs our scheme, as shown in Section 4.4.2. The main reason is because these schemes target ideal scenarios without multipath and multiple reflections. In real environments, it

is common to have noise, reflection from multiple objects, and multipath from the same reflection point. Our scheme improves the robustness by (i) using multiple estimates to improve the Doppler estimate and (ii) using the Doppler estimate to improve the FMCW peak selection.

4.3 Implementation

For the performance evaluation and the user study, we implement a tracking program in JAVA that processes the audio signal and tracks the movement of the user's hand in real-time. It runs on a Windows machine. We use external speakers and microphones as transmitters and receivers. To record the audio signal from the left and right channels separately, we use 2 microphones and connect them into one stereo input port using RCA-to-3.5 mm jack adapter. For FMCW, we generate chirp signals in MATLAB and store it as WAV format so that it can be played using JAVA library. The sampling rate of the audio file is 48 KHz, which allows to use up to 24KHz audio band. The bandwidth and the interval of the chirp signal are 2 KHz and 10ms, respectively. The left and right speakers use 16 - 18 KHz and 19 - 21 KHz spectrum, respectively.

The sampling interval is 100 ms. Every interval the tracking program detects the start of the chirp signal, reads 4800 audio samples, mixes the received signal with the transmitted chirp signal, performs FFT, estimates the range and the Doppler, and tracks the movement. The complexity of the mixing signals and performing FFT are $O(N)$ and $O(NlogN)$, respectively.

Considering that N is 4800, they cause negligible computation complexity. The complexity of the Doppler and range estimation are also very low, and the main overhead of our tracking system is finding the position of the hand by solving the optimization problem in Section 4.2.6. The total time to find the position in each sampling interval takes 1 ms on a PC with Intel i7 2.4 GHz processor.

4.4 Performance Evaluation

For the performance evaluation and the user study, we run our program on DELL XPS desktop with Intel i7 CPU and 8GB memory. Logitech S120 2.0 Speaker and Soonhua SH-666 Microphone are used as a transmitter and a receiver, respectively. The speaker and the microphone volumes are set to 50 out of 100 to make sure it works in normal volume range. Each pair of the speaker and the microphone are located as close as possible. By default, the distance between 2 speakers are 0.6 m, and the distance between the speaker and the hand are approximately 0.7 m.

For the performance evaluation, the user's hand movement is also tracked by the external camera that provides ground-truth of the movement. We put a blue label on the hand to ease the camera tracking. We implement a simple vision based object tracking program that tracks the movement of the hand in real-time using OpenCV. In general, computer vision based tracking is expensive, and requires visually distinct pattern and good lighting condition, which may not hold in general. We quantify the trajectory error by comparing the

ground-truth hand position with our estimated position every sampling interval (*i.e.*, 100 ms). Note that the trajectory error does not measure the absolute location error so the difference between the estimated initial position and the ground-truth initial position is ignored. In addition, we perform user study, where we show a target on the screen and ask the user to touch it by moving the cursor controlled by the hand movement.

4.4.1 Micro benchmark

We first present micro benchmark results. In particular, we evaluate the accuracy of measuring the speaker distance and the initial position.

Distance between speakers: Figure 4.9 shows the estimated distance between speakers during various measurement periods, where the exact speaker distance is 0.6 m. We get one distance estimation result every sampling interval, and the estimation for the longer duration is an average over all of the measurements during the interval. The result shows that the estimation converges within the error range 1 cm in 7 seconds. 1 cm error in the speaker distance estimation hardly affects the tracking [73]. Note that this calibration is required only when the speaker location is changed.

Finding the initial location: We examine the accuracy of estimating the initial location. The user initializes the tracking by performing grabbing gesture twice. By finding the peak frequency in the FMCW signal, we estimate the distance from the two speakers, and localize the hand by finding the intersections of two circles whose centers are at the two speakers. Figure 4.10

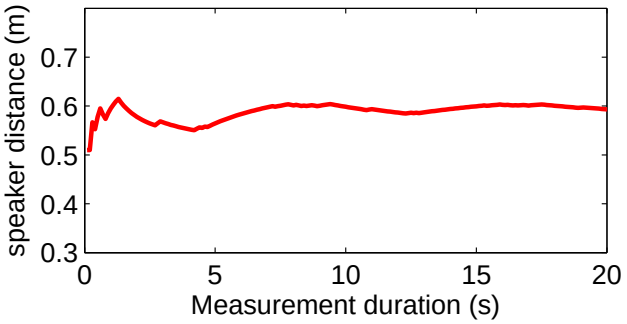


Figure 4.9: Speaker distance estimation over time.

shows the CDF of the initial location estimation error. The median error is 4.3 cm, and the 90th percentile is 8.2 cm.

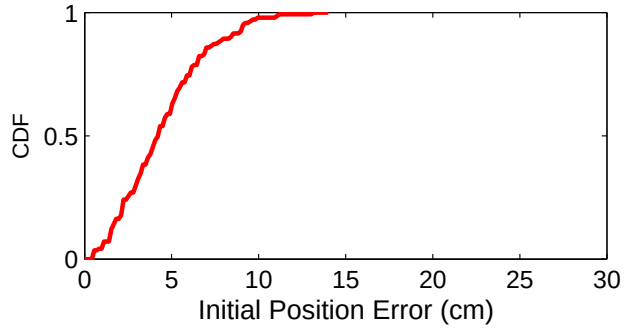


Figure 4.10: CDF of the initial position error.

To further understand the impact of the error in the initial position estimation, Figure 4.11 plots the median trajectory error as we inject varying amount of errors to the known initial position, where the error bars represent 90% confidence interval. The median trajectory errors under the initial position errors of 0, 5, and 10 cm are 1.86, 1.93, and 2.06 cm, respectively. The median error of the initial position is 5cm, which corresponds to 1.93 cm tracking error. Therefore, the initial position error does not significantly degrade the tracking accuracy in our system.

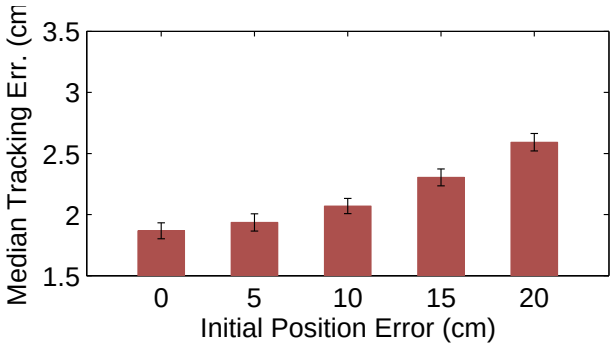


Figure 4.11: Impact of the initial position error on the tracking accuracy.

4.4.2 Tracking evaluation

In this section, we evaluate the tracking accuracy of our scheme with various parameters and settings, and compare it with MTD [68], an existing FMCW-based approach for the range and Doppler separation. Unless otherwise specified, we use multiple peaks to estimate the Doppler shift, combine them using MRC, and use both range and Doppler estimates to track the position with the corresponding weights 0.5 and 1, respectively.

Performance comparison with MTD: We compare the performance of our scheme with Movement Tracking Doppler (MTD), an existing FMCW-based approach to track a moving object. We carefully follow [68] to get the Doppler and range estimates. After getting these estimates, we use the same optimization in Section 4.2.6 to track the hand movement.

Figure 4.12 compares the CDF of the trajectory errors of the two schemes. The accuracy of MTD is much worse than our scheme. Its median and 90th percentile of the errors are 8.5 and 15.8 cm, respectively. In comparison, the corresponding numbers of our scheme are 1.94 and 3.3 cm,

respectively. In general, MTD roughly tracks the direction, but is vulnerable to noise and sometimes gets incorrect Doppler estimates, because it simply detects the Doppler and range by finding the highest peak in 2D-FFT output. This does not work well in practice due to multipath and noise. In comparison, we use the MRC based combining to enhance the accuracy of the Doppler estimate and applies the improved Doppler estimate to refining the FMCW peak selection, both of which significantly improve robustness.

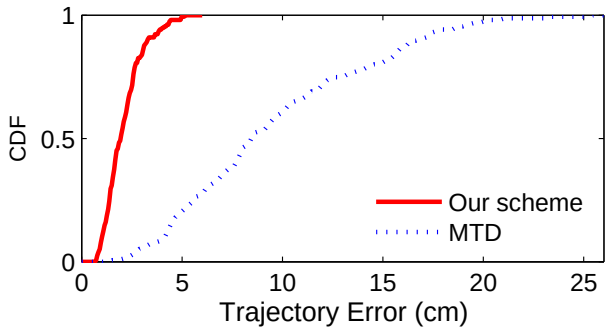


Figure 4.12: Tracking error comparison with MTD.

Using multiple peaks in Doppler estimation: As we explained in Section 4.2, the hand movement causes multiple peaks in the FMCW signal, and how to use them for the Doppler estimation significantly affects the tracking accuracy. We propose combining the estimates from the 5 peaks next to the highest peak. Figure 4.13 shows the CDF of the trajectory error where the Doppler shift is estimated using (i) one highest peak, (ii) 5 peaks next to the highest peak, and (iii) all peaks higher than a threshold (0.008).

(i) yields the median trajectory error of 8.76 cm. Since the reflected signal from the hand is weak, it is easily affected by the noise, which makes the

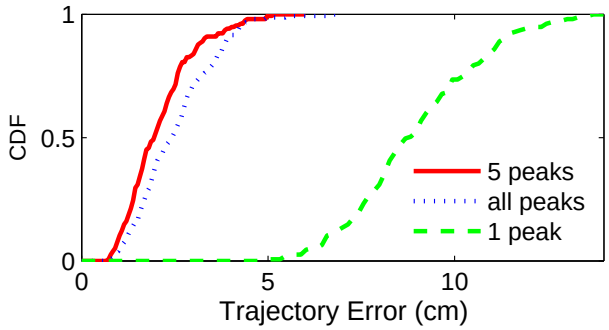


Figure 4.13: Tracking error with various number of peaks for Doppler.

Doppler estimation only relying on one peak very unreliable. In comparison, (ii) achieves 1.94 cm tracking error due to the use of multiple peaks. (iii) uses all peaks higher than the threshold, and its median trajectory error is 2.40 cm, which is 23% higher than the proposed estimation. This is because the peaks are caused not only by the hand but also by the movement of the other parts of the body, such as arm, elbow, and chest, which have different velocities from the hand and causes incorrect Doppler estimation. By limiting the number of observing peaks to 5, we can effectively filter out the Doppler other than the hand and improve the tracking accuracy.

Impact of MRC: Given the Doppler shift estimations from the five peaks, we perform maximal ratio combining so that the more reliable measurement gets the higher weight. Figure 4.14 shows the CDF of the median trajectory error, where the weights are set according to the amplitude of the peaks (MRC weights) or the same (same weights). The median trajectory errors with the MRC weights and same weights are 1.94 cm and 2.34 cm, respectively. Assuming the noise is constant across the spectrum, the peak with higher magnitude

is less susceptible to noise, so it is more reliable. By giving more weights to them, we improve the accuracy of the Doppler shift estimate and reduce tracking error.

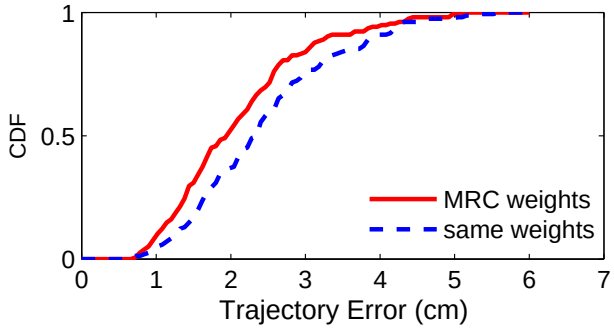


Figure 4.14: Tracking with and without MRC.

Using more paths: Besides the two Doppler measurements estimated from two speaker-microphone pairs, we can get additional Doppler estimation from the intersecting paths (*e.g.*, from the left speaker to the right microphone) and use it to improve the Doppler estimation accuracy, as explained in Section 4.2.7. Figure 4.15 shows the impact of using Doppler estimation from the crossing path. The median trajectory error is reduced from 2.30 cm to 1.94 cm, which is 18% improvement.

Impact of weight on the range: Using the Doppler and range estimation from the FMCW signal, we find the position of the hand by solving the optimization problem in Section 4.2. Here, the parameters we can control are α and β that determine the weights of the Doppler and the range estimations. This section examines the impact of the weights on tracking accuracy. For simplicity, we fix β to 1 and vary α since only their relative ratio matters.

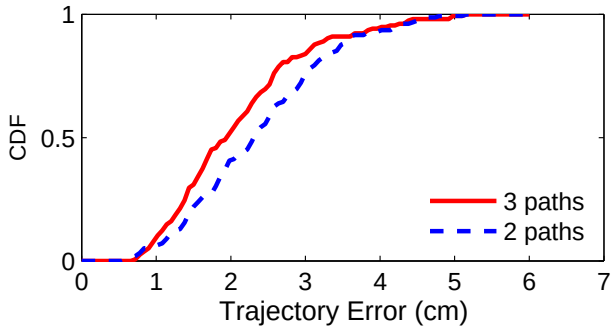


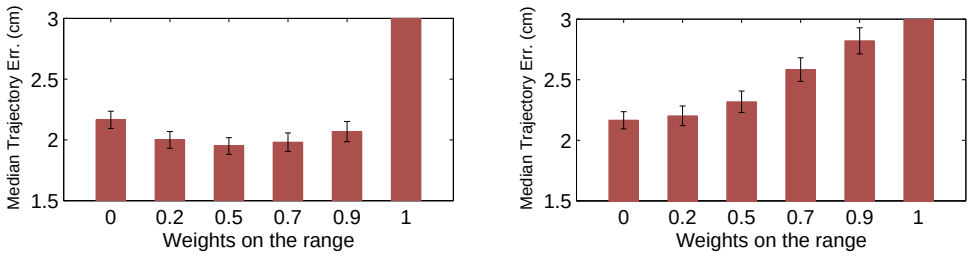
Figure 4.15: Tracking error with different numbers of paths.

Figure 4.16(a) shows the median trajectory errors as α varies from 0 to 1. The result shows that a proper weight improves the accuracy. When α is 0.5, the tracking error reduces by 10% compared to the case α is 0. In general, the range estimation is less reliable than the Doppler estimation, so giving too high weight can reduce the tracking accuracy. If the weights for the Doppler and range are equal, the tracking accuracy significantly degrades: the median error increases to 6.97 cm.

Figure 4.16(b) further shows the tracking error when the highest peak is selected for the range estimation instead of the algorithm in Section 4.2.5. As we can see, the best tracking error increases from 1.94 cm to 2.32 cm when α is 0.5. This demonstrates the effectiveness of our refinement algorithm.

4.4.3 User Study

To evaluate the usability of our scheme as a user interface to point a target by the hand movement, we design a user study that shows a target on the screen and ask the user to touch it by moving the cursor controlled



(a) Range is refined (b) Range is not refined

Figure 4.16: Tracking error with various weights on the range estimation.

by the hand movement. The target is located at random positions with 30 cm away from the origin. Whenever the user conduct a new experiment, the cursor position is reset to the origin and the user touches target by moving his hand at least 30 cm. We evaluate the usability by measuring the travel distance of the cursor. We use $R = D_a/D_s$ as the metric, where D_a and D_s are the actual distance traveled and the shortest path distance, respectively. If the tracking is accurate and the user is comfortable controlling the cursor, the travel distance will be close to the shortest path distance to the target and R should be close to 1. For the user study, we recruit 5 users and collect data. Users had 5 minutes training time before collecting the data.

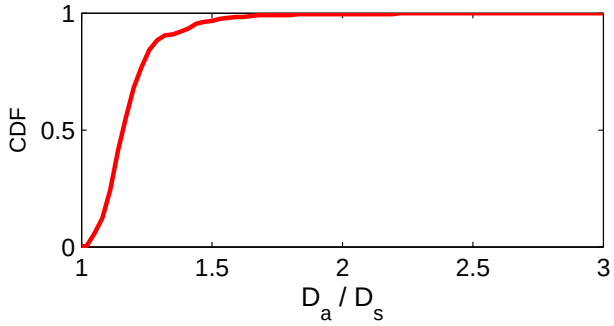


Figure 4.17: CDF of R in our user study.

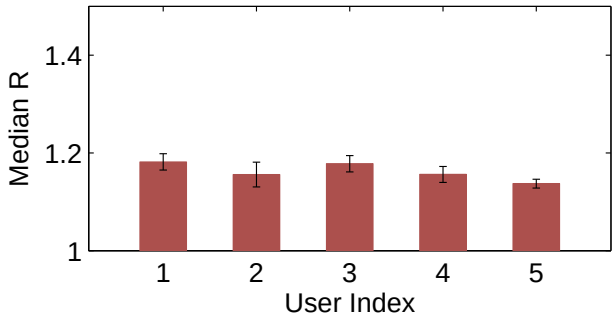


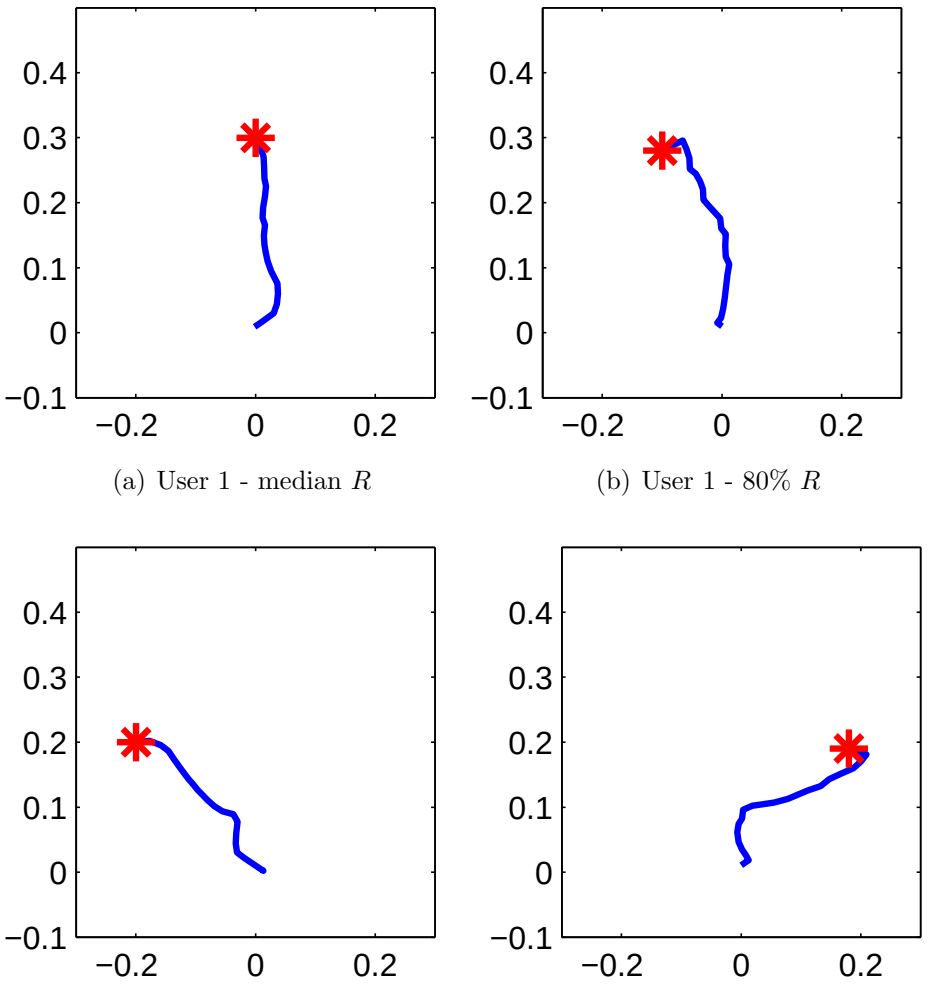
Figure 4.18: Median R in our user study.

Figure 4.17 shows the CDF of R from all of the users' experiments. The median and the 80th percentile of R are 1.15 and 1.23, respectively. These results show the users can move the cursor to the desired position without much overhead. Figure 4.18 is the median R of the five users. We do not see significant difference among the users. The lowest and highest median R in this result is 1.13 (user 5) and 1.18 (user 1), respectively.

Figure 4.19 shows the sample trajectories of the cursor controlled by the user's hand movement, which correspond the median and the 80th percentile of R. Although the trajectory is not a straight line due to the tracking error, the user can compensate the error and move the cursor towards the desired direction. Even in the case of the 80th percentile R, the length of the trajectory is still similar and the user can still easily point to the target.

4.4.4 Mobile Experiment

We further use Samsung Galaxy S4 to track a nearby finger movement. The hand is 30 cm away from the mobile initially. We track the trajectory



(a) User 1 - median R

(b) User 1 - 80% R

(c) User 3 - median R

(d) User 3 - 80% R

Figure 4.19: Cursor trajectory samples.

while the user is moving his finger 10 cm in a random direction. We get the ground-truth by letting the user move his finger on the top of a smart tablet, which collects the touch event on the screen and generates the ground-truth trajectory.

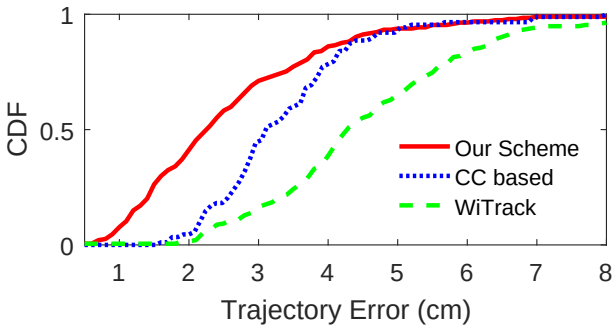
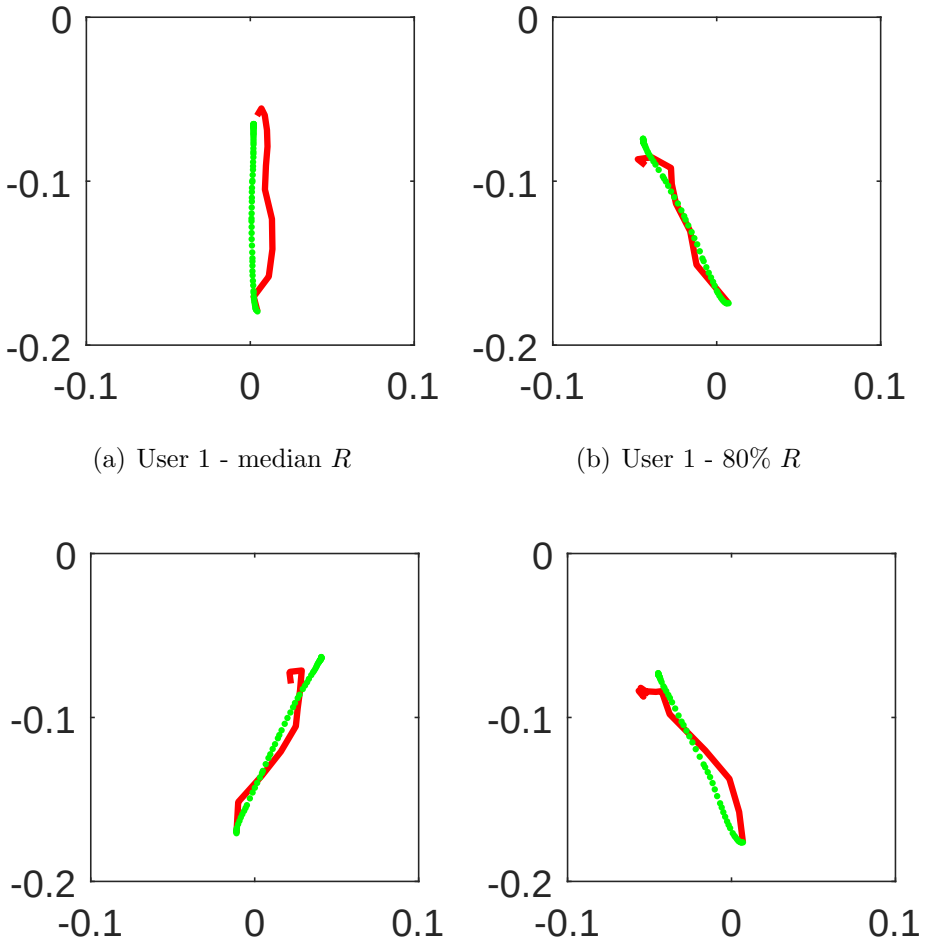


Figure 4.20: Tracking error with mobile phone.

Figure 4.20 shows the CDF of the trajectory errors of our scheme and compares it with with WiTrack [4] and CC-based tracking. The median errors of our scheme, CC-based tracking, and WiTrack are 2.2 cm, 3.1 cm, and 4.3 cm, respectively. The accuracy of our scheme does not significantly decrease on the mobile. So it is feasible to use as a UI for wearable devices, such as smart watch and VR headset. We closely follow [37] to implement CC-based tracking and use larger bandwidth and increased interval to enhance its accuracy. But its accuracy is still lower than reported in [37], likely due to undisclosed details in [37], such as parameter tuning and calibration. WiTrack targets wideband RF signals, and directly applying it to acoustic channel with limited bandwidth yields limited accuracy. Our approach out-performs both due to using the velocity to refine FMCW peak selection and using both velocity and distance for tracking.

Figure 4.21 shows example trajectories tracked by our scheme under median errors, where the green and red lines are the ground-truth and the estimated trajectories, respectively. The two closely follow each other.



(a) User 1 - median R

(b) User 1 - 80% R

(c) User 3 - median R

(d) User 3 - 80% R

Figure 4.21: Example tracked trajectories corresponding to the median error.

Chapter 5

Acoustic signal based Device-free Finger tracking

In this chapter, we develop Acoustic Phase Tracking (APT), a fine-grained device-free tracking scheme that can track the finger movement by observing the phase change of the reflected acoustic signal in close proximity.

5.1 System Overview

Fundamentally, acoustic signal is transmitted as waveform of which the phase changes as much as 2π whenever it travels the distance of wavelength. In inaudible frequency range, the wavelength is very short (*e.g.*, 1.9 cm in 18KHz audio frequency), so observing the phase allows us to track the distance change of the target in fine resolution. For example, if the target object moves 1 mm far away from the sender and the receiver, the travel distance of the reflected signal increases 2 mm, which causes the phase change of 0.66π.

However, it is challenging to observe the phase change of the acoustic signal reflected from the moving finger. The received signal from the microphone includes not only the signal reflected from the finger but also the directly received signal and the reflected signals from many of the nearby ob-

jects. Therefore, it is difficult to distinguish the phase change of the signal from the moving target from those incurred by the movement of the surrounding objects. Considering that this technique will be applied to the smart watch and the VR headset user interface design, the movement of nearby people and objects is unavoidable. So without proper mechanism to filter out the noise from surroundings, it will be infeasible to use it in practice. In order to address this challenge, we observe the phase from the estimated channel impulse response (CIR) rather than the received signal itself. CIR is a characterization of the all of the signal traversal paths with different delays and the magnitudes [49]. Specifically, it is a vector of channel tap values where each channel tap corresponds to multi-path effects within specific delay range. By focusing on the phase change of the certain channel taps whose corresponding delays are within the target delay range, we can effectively filter out the phase change incurred by the environmental changes.

To observe CIRs in acoustic channel, we design our own data communication channel. In RF based wireless channel where standard communication protocols and channel estimation techniques already exist, it is easy to access the channel coefficients in devices (*e.g.*, CSI in Intel 802.11n network interface cards). On the other hand, there is no pre-defined communication standard in acoustic channel so we design our own communication system and estimate the channel from it. We design it so that the transmission and the reception of the signal can be processed in real-time using moderate capability smartphones. The speaker continuously transmits the training sequences with constant in-

terval, and the microphone receives the signal and the tracking app processes it and tracks the phase change and eventually the distance change. To track the finger movement in 2D space, we consider a smartphone with at least one speaker and two microphones such as recent Samsung Galaxy S phones. Using the distance changes from two different microphones, we can eventually track the movement of the finger in 2D space.

Another important consideration in our acoustic channel design is sending the signal in inaudible frequency band. We only use the frequency higher than 18 KHz that is considered to be inaudible to people of most ages [50]. Also, we use the frequency lower than 22KHz because some smartphones support up to 22.05 KHz (*i.e.*, 44.1 KHz sampling rate). As a result, 4 KHz band between 18 KHz and 22 KHz is selected as the communication band. We design the channel according to the baseband equivalent model where the signal is first generated in baseband and delivered through passband by up-converting and down-converting with the center frequency of 20 KHz. In the remainder of the section, we consider the sampling rate of the signal as 48 KHz because Samsung Galaxy S4 phone that we use for testbed support up to 48 KHz, but this technique can be directly applied to the phone with the maximum sampling rate of 44.1 KHz.

The rest of the chapter is organized as follows. We first introduce the design of the acoustic data communication channel in Section 5.2. Section 5.3 describes how to observe the phase change for CIRs and track the finger movement in 2D space, and Section 5.4 addresses the practical issues such as frame

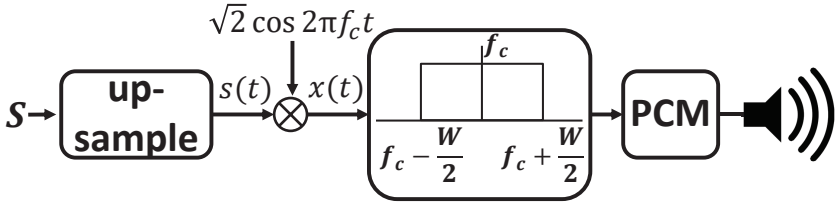
detection and channel estimation. We evaluate its performance in Section 5.5.

5.2 Designing Acoustic Data Communication Channel

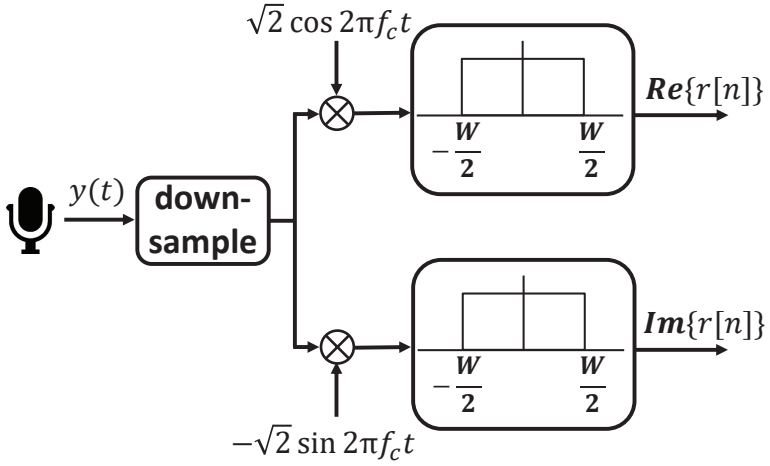
Defining our own data communication channel, one design choice we have is whether it will be single-carrier based or multi-carrier (*i.e.*, OFDM) based communication. One of the challenges of wireless communication is Inter-Symbol-Interference (ISI) where the symbol information is dispersed to nearby symbols. OFDM is very effective in avoiding ISI so it is widely used in most of the modern wireless communication techniques. However, it yields the channel estimation output in frequency domain, while channel information is often more useful in time domain for the purpose of tracking and localization [32]. So a transformation of the channel coefficients from frequency domain to time domain is needed [54], but it requires additional complexity and significant amount of noise might be injected during the process [70]. Therefore, we decided to design single-carrier based communication system to get the channels in time domain without extra processing.

5.2.1 Transmitted Signal Design

In order to estimate the channel, we need to send a known training sequence and observe the change of the received signal incurred by multi-path. Let $\mathbf{S} = \{s[1], ..., s[K]\}$ be the transmitted training sequence, where K is the length of the sequence. It can be any random bits, but we choose 26-bit GSM training sequence because it is known to have good properties for the purpose



(a) Transmitter design



(b) Receiver design

Figure 5.1: Transmitter and Receiver system diagram.

of synchronization and channel estimation [45] and GSM has been the most widely used single carrier based communication protocol. We modulate **S** to BPSK symbols, where bits 0 and 1 are mapped to baseband symbols 1 and -1, respectively. To transmit the modulated symbols over the inaudible frequency band, we first need to reduce the bandwidth of the signal so that it does not exceed the maximum allowed bandwidth of the inaudible band. Let f_s and B be the sampling rate and the bandwidth of the channel, respectively. In order

to limit the bandwidth of the transmitted symbols, we upsample the symbol with the rate of $\frac{f_s}{B}$, which is done by zero padding and low-pass filtering to smoothen the discontinuity [40]. Finally, we up-convert the signal to transmit it over the inaudible band. Let f_c be the center frequency of the passband. We change the frequency of the signal by multiplying $\sqrt{2}\cos(2\pi f_c t)$ to the baseband signal:

$$x(t) = \sqrt{2}\cos(2\pi f_c t)s(t),$$

where $s(t)$ and $x(t)$ are upsampled baseband signal and passband signal, respectively. In traditional digital communication with quadrature amplitude modulation, the real parts of the baseband signals (*i.e.*, in-phase component) are multiplied by $\sqrt{2}\cos(2\pi f_c t)$ while the imaginary parts (*i.e.*, quadrature component) are multiplied by $-\sqrt{2}\sin(2\pi f_c t)$. But in our case that uses BPSK-modulated symbols, all of the imaginary parts the symbols are zeros, so the latter process is unnecessary. $x(t)$ can be also represented as

$$x(t) = \sqrt{2}e^{-j2\pi f_c t}s(t).$$

To remove the noise outside of the transmission band, we perform band-pass filtering to $x(t)$ with pass-band range from $f_c - \frac{B}{2}$ Hz to $f_c + \frac{B}{2}$ Hz. The generated passband signals are transmitted through the speaker of the smartphone. Note that it is unnecessary to generate the signal in real-time, since the transmitted training sequence is always fixed. So we store the generated signal as a format of 16-bit Pulse Coded Modulation (PCM), and any smartphone that can play PCM (*i.e.*, WAV) format file can transmit the signal.

Figure 5.1(a) illustrates the signal generation and the transmission process.

We denote the training sequence as frame. Between frames, we insert fixed amount of gap (*i.e.*, zero symbols) to avoid inter-frame-interference. The gap should be sufficiently long so that the delayed signal from the previous frame does not interfere with the new frame signal. On the other hand, it should not be too long to shorten the frame interval. After investigation on the delay of the reflected signal, we decided to as 24 zero symbols between frames. As a consequence, the frame interval is 50 symbols that corresponds 12.5 ms[1].

5.2.2 Signal Reception

The acoustic signal is delivered through the inaudible band and received by the microphone. The received passband signal $y(t)$ is converted into baseband symbol $r[n]$ by the following down-conversion process: $y(t)$ is multiplied by $\sqrt{2}\cos(2\pi f_c t)$ and $-\sqrt{2}\sin(2\pi f_c t)$ to get the real and imaginary parts of the baseband symbol, respectively. After low-path filtering and down-sampling that selects the signal every baseband symbol interval, we get the

[1]The baseband symbol rate is $\frac{1}{B} = 0.25$ ms.

complex baseband signal [2]:

$$\begin{aligned} r[n] &= \sqrt{2}\cos(2\pi f_c t)y(t) - j\sqrt{2}\sin(2\pi f_c t)y(t), \\ &= \sqrt{2}e^{-j2\pi f_c t}y(t), \end{aligned}$$

where t is the time that the n-th baseband symbol is sampled (*i.e.*, $t = n \times T_s$, where T_s is symbol interval). The signal reception and the baseband conversion process is illustrated in Figure 5.1(b).

From the received signal, we first detect the frame and estimate the channel $h[n]$ to track the finger movement using known training sequence of the frame. They are less important issues in our tracking system design so we defer the discussion of them to Section 5.4.

5.3 Tracking finger movement using the reflected signal

In this section, we explain how the reflection affects the received signal and how we track the finger movement using the estimated channels.

5.3.1 Impact of Reflection

The received audio signal is not only the transmitted signal plus gaussian noise, but also the superposition of multiple signals with different delays. Considering the slow prorogation speed of the acoustic signal, such an ISI is

[2]Conventionally, $(\cdot)$ and $[\cdot]$ notations are used to represent analog and digital signals, respectively. Here, every signal is digital because a mobile app cannot access the analog signal. We use $(\cdot)$ and $[\cdot]$ notations to distinguish upsampled signal with rate f_s from the downsampled signal with rate B.

indispensable. The received signal affected by multi-path reflections is traditionally modeled as Linear Time-Invariant (LTI) system [61]. Let the channel has L paths and the received signal from path i has delay τ_i and amplitude a_i determined by the travel distance of the path. Then, without considering the impact of noise, the received signal $y(t)$ that is the summation of L signals can be represented as:

$$\begin{aligned} y(t) &= \sum_{i=1}^{L} a_i x(t-\tau_i) \\ &= \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_i} s(t-\tau_i) \\ &= h(t) * s(t), \end{aligned}$$

where $h(t)$ is the channel impulse response and

$$h(t) = \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_i} \delta(t-\tau_i).$$

The channel we estimate from the received baseband symbols is the discrete output of $h(t)$ sampled every T_s [61], which is

$$h[n] = \sum_{i=1}^{L} a_i e^{-j2\pi f_c \tau_i} \operatorname{sinc}(n - \tau_i W),$$

where $\operatorname{sinc}(t) = \frac{\sin(\pi t)}{\pi t}$. Conventionally, $h[n]$ is called the n-th channel tap value, because CIR is regarded as discrete-time filter in LTI system. Note that sinc function decays over time so the impact of delayed signal on the measured $h[n]$ is small when the difference between nT_s and τ_i are sufficiently large. However, if they are relatively close, it is captured to multiple channel taps

because of the signal dispersion effect of the sinc function. This is illustrated in Figure 5.2, where the channel is affected by one reflected signal with the travel distance 30 cm (*i.e.*, $\tau = 0.697$ ms). From the figure, we can observe that not only the closest channel tap from τ (*i.e.*, $h[3]$), but also the other nearby channels are affected by the single reflected signal.

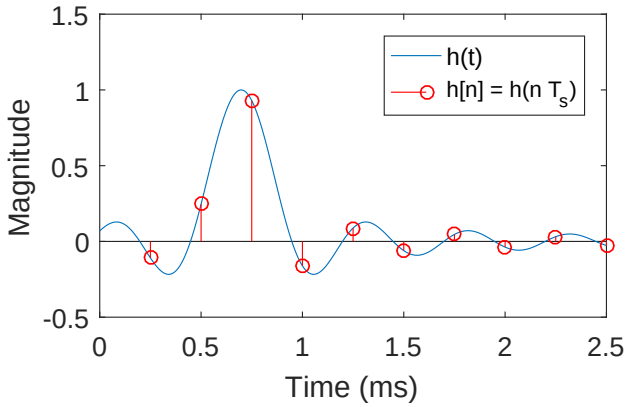


Figure 5.2: Impact of sinc function on the estimated channels.

In order to identify the impact of the reflected signal in practice, we observed the phase rotation of the received signal with a single and small moving object. We move a small aluminum ball with diameter less than 1 cm that is attached to a long and thin wood stick. The person moving the stick was at least 1 m away from the ball, so we guaranteed that there is a single moving object while receiving the audio signal. The initial distance from the microphone and the moving distance are approximately 30 cm and 20 cm, respectively. Figure 5.3 shows the phase of multiple channel taps while the ball is moving towards the speaker and the microphone. How to observe the phase change will be explained in detail in Section 5.3.2. The result shows that the

phase rotation is observed in multiple taps. Having the phase rotation in the measured channel implies the distance of the path is changing, which is caused by the moving ball. As shown in Figure 5.3, even with the movement of a single object, the phase rotation is observed in multiple taps. With the repeated experiments, we concluded that the phase rotation is observed approximately in 3 consecutive taps and assumed that the reflected signal with delay τ affects 3 $h[n]$s with smallest $|\tau - nT_s|$. As a result, $h[n]$ can be approximated as:

$$h[n] \approx \sum_k a_k e^{-j2\pi f_c \tau_k}, \qquad (n - \frac{3}{2})T_s < \tau_k < (n + \frac{3}{2})T_s.$$

In other words, each channel tap value $h[n]$ contains the phase and magnitude information of the reflected signals whose delays are between $(n - \frac{3}{2})T_s$ and $(n + \frac{3}{2})T_s$. The delay of a path is determined by the travel distance with the relationship $d_k = \tau_k V_c$, where d_k is the travel distance of the path k and V_c is the propagation speed of the audio (*i.e.*, $V_c \approx 340m/s$). Assuming that the speaker and the microphone are closely located, the distance from the microphone to the reflecting object is approximately half of the travel distance. Therefore, $h[n]$ contains the information on the objects whose distance from microphone are between $(n-\frac{3}{2})\frac{T_s V_c}{2}$ and $(n+\frac{3}{2})\frac{T_s V_c}{2}$. Considering that $T_s = 0.25$ ms given the 4 KHz bandwidth, $\frac{T_s V_c}{2} = 4.25$ cm and we can regard each tap includes the information on the object of approximately 12 cm range. This enables us to avoid the interference from the movement of the objects outside of the target range. Suppose we want to track the movement of finger whose distance from the mobile device is 50 cm. The movement outside of the target

range does not affect the channel tap values of whose indices are smaller than 8. By focusing only on those channel taps, we can easily filter out the movement of the objects outside of the target range.

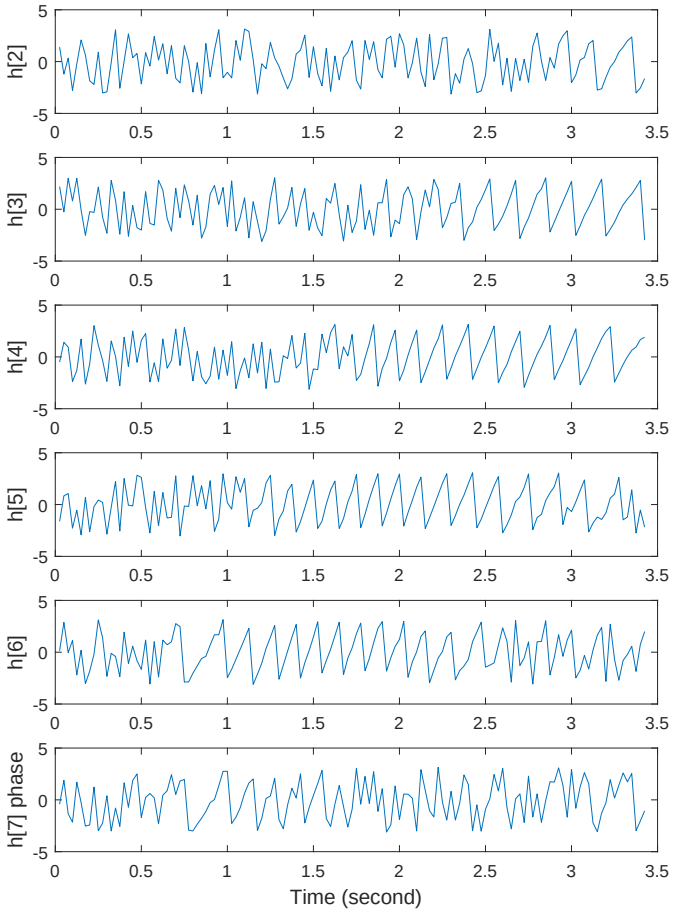


Figure 5.3: Phase change in multiple channel taps while moving a ball.

5.3.2 Tracking the phase change

In the previous section, we showed that from the sequence of the CIR vector, we can individually observe the channels of the paths with different distances. However, it does not mean we can directly observe the phase change caused by the target movement from CIR itself, because in each channel tap value, the information of multiple paths with similar distances are still mixed. From the summation of multiple channels, how to observe the phase change by the target movement is the first problem to resolve. Another problem is how to find the right channel tap that includes the path of the moving target. We will address the former problem here and discuss the latter problem in Section 5.3.3. So now we assume the k-th channel tap includes the path of the target.

In order to observe the phase change of the moving target, we compare two consecutive channel measurements. From single channel measurement, the phase change is not clearly visible due to the static components, but observing the difference of two consecutive samples make it distinct because it effectively removes dominant constant reflections. Let L_k denote the number of paths observed in $h[k]$, and suppose that the L_k-th path is the path reflected from the moving finger, while all of the other $L_k - 1$ paths are static during two

consecutive channel measurement periods $t-1$ and t. Then,

$$h[k]^{t-1} = \sum_{i=1}^{L_k} a_i e^{-j2\pi f_c \tau_i(t-1)},$$
$$h[k]^{t} = \sum_{i=1}^{L_k-1} a_i e^{-j2\pi f_c \tau_i(t)} + a_{L_k} e^{-j2\pi f_c (\tau_{L_k}(t-1)+\tau_d(t))},$$

where $h[k]^t$ is the k-th channel tap value estimated from the t-th frame and $\tau_d(t)$ is the delay difference caused by the movement of the target between the t-th and the $(t-1)$-th frame interval (*i.e.*, $\tau_d(t) = \tau_{L_k}(t) - \tau_{L_k}(t-1)$). By subtracting them, we get

$$h_d[k]^t = a_{L_k}(e^{-j2\pi f_c(\tau_{L_k}(t-1)+\tau_d(t))} - e^{-j2\pi f_c \tau_{L_k}(t-1)}), \tag{5.3.1}$$

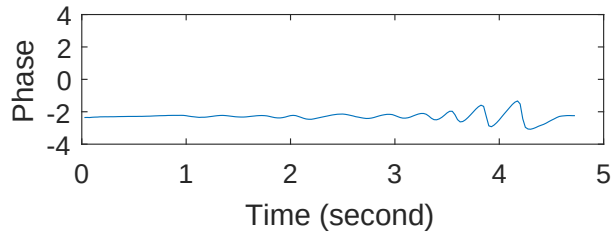
where $h_d[k]^t = h[k]^t - h[k]^{t-1}$. Equation 5.3.1 assumed that a_{L_k} is constant over two consecutive measurements. From the angle of $h_d[k]^t$, we can observe phase rotation caused by the change of $\tau_{L_k}(t)$.

$$\angle(h_d[k]^t) = \angle(e^{-j2\pi f_c(\tau_{L_k}(t-1)+\tau_d(t))} - e^{-j2\pi f_c \tau_{L_k}(t-1)}) \tag{5.3.2}$$
$$= \angle(e^{-j2\pi f_c \tau_{L_k}(t-1)}(e^{-j2\pi f_c \tau_d(t)} - 1)) \tag{5.3.3}$$
$$= \angle(e^{-j2\pi f_c \tau_{L_k}(t-1)}) + \frac{\angle(e^{-j2\pi f_c \tau_d(t)})}{2} + \frac{\pi}{2}, \tag{5.3.4}$$

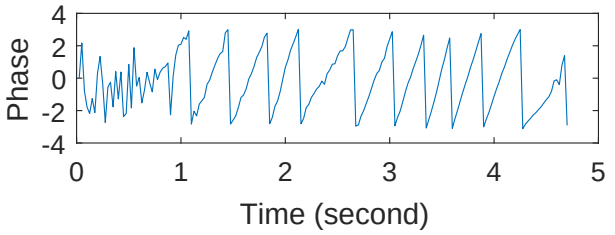
where $\angle(X)$ is the phase of the complex number X[3].

Figure 5.4 (a) and (b) show the phase of $h[k]$ and $h_d[k]$, respectively, from the trace we collected while a user is moving a finger to the direction of the speaker and the microphone. In the collected data, we conjecture $h[k]$

[3]It is not difficult to confirm $\angle(e^{-j2\pi a} - 1) = \frac{\angle(e^{-j2\pi a})}{2} + \frac{\pi}{2}$ geometrically.



(a) Phase of $h[k]$



(b) Phase of $h_d[k]$

Figure 5.4: Phase of the channel impulse responses while a finger is moving.

includes the finger movement related path between 1.0 second and 4.6 second. In Figure 5.4 (a), the phase is very stable and the change by the finger movement is not clearly shown because the majority portion of $h[k]$ is the signals from the static paths. In $h_d[k]$ where the impact of the static paths are effectively removed, we can observe clear phase rotation depending on the finger movement.

From the phase difference between $h_d[k]^{t+1}$ and $h_d[k]^t$, we can get the phase rotation caused by the delay difference $\angle(e^{-j2\pi f_c \tau_d(t)})$, and eventually the travel distance of the finger during the measurement interval using the relation between the phase change. Note that $\tau_d(t) = \tau_{L_k}(t) - \tau_{L_k}(t-1)$.

Using Equation 5.3.4, we can represent the phase difference as

$$\begin{aligned}\angle(h_d[k]^{t+1}) - \angle(h_d[k]^t) &= \angle(e^{-j2\pi f_c \tau_{L_k}(t)}) - \angle(e^{-j2\pi f_c \tau_{L_k}(t-1)}) \\ &\quad + \frac{1}{2}(\angle(e^{-j2\pi f_c \tau_d(t)}) - \angle(e^{-j2\pi f_c \tau_d(t-1)})) \\ &= \angle(e^{-j2\pi f_c \tau_d(t)}) + \frac{1}{2}(\angle(e^{-j2\pi f_c \tau_d(t)}) - \angle(e^{-j2\pi f_c \tau_d(t-1)})).\end{aligned}$$

By solving the above equation, we can calculate $\angle(e^{-j2\pi f_c \tau_d(t)})$. When no prior $\angle(e^{-j2\pi f_c \tau_d(t-1)})$ is given, we can simply assume $\tau_d(t) = \tau_d(t-1)$. Once we get the phase rotation, we can calculate the distance change from it:

$$d^t = \lambda \times \angle(e^{-j2\pi f_c \tau_d(t)}),$$

where d^t is the distance change of the dynamic path at time t, and λ is the wavelength of the audio signal.

5.3.3 Finding the channel tap with the finger movement

So far, we have assumed it is already known which tap to observe to track the finger movement. This section describes how to find the correct tap that includes the path reflected from finger among multiple possible taps. Note that as we discussed in Section 5.3.1, the phase rotation by the finger movement is observed in multiple taps rather than in a single tap, which eases the problem. The channel taps can be broadly divided into taps that includes the dynamic path or not, where the right tap should include the dynamic path. If all of the paths in the tap are static, the channel estimation result does not significantly change over time. For ease of description, we denote the channel

taps that includes dynamic paths as dynamic taps, and the other taps static taps. Our goal is classifying the static taps and filter them out.

We define the metric to detect the existence of the dynamic path in the tap k:

$$M_1[k]^t = \frac{|h[k]^t - h[k]^{t-1}|}{|h[k]^t|}, \tag{5.3.5}$$

which calculates the magnitude of the difference of two consecutive channels and normalize it by the magnitude of the current channel. If $M_1[k]^t$ is smaller than the threshold σ_1, we regard the channel k does not include any dynamic path and ignores it. In our tracking implementation and the performance evaluation, we set $\sigma_1 = 0.05$.

While the above metric effectively classifies dynamic and static taps, if noise is added in the estimated channel, mis-classification can happen. If a static tap is detected as a dynamic tap, the impact of it on the tracking is huge because the phase of the static tap randomly changes. To avoid it, we devise a second classification metric. Our another observation on the static tap k is that the phase rotation is very unstable because $h_d[k]$ is purely noise where all static paths are removed. On the other hand, if the phase rotation is caused by the distance change of the dynamic path, the phase rotation is more stable because the delay change over multiple measurement periods is relatively small. Recall Figure 5.3 and 5.4 where the phase randomly changes when the dynamic path is not included in the specific channel tap.

Based on this observation, we devise another metric to filer out the static tap. Among the dynamic tap candidates whose $M_1[k]^t$ are higher than

σ_1, we measure the stability of the phase change. We define the stability as the phase change difference over the last 3 measurements. Specifically, we find the maximum phase change over the 3 periods, which is

$$M_2[k]^t = \max_{i=t-2,t-1,t} f(i),$$
$$f(i) = |\angle(e^{-j2\pi f_c \tau_d^k(t)}) - \angle(e^{-j2\pi f_c \tau_d^k(t-1)})|.$$

If $M_2[k]^t$ is higher than σ_2, we regard it as static tap After that, from the remaining dynamic taps, find the tap whose index is the smallest and select it as the tap that includes the path from the moving finger. We select $\sigma_2 = 0.5$ in our implementation.

Among the dynamic taps that passes two tests above, we select the tap with smallest k as the tap with the finger movement. It is because we assume that the finger is the closest point from the speaker and the microphones in our application scenario.

5.3.4 Tracking the moving finger in 2D space

Using the channel tap selection and the phase rotation tracking explained in the previous two sections, we can continuously track the distance change of the finger from the microphone. This alone is already useful for the finger-based gesture recognition, but with the measurements from two microphones in different locations, we can track the movement of the finger in 2D space. This allows the user to point specific target on the screen and enables more diverse gesture design. This section explains how to track the

finger movement in 2D space using a smartphone with one speaker and two microphones.

From the form factor of the mobile phone, we know the relative location of the speaker and the microphones. Suppose the speaker is located at the origin of the coordinate, and the two microphones are located at (x_1, y_1) and (x_2, y_2), respectively. Using the previous position and the distance change estimated by the phase tracking, we can calculate d_k that is the travel distance of the signal transmitted from the speaker and reflected by the finger and received by the microphone k. Then we can find the position of the moving finger by solving the following optimization problem:

$$\min_{x,y} : \sum_{k=1}^{2} (\sqrt{(x_k - x)^2 + (y_k - y)^2} + \sqrt{x^2 + y^2} - d_k)^2.$$

One remaining issue is how to find the initial distance of d_k. Although the phase tracks the movement in fine resolution, it relies on the previous position to find the new position, which eventually requires information on the initial position in the beginning of the tracking. Using the current channel tap information, we can roughly track the absolute distance. As explained in Section 5.3.1, the channel tap $h[n]$ includes the path information reflected by the object with distance between $0.425 \times (n-1)$ cm and $0.425 \times n$ cm. Therefore, initially, if the finger movement is detected from $h[k]$, APT approximates the initial distance as $0.425 \times (k - \frac{1}{2})$ cm and keeps tracking the location by the phase tracking.

5.3.5 Improving the accuracy using the over-sampled signals

Lastly, we introduce a simple but efficient way to improve the accuracy of the phase estimation. In traditional digital communication, downsampling during the passband to basedband down-converting process is simply picking one sample every r samples from the over-sampled signals, where r is the upsampling rate. In APT, r is 12 so 11 samples out of 12 samples are dropped in the downsampling process. Here, instead of simply dropping over-sampled signals, we get the average of the them to reduce the noise of the output. Every r samples in each sampling interval, we pick the first l samples and get the average of them as downsampling output. After investigation, we chose $l = 4$ for our system design. The impact of l will be evaluated in microbenchmark of Section 5.5.

5.4 Practical Issues

This section describes the practical issues implementing APT such as how it detects the frame from the received signal and estimates the channel using it.

Frame detection: At the receiver side, the next thing to do after the passband-to-baseband signal conversion in Section 5.2.2 is detecting the first symbol of the received frame. It can be done by either energy detection or cross-correlation check. While a sophisticated scheme such as cross-correlation check is desirable, we found even a simple energy based scheme works well and

detects the beginning of the frame with very high accuracy. Since the positions of the speaker and the microphone are fixed and closely located, the SNR of the received signal is always maintained high enough and even a simple energy detection scheme accurately detects the frame regardless of the environmental noise [4]. Our frame detection algorithm works as follows. If the magnitude of the 3 consecutive symbols are higher than the threshold σ, we regard that the first symbol is the beginning of the frame symbols. In our implementation, we set σ to 0.003. Note the the frame detection procedure is only necessary in the beginning of the tracking. Since the frame interval is fixed, once the frame is detected, the next frames can be easily found without detection.

Channel estimation: Using the received frame and the known information on the training sequence, we can estimate the channel. There are several existing channel estimation algorithms in single carrier based communication system. While Maximum Likelihood Sequence Estimation (MLSE) is known to be optimum, its complexity is a bit high. Least-Squares (LS) channel estimation is computationally simple but suboptimal [55]. Considering that APT is implemented as a mobile app, we decide to use simple ML based channel estimation scheme. We mainly focus on our implementation of the algorithm while referring the reader to [45] for the fundamental theory behind it.

For LS channel estimation, one needs to decide the reference length P

[4]The SNR can depend on the volume of the speaker and the microphone that the user selects. However, we believe it is not difficult to fix the volume of the speaker and microphone high enough for tracking regardless of the user setting in implementation-level.

and the memory length L, where L determines the number of channel taps we can estimate and P plus L is the training sequence length. With larger L, we can estimate more channel taps but the reliability of the estimation is reduced. In our implementation, we decided to use $P = 16$ and $L = 10$, which implies we can track the movement approximately up to 50 cm distance. This is just our design choice and can be easily extended by using shorter P or longer training sequence.

The channel estimation is implemented as follows.
Let $\mathbf{m} = \{m_1, m_2, \ldots, m_{L+P}\}$ be the vector of training sequence. Then we define a circulant training matrix $\mathbf{M} \in \mathbb{R}^{P \times L}$ as

$$\mathbf{M} = \begin{bmatrix} m_L & m_{L-1} & m_{L-2} & \ldots & m_1 \\ m_{L+1} & m_L & m_{L-1} & \ldots & m_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{L+P} & m_{L+P-1} & m_{L+P-2} & \ldots & m_{P+1} \end{bmatrix}.$$

Let $\mathbf{y} = \{y_1, y_2, \ldots, y_{L+P}\}$ be the received training sequence. The channel is estimated by

$$\hat{\mathbf{h}} = (\mathbf{M}^H \mathbf{M})^{-1} \mathbf{M}^H \mathbf{y_L}, \tag{5.4.1}$$

where $\mathbf{y_L} = \{y_{L+1}, y_{L+2}, \ldots, y_{L+P}\}$.

Given the pre-computed $(\mathbf{M}^H \mathbf{M})^{-1} \mathbf{M}^H$, the computational cost of the channel estimation is only the matrix-to-vector multiplication with the complexity is $O(P \times L)$. Considering $P = 16$ and $L = 10$, the complexity of the channel estimation is pretty low and acceptable for mobile devices.

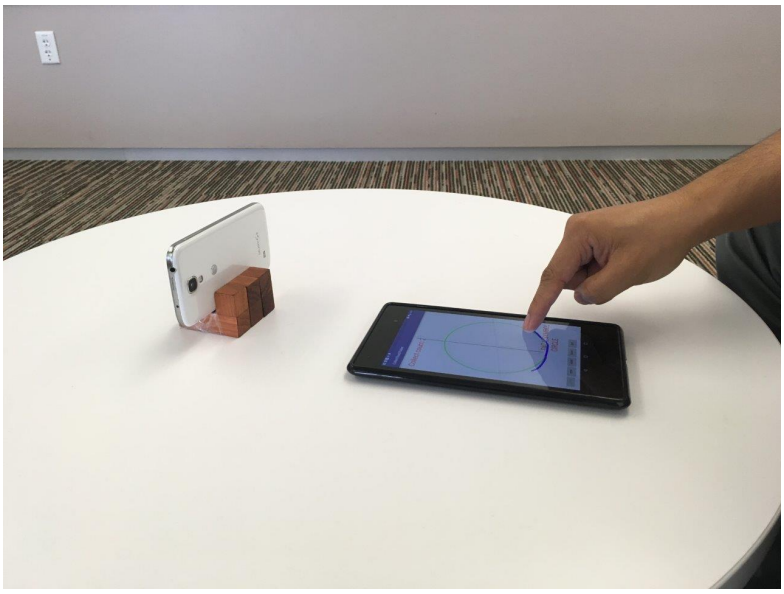


Figure 5.5: Testbed setup for the performance evaluation.

5.5 Performance Evaluation

5.5.1 Experimental setup

We evaluate the tracking performance of APT and compare it with the other tracking schemes. We used Samsung Galaxy S4 mobile phone as a testbed device, which has one rear speaker and two microphones at the top and the bottom of it with 14 cm spacing. We implemented an Android app that plays audio file generated as explained in Section 5.2, and receives the audio signal while a user is moving a finger. For audio signal transmission, we use inaudible frequency band between 18 KHz and 22 KHz, and the tracking interval is 12.5 ms. To improve the accuracy, we average 4 samples out of 12 upsampled signals in the downsampling process, as explained in Section 5.3.5. The impact of the number of samples in averaging in evaluated in Section 5.5.3.

To collect the ground-truth of the finger movement, we let the user move the finger on the top of the smart tablet, which collects the touch event of the screen and generate the ground-truth trajectory of the finger movement. In the experiment, we show simple shapes such as triangle, diamond, circle on the screen of the tablet and ask the user to follow the line of the shape. The average distance of the shapes is 22.3 cm. Figure 5.5 shows our testbed setup.

5.5.2 Compared Schemes

For the performance comparison, we implement the other two acoustic signal based device-free tracking schemes and compare the tracking capability with them.

Sine wave based phase tracking: In *sine wave* based tracking, the transmitter continuously sends sine waves and the receiver tracks the moving object by observing the phase change of the received waves. The transmitted signal $\sin(2\pi f_c t)$ is generated using MATLAB, stored as 16-bit PCM-format *WAV* file, and transmitted through the speaker. Similar to our receiver design in Section 5.2.2, the received signal from the microphone is first multiplied by $\sqrt{2}\cos(2\pi f_c t)$ and $-\sqrt{2}\sin(2\pi f_c t)$ and low-pass filtered to get the real and imaginary term of the received signal, respectively. The phase of the moving finger is tracked by Dual-differential Background Removal (DDBR) algorithm in [65] that filters out constant-path signals by observing the phase of the difference of two consecutive signals. We improve the accuracy of the phase estimation by averaging multiple received signals. Let T_s be the sampling in-

terval of the phase. We take the phase after averaging all of the received signals during T_s. T_s is selected as 12.5 ms so that both APT and *sine wave* based tracking has equivalent tracking delay. In addition, we achieve the frequency diversity gain by sending multiple the sine waves in different frequencies. Using 4KHz bandwidth between 18 KHz and 22 KHz, the transmitter sends multiple sine waves with 500 Hz interval. As a result, 9 waves are concurrently transmitted. By taking the mean of the phase estimations of them, the accuracy of the phase estimation can be improved. Once the phase is estimated, we use the same position tracking algorithm introduced in Section 5.3.4.

The main difference between APT and *sine wave* based tracking is that the former one separately measures the phase change of the signals with different delays while the latter one observes the phase change caused by all of the moving objects in a single measurement. Considering that several parts of body position may change together while the user moves the finger, observing the combined phase change yields less accurate finger tracking. Also, it is especially problematic when the user is surrounded by moving people and objects such as home and office environment. With the extensive comparison, we show the benefit of APT compared to the *sine wave* based tracking.

Cross-correlation based tracking: It tracks the position of the moving object by observing the change of the cross-correlation of the received signal called *echo profile*. The baseline of it is FingerIO [37] that transmits an OFDM symbol in each frame and finds the location of the moving finger that causes the change of the echo profiles of the two consecutive frames. We carefully followed

the description of [37] to implement FingerIO. However, we were not able to track the position of the finger with it, so we made following modifications to improve the accuracy and enable tracking. First, we used larger bandwidth and used longer FFT size for OFDM symbol generation. It is based on our observation that using more subcarriers for data transmission increases the accuracy of the tracking. We select 6 KHz bandwidth between 16 KHz and 22 KHz as the data transmission bandwidth, and the FFT size of 256. Second, when we locate the finger by comparing two echo profiles, instead of detecting the location with the difference higher than a threshold, we select the location that gives the maximum difference. Our observation on the echo profile is, even with larger bandwidth and longer interval, the difference of two profiles while the finger is moving is very small, which makes it quite challenging to select an appropriate threshold. Therefore, we find the position with the maximum cross-correlation difference. It roughly tracks the position of the moving finger, but often detects random location due to noise. To address it, we ignore the result when the detected position is significantly different from the previous position. We set the threshold for the deviation to 10 cm. After getting the distance estimation, we use the same algorithm introduced in Section 5.3.4 to find the position of the finger in 2D plane.

5.5.3 Experimental Result

Tracking accuracy in 1D: We first evaluate the accuracy of the distance change tracking in one dimension. In this experiment, the user initially places

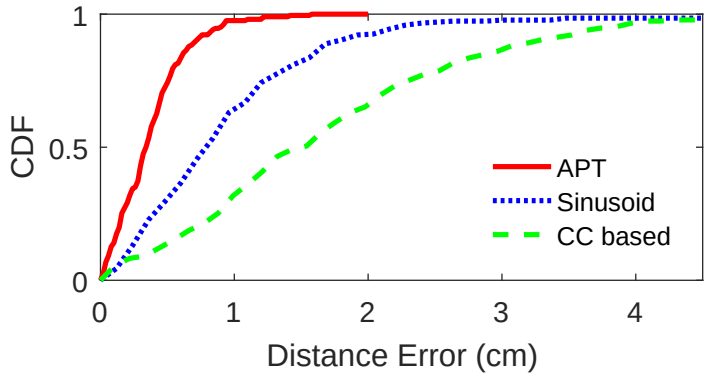


Figure 5.6: CDF of the distance tracking error.

the finger 20 cm away from the mobile phone and moves 10 cm towards it. We repeat the experiment 200 times for each of the scheme and collected the CDF of the distance error, which is shown in Figure 5.6. The median errors of APT, *sine wave* based tracking (*i.e.*, sinusoid), and cross-correlation based tracking (*i.e.*, CC based) are 0.3 cm, 0.8 cm, and 1.5 cm, respectively. The 90th percentile error of them are 0.7 cm, 1.8 cm, and 3.2 cm, respectively. The result reveals that APT can accurately track the movement by observing the phase rotation from CIR. The median tracking error of it is less than half of the *sine wave* based tracking's. Since APT separately tracks the phase in the CIR vector, it can effectively filter out the measurement noise from the user's body movement while the *sine wave* based tracking cannot. The accuracy of the cross-correlation based tracking is much lower than the phase based tracking.

One of the advantages of APT is that it can distinguish the movements with different delays. This allows us to easily filter out the interference

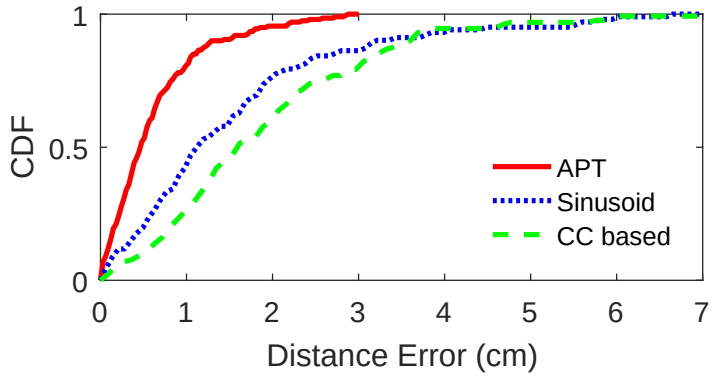


Figure 5.7: CDF of the distance tracking error with interrupting user.

incurred by the movement of the surrounding objects in the phase tracking. By limiting the phase observation to the channel tap smaller than the 10th one, we can ignore the phase change caused by the moving object with the distance approximately larger than 50 cm. To evaluate it, we performed the distance tracking experiment with a moving person. Here, a user is moving one's body in 50 cm distance from the mobile phone while we are conducing the experiment, where the CDF of the distance tracking error is shown in Figure 5.7. In APT, the tracking accuracy is not seriously affected by the interfering user. Compared withe the experiment with no interference, the median error increases by 1 mm. The cross-correlation based tracking is not significantly influenced by the moving user either. Here, we set it to focus on the distance change within the range between 0 to 40 cm. Even if there is any echo-profile change in the distance of the moving user, we did not track it. As a result, it can effectively avoid the interference. On the other hand, *sine wave* based tracking undergoes degraded tracking performance due to the moving

user. The median error increases from 0.8 cm to 1.2 cm. Since it does not has any mechanism to distinguish the phase change caused by multiple objects, the movement in surroundings affects its tracking more significantly.

Figure 5.8 is the average 1D tracking error with 95% confidence interval when various number of samples are used for averaging downsampled signals as explained in Section 5.3.5. The result shows that averaging 4 samples improves the accuracy from 0.39 cm to 0.34 cm compared to simple downsampling. Moreover, it is effective in reducing the variation of the result. The 95th percentile error decreases from 0.9 cm to 0.7 cm. Using more than 4 samples does not significantly improve the accuracy, so we decided to use 4 samples to reduce the computational complexity.

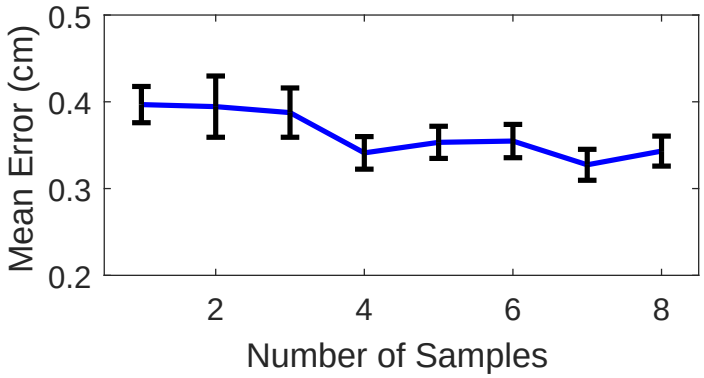


Figure 5.8: CDF of the distance tracking error.

Tracking accuracy in 2D: Next, we evaluate the tracking error in 2D plane. We collect the data while the finger of the user is following the shapes on the screen of the tablet, and the trajectory error is calculated by averaging the distance difference between the estimated and ground-truth positions from

all samples. The absolute position offset in the beginning of the tracking is ignored. Figure 5.9 shows the CDF of the trajectory errors of the three schemes. The median errors of APT, *sine wave* based tracking, and cross-correlation based tracking are 0.9 cm, 1.8 cm, and 3.5 cm, respectively. The 90th percentile error of them are 2.2 cm, 3.4 cm, and 4.2 cm, respectively. Again, APT provides much lower tracking error than the compared schemes.

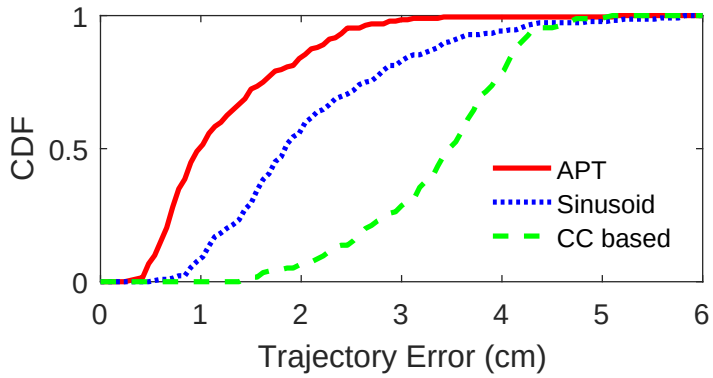
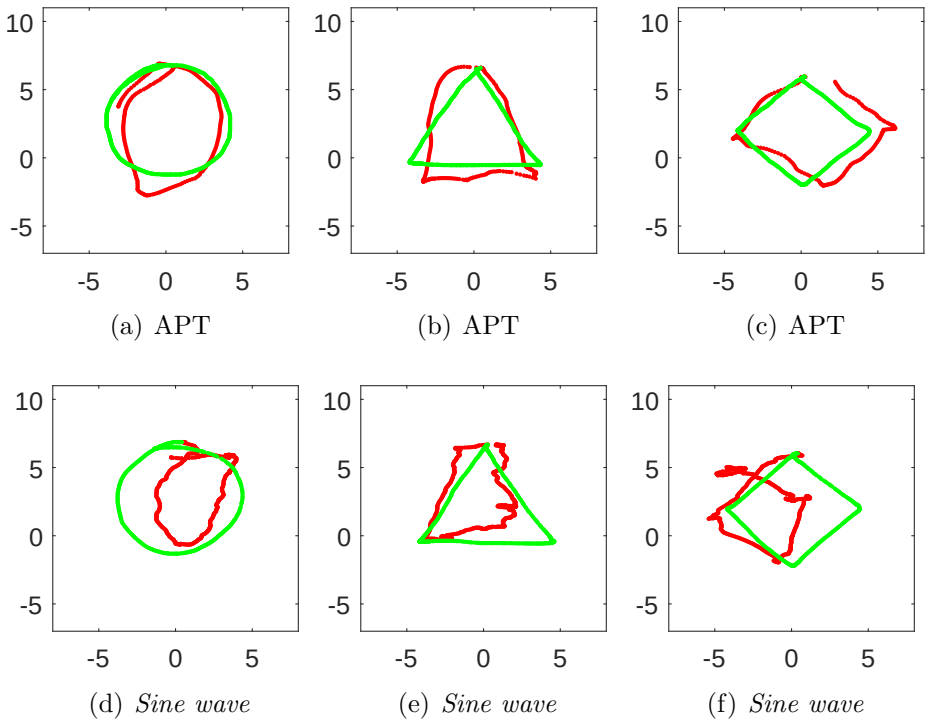


Figure 5.9: CDF of the trajectory error.

Figure 5.10 shows the median error samples of the finger movement tracked by APT and *sine wave* based tracking while the users are following the shapes on the table screen. The result shows that the trajectory tracked by APT is close to the ground-truth finger movement. We believe this is sufficient to recognize sophisticated gestures drawn by finger. On the other hand, the drawings by *sine wave* based tracking significantly deviate from the original shape.



(a) APT (b) APT (c) APT

(d) *Sine wave* (e) *Sine wave* (f) *Sine wave*

Figure 5.10: Shapes drawn by finger movement tracking.

Chapter 6

Conclusion

In this dissertation, we provided accurate object tracking techniques that are useful to design the natural user interface for smart devices.

In Chapter 2, we develop a novel system that can accurately track hand movement and apply it to realize a mouse. A unique advantage of our scheme is that it achieves high tracking accuracy (*e.g.*, median error of around 1.4 cm) using the existing hardware already available in the mobile devices and equipment to be controlled (*e.g.*, smart TVs). Our evaluation and user study demonstrate the feasibility and effectiveness of our approach. Moving forward, we are interested in further enhancing the accuracy and robustness of tracking. Moreover, we would like to conduct larger-scale user study and develop interesting applications that benefit from accurate tracking.

Chapter 3 presents a novel device-free based tracking using audio signals. We extract both distance and velocity estimates from a single reflected chirp signal. To improve the robustness, we combine multiple peaks next to the highest peaks in FMCW to estimate velocity, and use the estimated velocity to select an appropriate FMCW peak to estimate the distance. We combine both estimates to continuously track the hand. Using micro benchmarks and user

study, our evaluation shows the feasibility and effectiveness of this approach. We hope our work will bring us closer to the vision of making devices easy to use. Moving forward, we are interested in further enhancing the capabilities and robustness of our tracking systems under more extensive usage scenarios, such as tracking 3D motion, multiple players, and under different environmental conditions. In particular, our motion tracking algorithm is applicable to 3D

In Chapter 4, we provide a fined-grained finger tracking. Since the wavelength of the acoustic signal in inaudible frequency is very short, even a slight movement causes significant phase change, which enables fine-grained tracking. The challenge is that the received signal from the microphone is the superposition of multiple reflected signals with different amount of movements. To address this challenge, we observe the phase from the channel impulse response where the reflected signals with different delay ranges are easily separable. We design our own acoustic data communication channel and estimate the channel from it. Then we extract the phase change caused by the finger movement with our novel static reflection cancellation and channel tap selection algorithms, and eventually track the finger movement. The performance evaluation result shows that our scheme provides the median error of 0.3 cm in 1D distance tracking and 1 cm median error in 2D trajectory tracking. Given the capability of the accurate finger tracking, we plan to extend it to device-free gesture recognition that is useful for natural user interface and interactive game design for wearable devices.

Bibliography

[1] Microsoft X-box Kinect. `http://xbox.com`.

[2] Nintendo Wii. `http://www.nintendo.com/wii`.

[3] Circle-circle intersection. `http://mathworld.wolfram.com/Circle-CircleIntersection.html`.

[4] Fadel Adib, Zach Kabelac, Dina Katabi, and Rob Miller. Witrack: Motion tracking via radio reflections off the body. In *Proc. of NSDI*, 2014.

[5] Sandip Agrawal, Ionut Constandache, Shravan Gaonkar, Romit Roy Choudhury, Kevin Caves, and Frank DeRuyter. Using mobile phones to write in air. In *Proc. of ACM MobiSys*, pages 15–28, 2011.

[6] Logitech air mouse. `http://www.logitech.com`.

[7] Siavash M Alamouti. A simple transmit diversity technique for wireless communications. *Selected Areas in Communications, IEEE Journal on*, 16(8):1451–1458, 1998.

[8] Kamran Ali, Alex X. Liu, Wei Wang, and Muhammad Shahzad. Keystroke recongition using wifi signals. In *Proc. of ACM MobiCom*, 2015.

[9] Md Tanvir Islam Aumi, Sidhant Gupta, Mayank Goel, Eric Larson, and Shwetak Patel. Doplink: Using the doppler effect for multi-device interaction. In *Proc. of ACM UbiComp*, pages 583–586, 2013.

[10] Gary Bradski. The OpenCV library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[11] M. Burkhart, P. v. Richenbach, R. Wattenhofer, and A. Zollinger. Does topology control reduce interference? In *Proc. of ACM MOBIHOC*, May 2004.

[12] Ke-Yu Chen, Daniel Ashbrook, Mayank Goel, Sung-Hyuck Lee, and Shwetak Patel. Airlink: Sharing files between multiple devices using in-air gestures. In *Proc. of ACM Ubicomp*, 2014.

[13] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proc. of Robotics and Automation*, volume 2, pages 1322–1328, 1999.

[14] Frame rate. `http://en.wikipedia.org/wiki/Frame_rate`.

[15] Daniel Goehl and David Sachs. Motion sensors gaining inertia with popular consumer electronics. *White Paper, IvenSense Inc*, 2007.

[16] Sidhant Gupta, Daniel Morris, Shwetak Patel, and Desney Tan. Soundwave: using the doppler effect to sense gestures. In *Proc. of the SIGCHI*, pages 1911–1914, 2012.

[17] Sidhant Gupta, Daniel Morris, Shwetak Patel, and Desney Tan. Soundwave: using the doppler effect to sense gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1911–1914. ACM, 2012.

[18] Robin Heydon and Nick Hunn. Bluetooth low energy. *CSR Presentation, Bluetooth SIG https://www. bluetooth. org/DocMan/handlers/DownloadDoc. ashx*, 2012.

[19] Wenchao Huang, Yan Xiong, Xiang-Yang Li, Hao Lin, Xufei Mao, Panlong Yang, and Yunhao Liu. Shake and walk: Acoustic direction finding and fine-grained indoor localization using smartphones. In *Proc. of IEEE INFOCOM*, 2014.

[20] InvenSense. `http://www.invensense.com`.

[21] Eric Jacobsen and Richard Lyons. The sliding DFT. *IEEE Signal Processing Magazine*, 20(2):74–80, 2003.

[22] Kiran Joshi, Dinesh Bharadia, Manikanta Kotaru, and Sachin Katti. Wideo: Fine-grained device-free motion tracing using RF backscatter. In *Proc. of NSDI*, 2015.

[23] Bryce Kellogg, Vamsi Talla, and Shyam Gollakota. Bringing gesture recognition to all devices. In *Proc. of NSDI*, April 2014.

[24] Swarun Kumar, Diego Cifuentes, Shyamnath Gollakota, and Dina Katabi. Bringing cross-layer MIMO to today's wireless lans. In *In Proc. of ACM SIGCOMM*, August 2013.

[25] Swarun Kumar, Stephanie Gil, Dina Katabi, and Daniela Rus. Accurate indoor localization with zero start-up cost. In *Proc. of ACM MobiCom*, pages 483–494, 2014.

[26] Patrick Lazik and Anthony Rowe. Indoor pseudo-ranging of mobile devices using ultrasonic chirps. In *Proc. of ACM SenSys*, pages 99–112, 2012.

[27] Chang-Hsing Lee, Jau-Ling Shih, Kun-Ming Yu, and Hwai-San Lin. Automatic music genre classification based on modulation spectral analysis of spectral and cepstral features. *IEEE Transactions on Multimedia*, 11(4):670–682, 2009.

[28] LG magic motion remote. `http://www.lg.com/us/tv-accessories/lg-AN-MR200-motion-remote`.

[29] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proc. of UbiComp*, pages 421–430, 2012.

[30] Belinda J. Lipa and Donald E. Barrick. FMCW signal processing. Mirage Systems, Sunnyvale, California.

[31] Apple 13-inch MacBook Pro technical specifications. https://www.apple.com/macbook-pro/specs/.

[32] Alex T Mariakakis, Souvik Sen, Jeongkeun Lee, and Kyu-Han Kim. Sail: Single access point-based indoor localization. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 315–328. ACM, 2014.

[33] Adriano Meta, Peter Hoogeboom, and Leo P Ligthart. Signal processing for FMCW SAR. *Geoscience and Remote Sensing, IEEE Transactions on*, 45(11):3519–3532, 2007.

[34] Robert A Meyers. Encyclopedia of physical science and technology. *Facts on File*, 1987.

[35] Top EDM 2014 music playlist tracklist. https://www.youtube.com/watch?v=PHIRcu3Ero0.

[36] Rajalakshmi Nandakumar, Shyam Gollakota, and Nathaniel Watson. Contactless sleep apnea detection on smartphones. In *Proc. of ACM MobiSys*, 2015.

[37] Rajalakshmi Nandakumar, Vikram Iyer, Desney Tan, and Shyamnath Gollakota. Fingerio: Using active sonar for fine-grained finger tracking. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1515–1525. ACM, 2016.

[38] Hanna Nyqvist and Fredrik Gustafsson. A high-performance tracking system based on camera and IMU. In *Proc. of 16th IEEE International Conference on Information Fusion (FUSION)*, pages 2065–2072, 2013.

[39] Makoto Ono, Buntarou Shizuki, and Jiro Tanaka. Touch & activate: adding interactivity to existing objects using active acoustic sensing. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 31–40. ACM, 2013.

[40] Alan V Oppenheim, Ronald W Schafer, John R Buck, et al. *Discrete-time signal processing*, volume 2. Prentice-hall Englewood Cliffs, 1989.

[41] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals &Amp; Systems (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[42] Taiwoo Park, Jinwon Lee, Inseok Hwang, Chungkuk Yoo, Lama Nachman, and Junehwa Song. E-gesture: a collaborative architecture for energy-efficient gesture recognition with hand-worn sensor and mobile devices. In *Proc. of ACM SenSys*, pages 260–273, 2011.

[43] Chunyi Peng, Guobin Shen, Yongguang Zhang, Yanlin Li, and Kun Tan. BeepBeep: a high accuracy acoustic ranging system using COTS mobile devices. In *Proc. of ACM SenSys*, 2007.

[44] Qifan Pu, Sidhant Gupta, Shyam Gollakota, and Shwetak Patel. Whole-home gesture recognition using wireless signals. In *Proc. of ACM MobiCom*, 2013.

[45] Markku Pukkila. Channel estimation modeling. *Nokia Research Center*, 2000.

[46] Jian Qiu, David Chu, Xiangying Meng, and Thomas Moscibroda. On the feasibility of real-time phone-to-phone 3D localization. In *Proc. of ACM MobiSys*, 2011.

[47] Radar. `http://www.ll.mit.edu/workshops/education/videocourses/introradar/`.

[48] Anshul Rai, Krishna Kant Chintalapudi, Venkata N. Padmanabhan, and Rijurekha Sen. Zee: zero-effort crowdsourcing for indoor localization. In *Proc. of ACM MobiCom*, 2012.

[49] Theodore S Rappaport et al. *Wireless communications: principles and practice*, volume 2. Prentice Hall PTR New Jersey, 1996.

[50] A Rodríguez Valiente, A Trinidad, JR García Berrocal, C Górriz, and R Ramírez Camacho. Extended high-frequency (9–20 khz) audiometry reference thresholds in 645 healthy subjects. *International journal of audiology*, 53(8):531–545, 2014.

[51] Roku 3 streaming player. `https://www.roku.com/products/roku-3`.

[52] CES 2014: Samsung shows off new gyroscopic remote control. http://www.digitalversus.com/tv-television/ces-2014-samsung-shows-off-new-gy

[53] Kurt Seifert and Oscar Camacho. Implementing positioning algorithms using accelerometers. *Freescale Semiconductor*, 2007.

[54] Souvik Sen, Jeongkeun Lee, Kyu-Han Kim, and Paul Congdon. Avoiding multipath to revive inbuilding wifi localization. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 249–262. ACM, 2013.

[55] Bernard Sklar. *Digital communications*, volume 2. Prentice Hall NJ, 2001.

[56] CES 2014 trends: New remotes and interfaces to make smart TVs actually usable. http://spectrum.ieee.org/tech-talk/consumer-electronics/audiovideo/ces-2014-t

[57] Adam Smith, Hari Balakrishnan, Michel Goraczko, and Nissanka Priyantha. Tracking moving devices with the cricket location system. In *Proc. of ACM MobiSys*, 2005.

[58] Andrew G Stove. Linear FMCW radar techniques. In *IEE Proceedings F (Radar and Signal Processing)*, volume 139(5), pages 343–350. IET, 1992.

[59] L. Sun, S. Sen, D. Koutsonikolas, and K. Kim. Widraw: Enabling hands-free drawing in the air on commodity wifi devices. In *Proc. of ACM MobiCom*, 2015.

[60] Zheng Sun, Aveek Purohit, Raja Bose, and Pei Zhang. Spartacus: spatially-aware interaction for mobile devices through energy-efficient audio sensing. In *Proc. of ACM Mobisys*, pages 263–276, 2013.

[61] David Tse and Pramod Viswanath. *Fundamentals of wireless communication*. Cambridge university press, 2005.

[62] Jue Wang and Dina Katabi. Dude, where's my card? RFID positioning that works with multipath and non-line of sight. In *Proc. of the ACM SIGCOMM*, pages 51–62, 2013.

[63] Jue Wang, Deepak Vasisht, and Dina Katabi. RF-IDraw: virtual touch screen in the air using rf signals. In *Proc. of ACM SIGCOMM*, 2014.

[64] Wei Wang, Alex X Liu, Muhammad Shahzad, Kang Ling, and Sanglu Lu. Understanding and modeling of wifi signal based human activity recognition. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 65–76. ACM, 2015.

[65] Teng Wei and Xinyu Zhang. mTrack: high precision passive tracking using millimeter wave radios. In *Proc. of ACM MobiCom*, 2015.

[66] Peter D Welch. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified

periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 1967.

[67] Chadwick A Wingrave, Brian Williamson, Paul D Varcholik, Jeremy Rose, Andrew Miller, Emiko Charbonneau, Jared Bott, and JJ LaViola. The wiimote and beyond: Spatially convenient devices for 3d user interfaces. *IEEE Computer Graphics and Applications*, 30(2):71–85, 2010.

[68] Volker Winkler. Range doppler detection for automotive FMCW radars. In *Microwave Conference, 2007. European*, pages 1445–1448. IEEE, 2007.

[69] Grace Woo, Pouya Kheradpour, Dawei Shen, and Dina Katabi. Beyond the bits: Cooperative packet recovery using physical layer information. In *Proc. of ACM MobiCom*, 2007.

[70] Yaxiong Xie, Zhenjiang Li, and Mo Li. Precise power delay profiling with commodity wifi. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 53–64. ACM, 2015.

[71] Jie Xiong and Kyle Jamieson. Arraytrack: A fine-grained indoor location system. In *Proc. of NSDI*, pages 71–84, 2013.

[72] Lei Yang, Yekui Chen, Xiang-Yang Li, Chaowei Xiao, Mo Li, and Yunhao Liu. Tagoram: Real-time tracking of mobile RFID tags to high

precision using cots devices. In *Proc. of ACM MobiCom*, 2014.

[73] Sangki Yun, Yi chao Chen, and Lili Qiu. Turning a mobile device into a mouse in the air. In *Proc. of ACM MobiSys*, May 2015.

[74] Zengbin Zhang, David Chu, Xiaomeng Chen, and Thomas Moscibroda. Swordfight: Enabling a new class of phone-to-phone action games on commodity phones. In *Proc. of ACM MobiSys*, 2012.

[75] Yanzi Zhu, Yibo Zhu, Ben Y. Zhao, and Haitao Zheng. Reusing 60 GHz radios for mobile radar imaging. In *Proc. of ACM MobiCom*, 2015.