Depth Aware Finger Tapping on Virtual Displays

Ke Sun[†], Wei Wang[†], Alex X. Liu[†‡], Haipeng Dai[†]

[†]State Key Laboratory for Novel Software Technology, Nanjing University, China
[‡]Dept. of Computer Science and Engineering, Michigan State University, U.S.A.
kesun@smail.nju.edu.cn,ww@nju.edu.cn,alexliu@cse.msu.edu,haipengdai@nju.edu.cn

ABSTRACT

For AR/VR systems, tapping-in-the-air is a user-friendly solution for interactions. Most prior in-air tapping schemes use customized depth-cameras and therefore have the limitations of low accuracy and high latency. In this paper, we propose a fine-grained depth-aware tapping scheme that can provide high accuracy tapping detection. Our basic idea is to use light-weight ultrasound based sensing, along with one COTS mono-camera, to enable 3D tracking of user's fingers. The mono-camera is used to track user's fingers in the 2D space and ultrasound based sensing is used to get the depth information of user's fingers in the 3D space. Using speakers and microphones that already exist on most AR/VR devices, we emit ultrasound, which is inaudible to humans, and capture the signal reflected by the finger with the microphone. From the phase changes of the ultrasound signal, we accurately measure small finger movements in the depth direction. With fast and light-weight ultrasound signal processing algorithms, our scheme can accurately track finger movements and measure the bending angle of the finger between two video frames. In our experiments on eight users, our scheme achieves a 98.4% finger tapping detection accuracy with FPR of 1.6% and FNR of 1.4%, and a detection latency of 17.69*ms*, which is 57.7*ms* less than video-only schemes. The power consumption overhead of our scheme is 48.4% more than video-only schemes.

CCS CONCEPTS

• **Human-centered computing → Interface design prototyping**; *Gestural input*;

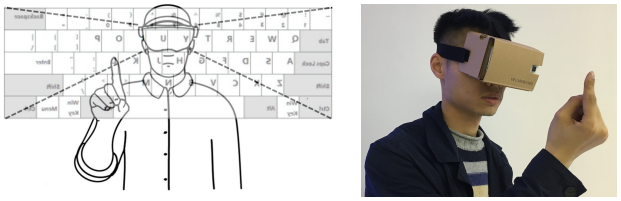
KEYWORDS

Depth aware, Finger tapping, Ultrasound, Computer Vision

ACM Reference Format:
Ke Sun[†], Wei Wang[†], Alex X. Liu[†‡], Haipeng Dai[†]. 2018. Depth Aware Finger Tapping on Virtual Displays. In *Proceedings of MobiSys'18*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3210240.3210315

MobiSys'18, June 10–15, 2018, Munich, Germany

ACM ISBN 978-1-4503-5720-3…$15.00
https://doi.org/10.1145/3210240.3210315



(a) Virtual keypad (b) Cardboard VR setup

Figure 1: Tapping in the air on virtual displays

1 INTRODUCTION

In this paper, we consider to measure the movement depth of in-air tapping gestures on virtual displays. Tapping, which means selecting an object or confirming, is a basic Human Computer Interaction (HCI) mechanism for computing devices. Traditional tapping-based interaction schemes require physical devices such as keyboards, joysticks, mouses, and touch screens. These physical devices are inconvenient for users to interact on virtual displays because users need to hand hold them during the interaction with the AR/VR system, which limits the freedom of user hands in interacting with other virtual objects on the display. For AR/VR systems, *tapping-in-the-air* is a user-friendly solution for interactions. In such schemes, users can input text, open apps, select and size items, and drag and drop holograms on virtual displays, as shown in Figure 1. Tapping-in-the-air mechanisms enrich user experience in AR/VR as user hands are free to interact with other real and virtual objects. Furthermore, fine-grained bending angle measurements of in-air tapping gestures provide different levels of feedbacks, which compensates for the lack of haptic feedback.

Most prior in-air tapping based schemes on virtual displays use customized depth-cameras and therefore have the limitations of low accuracy and high latency. First, most depth-cameras provide depth measurement with a centimeter level accuracy [17, 41], which is inadequate for tapping-in-the-air because tapping gesture often involves small finger movements in the depth direction depending on the finger length and the bending angle of fingers [12]. That explains why they often require users to perform finger movements of several inches, such as touching the index finger with the thumb, to perform a click [22], which leads to much lower tapping speed and low key localization accuracy. Second, the latency of camera based gesture schemes is limited by their frame rate and their high computational requirements. Due to the lack of haptic feedback, interactions with virtual objects are different from interactions with physical keypads, and they solely rely on visual feedback [24]. Visual feedback with a latency of more than 100*ms* is noticeable to

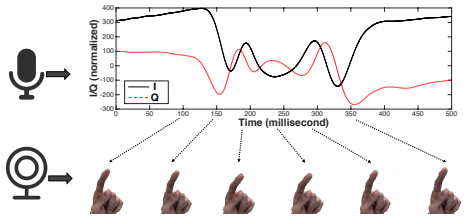


Figure 2: Comparison between video and audio streams

users and degrades user experience [23]; however, it is challenging to provide visual feedback within 100*ms*, because vision-based schemes require a series of high latency operations, such as capturing the video signal, recognizing gestures using computer vision algorithms, and rendering the virtual object on the display. While high-end cameras on smartphones can now provide high speed video capture at more than 120 fps, the high computational costs still limit the processing to a low frame rate in realtime, *e.g.*, 15 fps [43]. This explains why commercial AR systems such as Leap Motion [25] rely on the computational power of a desktop and cannot be easily implemented on low-end mobile devices.

In this paper, we propose a fine-grained depth-aware tapping scheme for AR/VR systems that allows users to tap in-the-air, as shown in Figure 1. Our basic idea is to use light-weight ultrasound based sensing, along with one Commercial Off-The-Shelf (COTS) mono-camera, to enable 3D tracking of users' fingers. To track fingers in the 2D space, the mono-camera is enough for us to achieve that with light-weight computer vision algorithms. To capture the depth information in the 3D space, the mono-camera is no longer sufficient. Prior vision-based schemes require extra cameras and complex computer vision algorithms to obtain the depth information [17, 41]. In this paper, we propose to use light-weight ultrasound based sensing to get the depth information. Using the speakers and microphones that already exist on most AR/VR devices, we emit inaudible sound wave from the speaker and capture the signal reflected by the finger with the microphone. We first use ultrasound information to detect that there exists a finger that performs the tapping down motion, and then use the vision information to distinguish which finger performs the tapping down motion. By measuring the phase changes of the ultrasound signals, we accurately measure fine-grained finger movements in the depth direction and estimate the bending angles of finger tappings. With fast and light-weight ultrasound signal processing algorithms, we can track finger movements within the gap between two video frames. Therefore, both detecting finger tapping motion and updating the virtual objects on virtual display can be achieved within one-video frame latency. This fast feedback is crucial for tapping-in-the-air as the system can immediately highlight the object that is being pressed on user display right after detecting a user tapping motion.

There are three challenges to implement a fine-grained depth-aware tapping scheme. The first challenge is to achieve high recognition accuracy and fine-grained depth measurements for finger tappings. Using either the video or the ultrasound alone is not enough to achieve the desired detection accuracy. For the camera-based approach, the detection accuracy is limited by the low frame-rate where the tapping gesture is only captured in a few video frames. For the ultrasound-based approach, the detection accuracy is limited by the interference of finger movements because it is difficult to tell whether the ultrasound phase change is caused by finger tapping or lateral finger movements. To address this challenge, we combine the ultrasound and the camera data to achieve higher tapping detection accuracy. We first detect the finger movements using ultrasound signal. We then look back at the results of previously captured video frames to determine which finger is moving and the movement direction of the given finger. Our joint finger tapping detection algorithm improves the detection accuracy for gentle finger tappings from 58.2% (camera-only) to 97.6%.

The second challenge is to achieve low-latency finger tapping detection. In our experiments, the average duration of finger tapping gestures is 354*ms*, where the tapping down (from an initial movement to "touching" the virtual key) lasts 152*ms* and the tapping up (moving back from the virtual key to the normal position) lasts 202*ms*. Therefore, a 30-fps camera only captures less than 4 frames for the tapping down gesture in the worst case. However, the feedback should be provided to the user as soon as the finger "touches" the virtual key; otherwise, the user tends to move for an extra distance on each tapping, which slows down the tapping process and worsen user experience. To provide fast feedback, a system should detect finger movements during the tapping down stage. Accurately recognizing such detailed movements in just four video frames is challenging, while waiting for more video frames leads to higher feedback latency. To address this challenge, we use the ultrasound to capture the detailed movement information as shown in Figure 2. We design a state machine to capture the different movement states of user's fingers. As soon as the state machine enters the "tapping state", we analyze both the ultrasound signal and the captured video frames to provide a robust and prompt decision on the tapping event. Thus, our scheme can feedback at the precise timing of "touching", rather than waiting for more frames to see that the finger starts moving back.

The third challenge is to achieve affordable hardware and computational cost on mobile devices. Traditional depth-camera based approaches need dual-camera or extra time-of-flight depth sensors [2, 10]. Furthermore, the computer vision algorithm for 3D fingertip localization incurs high computational costs. It is challenging to achieve 30 fps 3D finger localization, especially on mobile devices such as the Head-Mounted Display (HMD) or mobile phones. To address this challenge, we use speakers/microphones as the depth sensor and combine it with the 2D position information obtained from ordinary mono-camera with light-weight computer vision algorithms. Thus, 3D finger location can be measured using existing sensors on mobile devices with affordable computational costs.

We implemented and evaluated our scheme using commercial smartphones without any hardware modification. Compared to the video-only scheme, our scheme improves the detection accuracy for gentle finger tappings from 58.2% to 97.6% and reduces the detection latency by 57.7*ms*. Our scheme achieves 98.4% detection accuracy with FPR of 1.6% and FNR of 1.4%. Furthermore, the fine-grained bending angle measurements provided by our scheme enables new dimensions for 3D interaction as shown by our case study. However, compared to a video-only solution, our system incurs a significant power consumption overhead of 48.4% on a Samsung Galaxy S5.

System	Sensing methods	Sensors	Range	Depth arruracy	Iteraction
Kinect v1[21, 34]	Light Coding	IR projector&IR camera	$0.8 \sim 4m$	about $4cm$	Human pose
Kinect v2[21, 34]	Time of Flight	IR projector&IR camera	$0.5 \sim 4.5m$	about $1cm$	Human pose
Leap Motion[25, 41]	Binocular camera	IR cameras&IR LEDs	$2.5 \sim 60cm$	about $0.7mm$	Hand track and gesture
Hololens[22]	Time of Flight	IR projector&IR camera	$10 \sim 60cm$	about $1cm$	Hand gesture and gaze
RealSense[15]	Light Coding	IR projector&IR camera	$20 \sim 120cm$	about $1cm$	Hand track and gesture
Air+Touch[8]	Infrared image	IR projector&IR camera	$5 \sim 20cm$	about $1cm$	Single finger gesture
Our scheme	**Phase change**	**Microphone&mono-camera**	**$5 \sim 60cm$**	**$4.32mm$**	**Hand track and gesture**

Table 1: Existing interface schemes for augmented reality systems

2 RELATED WORK

Related work can be categorized into four classes: AR/VR gesture recognition, in-air tapping-based interaction on virtual displays, tapping based interaction for mobile devices, and device-free gesture recognition and tracking.

AR/VR Gesture Recognition: Most existing AR/VR devices use IR projectors/IR cameras to capture the depth information for gesture recognition based on structured light [2] or time of flight [10], as shown in Table 1. Structured light has been widely used for 3D scene reconstruction [2]. Its accuracy depends on the width of the stripes used and their optical quality. A time-of-flight camera (ToF camera) [10] is a range imaging camera system that resolves distance based on the time-of-flight measurements of a light signal between the camera and the subject for each point of the image. However, neither of them focuses on moving object detection and they often incur high computational cost. There are other interaction schemes, including gaze-based interactions [33], voice-based interactions [4, 46], and brain-computer interfaces [32]. However, tapping on virtual buttons is one of the most natural ways for users to input text on AR/VR devices.

In-air Tapping-based Interaction on Virtual Displays: Existing interaction schemes for VR/AR environments are usually based on in-air tapping [14, 15, 21, 22, 25, 42]. Due to the high computational cost and low frame rate, commercial schemes are inconvenient for users [15, 21, 22, 25]. Higuchi *et al.* used 120 fps video cameras to capture the gesture and enable a multi-finger AR typing interface [14]. However, due to the high computational cost, the video frames are processed on a PC instead of the mobile device. Comparing with such systems, our scheme uses a light-weight approach that achieves high tapping speed and low latency on widely available mobile devices.

Tapping Based Interaction for Mobile Devices: Recently, various novel tapping based approaches for mobile devices have been proposed, such as camera-based schemes [26, 43], acoustic signals based schemes [18, 37], and Wi-Fi based schemes [3, 6]. These approaches focus on exploring alternatives for tapping on the physical materials in the 2D space [3, 6, 18, 26, 37, 43]. In comparison, our approach is an in-air tapping scheme addressing the 3D space localization problem, which is more challenging and provides more flexibility for AR/VR.

Device-free Gesture Recognition and Tracking: Device-free gesture recognition is widely used for human-computer interaction, which mainly includes vision-based [8, 21, 22, 25, 35, 45], RF-based [1, 11, 16, 20, 31, 36, 39, 40] and sound-based [7, 13, 27, 38, 44]. Vision based systems have been widely used in AR/VR systems that have enough computational resources [8, 21, 22, 25, 35]. However, they incur high computational cost and have limited frame rates so that they cannot be easily ported to mobile devices. RF based systems use the radio waves reflected by hands to recognize predefined gestures [1, 7, 13, 16, 20]. However, they cannot provide high accuracy tracking capability, which is crucial for in-air tappings. In comparison, our scheme provides fine-grained localization for fingertips and can measure the bending angle of the moving finger. Sound-based systems, such as LLAP [38] and Strata [44] , use phase changes to track hands and achieve cm-level accuracy for 1D and 2D tracking, respectively. FingerIO [27] proposes an OFDM based hand tracking system and achieves a hand location accuracy of $8mm$ and allows 2D drawing in the air using COTS mobile devices. However, both schemes treat the hand as a single object and only provide tracking in the 2D space. The key advantage of our scheme is on achieving fine-grained multi-finger tracking in the 3D space as we fuse information from both ultrasound and vision.

3 SYSTEM OVERVIEW

Our system is a tapping-in-the-air scheme on virtual displays. It uses a mono-camera, a speaker, and two microphones to sense the in-air tapping. The camera captures the video of users' fingers at a speed of 30 fps, without the depth information. The speaker emits human inaudible ultrasound at a frequency in the range of $18 \sim 22\,kHz$. The microphones capture ultrasound signals reflected by users' fingers to detect finger movements. The system architecture consists of four components as shown in Figure 3.

Fingertip Localization (Section 4): Our system uses a light-weight fingertip localization algorithm in video processing. We first use skin color to separate the hand from the background and detects the contour of the hand, which is a commonly used technique for hand recognition [30]. Then, we use a light-weight algorithm to locate all the fingertips captured in the video frame.

Ultrasound Signal Phase Extraction (Section 5): First, we down convert the ultrasound signal. Second, we extract the phase of the reflected ultrasound signal. The ultrasound phase change corresponds to the movement distance of fingers in the depth direction.

Tapping Detection and Tapping Depth Measurement (Section 6): We use a finite state machine based algorithm to detect the start of the finger tapping action using the ultrasound phase information. Once the finger tapping action is detected, we trace back the last few video frames to confirm the tapping motion. To measure the strength of tapping, we combine the depth acquired from the ultrasound phase change with the depth acquired from the video frames to get the bending angle of the finger.

Keystroke Localization (Section 7): When the user tries to press a key, both the finger that presses the key and the neighboring fingers will move at the same time. Therefore, we combine the tapping depth measurement with the videos to determine the finger that has the largest bending angle to recognize the pressed key.

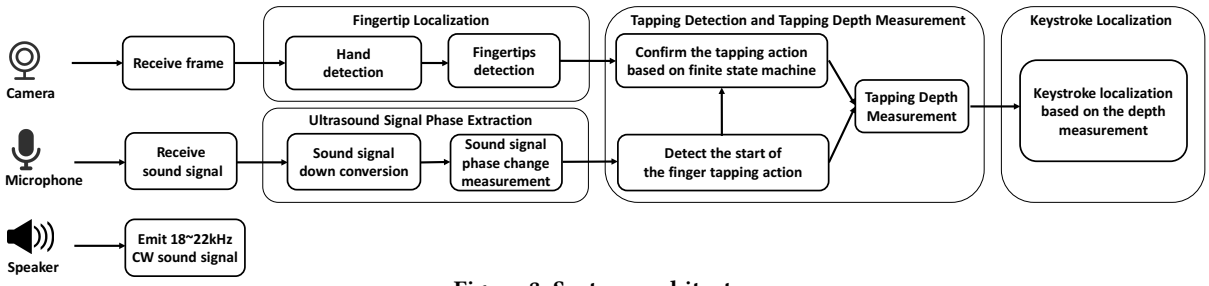


Figure 3: System architecture

4 FINGERTIPS LOCALIZATION

In this section, we present fingertip localization, the first step of video processing. We use light-weight computer vision algorithm to locate the fingertips in the horizontal 2D space of the camera.

4.1 Adaptive Skin Segmentation

Given a video frame, skin segmentation categorizes each pixel to be either a skin-color pixel or a non-skin-color pixel. Traditional skin segmentation methods are based on the YUV or the $YCrCb$ color space. However, surrounding lighting conditions have impacts of the thresholds for Cr and Cb. We use an adaptive color-based skin segmentation approach to improve the robustness of the skin segmentation scheme. Our scheme is based on the Otsu's method for pixel clustering [29]. In the $YCrCb$ color space, we first isolate the red channel Cr, which is vital to human skin color detection. The Otsu's method calculates the optimal threshold to separate the skin from the background, using the grayscale image in the Cr channel. However, the computational cost of Otsu's method is high and it costs $25ms$ for a 352×288 video frame when implemented on our smartphone platform. To reduce the computational cost, we use Otsu's method to get the threshold only on a small number of frames, *e.g.*, when the background changes. For the other frames, we use the color histogram of the hand region learned from the previous frame instead of the Otsu's method. Note that although our color-based skin segmentation method can work under different lighting conditions, it is still sensitive to the background color. When the background color is close to the skin color, our method may not be able to segment the hand successfully.

4.2 Hand Detection

We perform hand detection using the skin segmentation results, as shown in Figure 4(b). We first reduce the noise in the skin segmentation results using the erode and dilate methods. After that, we use a simplified hand detection scheme to find hand contour.

Our simplified detection scheme is based on the following observations. First, in the AR scenario, we can predict the size of the hand in the camera view. As the camera is normally mounted on the head, the distance between the hand and the camera is smaller than the length of the arm. Once the full hand is in the view, the size of the hand contour should at least be larger than a given threshold. Such threshold can be calculated through the statistics of human arm length [12] and the area of palm. Therefore, we only need to perform hand contour detection when there are skin areas larger than the given threshold. Second, the hand movement has a limited speed so that we can use the centroid of the hand to track the movement under 30 fps frame rate. After determining that one of the large contours in the view is the hand, we retrieve the point that has the maximum distance value from the Distance Transform [5] of the segmentation image to find the centroid of the palm, as shown in Figure 4(c). We trace the centroid of the hand rather the entire contour. This significantly simplifies the tracing scheme because the centroid normally remains within the hand contour captured in the last frame due to that the hand movement distance should be smaller than the palm size between two consecutive frames.

4.3 Fingertip Detection

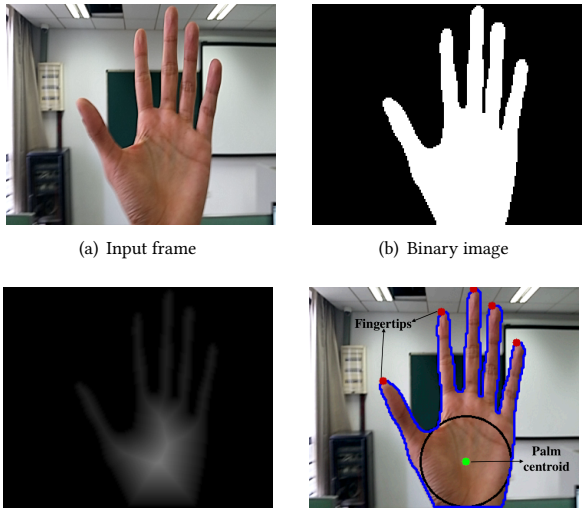
We then detect the fingertips using the hand contour when the user makes a tapping gesture. Our model is robust to detect fingertips' location with different numbers of fingers. As shown in Figure 5, we present the most complex situation of a tapping gesture with five fingertips. Traditional fingertip detection algorithms have high computational cost, as they detect fingertips by finding the convex vertex of the contour. Consider the case where the points on the contour are represented by P_i with coordinates of (x_i, y_i). The curvature at a given point P_i can be calculated as:

$$\theta_i = \arccos \frac{\overrightarrow{P_iP_{i-q}} \overrightarrow{P_iP_{i+q}}}{\|\overrightarrow{P_iP_{i-q}}\| \|\overrightarrow{P_iP_{i+q}}\|} \quad (1)$$

where P_{i-q} and P_{i+q} are the q^{th} point before/after point P_i on the contour, $\overrightarrow{P_iP_{i-q}}$ and $\overrightarrow{P_iP_{i+q}}$ are the vectors from P_i to P_{i-q} and P_{i+q}, respectively. The limitation of this approach is that we have to go through all possible points on the hand contour. Scanning through all points on the contour takes $42ms$ on smartphones on average in our implementation. Thus, it is not capable to achieve 30 fps rate.

To reduce the computational cost for fingertip detection, we first compress the contour into segments and then use a heuristic scheme to detect fingertips. Our approach is based on the observations that while tapping, people usually put their hand in front of the camera with the fingers above the palm as shown in Figure 5. This gesture can serve as an initial gesture to reduce the effort of locating the fingertips. Under this gesture, we can segment the contour by finding the extreme points on the Y axis as shown in Figure 5. The four maximum points, R_2, R_4, R_5 and R_6 correspond to the roots of fingers. Using this segmentation method, we just need to consider these extreme points while ignoring the contour points in between to reduce the computational costs.

Although the extreme-points-based scheme is efficient, it might lead to errors as the hand contour could be noisy. We use the



(a) Input frame (b) Binary image

(c) Hand contour distance transform image (d) Fingertips image

Figure 4: Adaptive fingertip 2D localization

geometric features of the hand and the fingers to remove these noisy points on the hand contour. First, the fingertips should be above the palm, shown as the black circle in Figure 4(d). Suppose that $C(x'', y'')$ is the centroid of the palm calculated by the Distance Transform Image.

We check that all the fingertips points F_i, with coordinates of (x_i, y_i), should satisfy:

$$y_i < y'' - r, \ \forall i \in \{1, 2, 3, 4, 5\}. \tag{2}$$

Second, the length of the fingers, including the thumb, is three times than its' width [48]. We can calculate the width of fingers by:

$$w_i = \begin{cases} \|\overrightarrow{R_1R_2}\|, & \text{if } i \in \{1\} \\ \|\overrightarrow{R_{i+1}R_{i+2}}\|, & \text{if } i \in \{2, 3, 4, 5\} \ . \end{cases} \tag{3}$$

The lengths of the fingers are

$$l_i = \begin{cases} \left\| \dfrac{\overrightarrow{R_1F_1}\|\overrightarrow{R_1R_2}\|^2 - \overrightarrow{R_1R_2}\overrightarrow{R_1F_1}\overrightarrow{R_1R_2}}{\|\overrightarrow{R_1R_2}\|^2} \right\|, & \text{if } i \in \{1\} \\ \left\| \dfrac{\overrightarrow{R_{i+1}F_{i+2}}\|\overrightarrow{R_{i+1}R_{i+2}}\|^2 - \overrightarrow{R_{i+1}R_{i+2}}\overrightarrow{R_{i+1}F_{i+1}}\overrightarrow{R_{i+1}R_{i+2}}}{\|\overrightarrow{R_{i+1}R_{i+2}}\|^2} \right\|, & \text{if } i \in \{2, 3, 4, 5\}. \end{cases} \tag{4}$$

We check that all the detected fingertips should satisfy:

$$\frac{l_i}{w_i} > threshold, \ \forall i \in \{1, 2, 3, 4, 5\}. \tag{5}$$

In our implementation, we set the *threshold* to 2.5. The maximum points in the contour that can satisfy both Eq. (2) and Eq. (5) correspond to the fingertips.

As the tapping gesture like Figure 5 recur frequently during tapping, we calibrate our fingertips' number and location when we detect such gestures with different number of fingers. In the case that two fingers are close to each other or there is a bending finger, we use the coordinates of fingertips on the x axis to interpolate

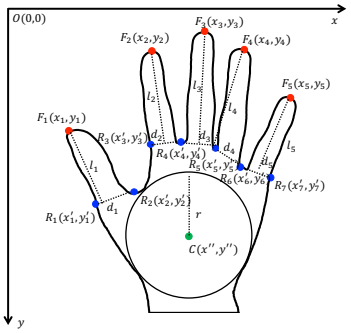


Figure 5: Hand geometric model

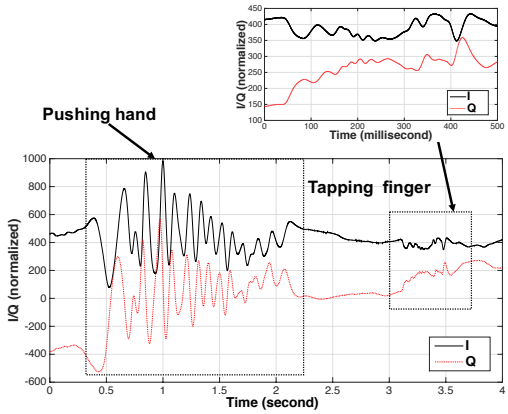
the fingertip locations. Note that our finger detection algorithm focuses on the case for tapping. It might not be able to detect all fingers when the fingers are blocked by other parts of the hand.

5 DEPTH MEASUREMENT

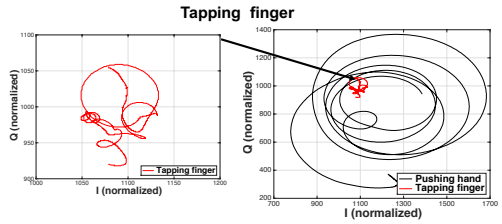
We use the phase of ultrasound reflected by the fingers to measure finger movements. This phase-based depth measurement has several key advantages. First, ultrasound based movement detection has low latencies. It can provide instantaneous decision of the finger movement between two video frames. Second, ultrasound based movement detection gives accurate depth information, which helps us to detect finger tappings with a short movement distance.

Existing ultrasound phase measurement algorithms, such as LLAP [38] and Strata [44], cannot be directly applied to our system. This is because they treat the hand as a single object, whereas we detect finger movements. The ultrasound signal changes caused by hand movements are much larger than that caused by the finger movements and the multipath interference in finger movements is much more significant than hand movements. As illustrated in Figure 6, the user first pushes the whole hand towards the speaker/microphones and then taps the index finger. The magnitude of signal change caused by hand movement is 10 times larger than that of tapping a single finger. Furthermore, we can see clear regular phase changes when moving the hand in Figure 6. However, for the finger tapping, the phase change is irregular and there are large direct-current (DC) trends during the finger movements caused by multipath interference. This makes the depth measurement for finger tapping challenging.

To rule out the interference of multipath and measure the finger tapping depth under large DC trends, we use a heuristic algorithm called Peak and Valley Estimation (PVE). The key difference between PVE and the existing LEVD algorithm [38] is that PVE specifically focuses on tapping detection and avoids the error-prone step of static vector estimation in LEVD. As shown in Figure 6, it is difficult to estimate the static vector for finger tapping because the phase change of finger tapping is not obvious and it is easy to be influenced by multipath interference. To handle this problem, we rely on the peak and valley of the signal to get the movement distance. Each time the phase changes by 2π, there will be two peaks and two valleys in the received signal. We can measure the phase changes of $\pi/2$ by counting the peaks and valleys. For example, when the phase changes from 0 to $\pi/2$, we will find that the signal change from the I component peak to the Q component peak in time domain.



(a) I/Q waveforms



(b) Complex I/Q traces

Figure 6: The difference of phase change between the pushing hand and tapping finger

In order to mitigate the effect of static multipaths, we take two factors into consideration. First, we use the phase magnitude caused by the reflected moving part to remove large movements. As shown in Figure 6, the magnitude of signal change caused by hand movement is 10 times larger than that of taping a single finger. As a result, we set the threshold of the magnitude gap between the adjacent peak and valley to isolate the finger movement from other movements, which is called "FingerInterval". Second, there are many fake extreme points as shown in Figure 7, which are caused by the noise of static vector. We use the speed of the finger tappings to exclude the fake extreme points. As shown in Figure 8(d), the finger tapping only lasts 150ms on average. We can estimate the speed of the path length change of finger tappings. As the ultrasound phase changes by 2π whenever the movement distance causes a path length change equal to the ultrasound wavelength, we set the threshold of the time duration of $\pi/2$ phase change, which is called "SpeedInterval" in PVE. Using this model, we can exclude fake extreme points in the signal: if the interval between two continuous extreme points in I/Q component is beyond the scope of "SpeedInterval", we will treat it as an fake extreme point. Note that this approach only helps us to measure the phase change of integer multiple of $\pi/2$, it can estimate the distance with a granularity of about 5 mm. To further reduce the measurement error, we use the peak and valley near the beginning and end to estimate the phase change in the beginning and end of the phase change. We use the sum of last valley and peak of each component as the static vector to estimate the beginning and ending phases. To mitigate dynamic multipaths, we also combine the results of different frequencies using linear regression.

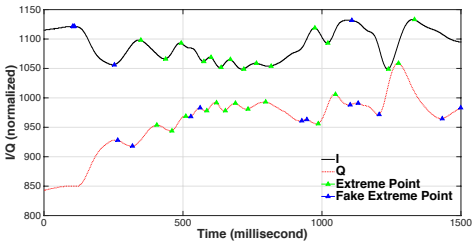


Figure 7: Peak and valley estimate

6 FINGER TAPPING DETECTION

In this section, we present the finger tapping detection algorithm which combines the information captured by the camera and microphones to achieve better accuracy.

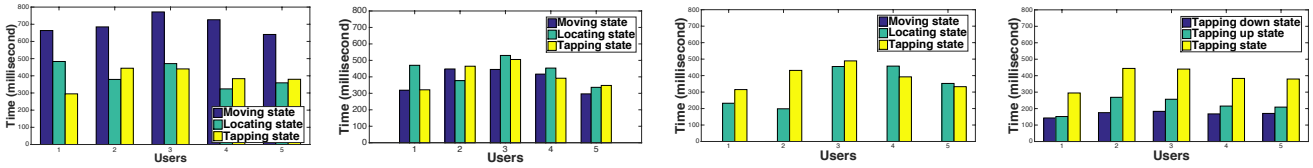
6.1 Finger Motion Pattern

Tapping-in-the-air is slightly different from tapping on the physical devices. Due to the absence of haptic feedback from the physical keys [9], it is hard for the user to perform concurrent finger tappings in-the-air and resolve the typing sequence using visual feedback. Furthermore, on virtual keypads, the users should first move their hand to locate the key then tap from the top of the key. As a result, we mainly focus on supporting one finger/hand typing in this work. We leave two hand typing as our future work.

We divide the finger movement during the tapping-in-the-air process into three parts. The first state is the "moving state", during which the user moves their finger to the key that he/she wants to press. During this state, the movement pattern of the fingers and hands is quite complex, due to the various ways to press different keys on virtual displays. It is difficult to build a model for the video and ultrasound signals in this state. Therefore, we just detect the state without wasting computational resources and energy in analyzing the complex pattern. The second state is the "locating state", where the user keeps their finger on the target key position briefly before tapping it. Although this state can hardly be perceived by human beings, this short pause can be clearly detected by the ultrasound or the 120 fps video. The average duration of the "locating state" is 386.2ms as shown in Figure 8. During this state, both video and audio signals remain static for a short interval, because the finger is almost static. The third state is the "tapping state", where the user slightly moves their finger up and down to press the key. In order to detect the finger tap, we divide the "tapping state" into two states, the "tapping down state", and the "tapping up state".

We use RM-ANOVA to analyze the motion pattern of in-air finger tappings. Five volunteers participated in our user study. Each user taps on the virtual QWERTY keyboard in AR environments with a single index finger for five minutes. The virtual keyboard is rendered on the screen of the smartphone. Since the resolution of the smartphone used in our experiments is 1920×1080, we set the size of the virtual keys as 132×132 pixels.

We use 120 fps video camera to capture in-the-air tapping procedure and do offline computer vision process to analyze the users' behavior. The offline analysis is manually verified to remove incorrect state segments. The statistical results for the user study are shown in Figure 8. In general, the process of tapping a single key on the virtual display will go through all of the three states. However, we still find three different types of patterns. The first pattern corresponds to the



(a) Tapping non-adjacent keys (b) Tapping adjacent keys (c) Tapping the same key (d) Different motion states

Figure 8: Duration of states in different motions

case when the user is tapping a key that is not adjacent to the last key. The duration of the three states in this case are shown in Figure 8(a). The average duration of "moving state", "locating state", and "taping state" is 697.4ms (SD = 198.4ms), 403.2ms (SD = 36.6ms), and 388.4ms (SD = 32.4ms), respectively. The second pattern corresponds to the case when the user is tapping a neighboring key which is adjacent to the last key previously tapped. In this case, the average duration of "moving state", "locating state", and "tapping state" is 385.1ms (SD = 85.4ms), 433.4ms (SD = 37.4ms), and 406.2ms (SD = 32.2ms), respectively. We observe that the average duration of "moving state" drops significantly. In some samples, the "moving state" may even totally disappear, because the fingertip is close to the expected key and the user directly moves the finger while tapping. The third pattern corresponds to the case when the user is repeatedly tapping the same key. In this case, the "moving state" is always missing, as shown in Figure 8(c). The average duration of the "locating state" also drops significantly, because the user doesn't need to adjust the location of the fingertip when they are tapping the same key repeatedly. In some samples, the latter two cases still have the same patterns as the first case. This is mainly due to the randomness in the tapping process, especially when the user is not familiar with the QWERTY keyboard. Figure 8(d) shows the duration for the "tapping down state" and "tapping up state". We observe that the average "tapping down state" duration is just 168.1ms so that it is difficult to use 30 fps video to determine the exact time of the finger to touch the virtual key.

6.2 Finger Tapping Detection

Our finger tapping detection algorithm is based on the state machine as shown in Figure 9. We divide the detection process into three stages. In the first stage, we use the ultrasound to detect that the motion state enters the "tapping state". This is because that ultrasound has much higher sampling rate compared to the video and it is more sensitive to the motion in the depth direction. As the ultrasound may have high false positive rates, we invoke the video processing once motion is detected. Therefore, in the second stage, the video process will look back to the previous frames captured by the video to measure the duration of "moving state" and "locating state". We check if these states satisfy the state machine as shown in Figure 9. This helps us to remove false alarms introduced by ultrasound-based detection. Finally, in the third stage, we use the nearest-neighbor algorithm to determine the pressed virtual key, based on the fingertip location during the "locating state".

In our design, we try to strike a balance between the robustness of finger tapping detection and the delay of the detection algorithm. On one hand, to improve the robustness of finger tap detection, we use the state machine to confirm the finger tapping. On the other

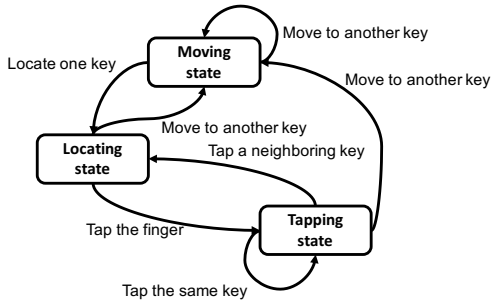
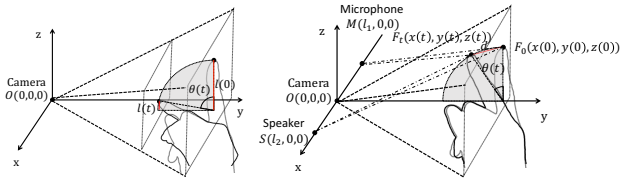


Figure 9: State machine of finger tapping detection



(a) Video model (b) Audio model

Figure 10: Geometric model

hand, we reduce the delay for displaying the finger tap result by using the ultrasound-based detection. Once we detect the ultrasound phase change in "tapping down state", the tapping action is confirmed by the previous video frames and the result can be rendered in the next output display frame. As a result, the upper bound for detection delay is one video frame, which is about 41.7ms for 24 fps video stream.

6.3 Determining the Depth of Finger Tapping

After detecting the tapping action, we measure the fine-grained depth information for the tapping. With the depth information for tapping, we can measure the tapping strength to improve users' visual feedback. Meanwhile, we can design different keys for different tapping depth, which will improve users' input speed. For example, we can use two different tapping depths to input lower-case letters and capital letters. The finger tapping depth is represented by the bending angle of the finger, which is denoted as $\theta(t)$, as shown in Figure 10.

Deep finger tapping: When the finger tapping is performed with a large bending angle, the duration of the tapping is longer. Therefore, the video camera can capture more frames during the tapping. Furthermore, deep finger tappings introduce large finger length changes in the y axis on the video frames, as shown in Figure 10(a). As a result, we use the camera-based model to measure $\theta(t)$

for deep finger tappings. Suppose that the initial finger length is $l(0)$ at time 0 and the shortest finger length during the tapping is $l(t)$ at time t. Then, the bending angle of finger is given by:

$$\theta(t) = \arccos \frac{l(t)}{l(0)}. \tag{6}$$

Gentle finger tapping: When the bending angle of finger is small, the duration of the finger tap is short and the camera is not able to capture enough video frames during the tapping. Furthermore, due to the small bending angle of finger, the finger length change can hardly been detected by the video frame. The finger length changes less than 10 pixels in 352×288 resolution. Therefore, we use the ultrasound phase change to estimate the bending angle for gentle finger tapping. The propagation path change during the finger tap from time $0 \sim t$ can be measured by the phase change as:

$$\Delta d = d(t) - d(0) = -\frac{\varphi_d(t) - \varphi_d(0)}{2\pi}\lambda, \tag{7}$$

where λ is the wavelength of the ultrasound, $\varphi_d(0)$ and $\varphi_d(t)$ are the initial and the final phase of the ultrasound, respectively. However, the propagation path change is different from the depth of finger tap. As shown in Figure 10(b), the propagation path change during the finger tap is

$$\Delta d = \left|\overrightarrow{SF_0}\right| + \left|\overrightarrow{F_0M}\right| - \left|\overrightarrow{SF_t}\right| - \left|\overrightarrow{F_tM}\right|, \tag{8}$$

where $M(l_1, 0, 0)$ is the location of the microphone, $S(l_2, 0, 0)$ is the location of the speaker, $F_0(x(0), y(0), z(0))$ is the location of the fingertip at time 0, $F_t(x(t), y(t), z(t))$ is the location of the fingertip at time t, and d is the euclidean distance between F_0 and F_t, respectively. According to the triangle inequality, $d > \Delta d/2$. Meanwhile, the different locations of the finger have different $F_0(x(0), y(0), z(0))$, which will result in different lengths of Δd given the same finger tapping depth of d. When users are tapping slightly, we can assume that $x(0) \approx x(t)$ and $z(0) \approx z(t)$ during the finger tap. As a result, the final position is $F_t(x(0), y(0) - d, z(0))$ and we can get the relationship between d and Δd as:

$$\begin{aligned}\Delta d = &\left|\overrightarrow{(x(0) - l_2,\, y(0),\, z(0))}\right| + \left|\overrightarrow{(l_1 - x(0),\, -y(0),\, -z(0))}\right| \\ &- \left|\overrightarrow{(x(0) - l_2,\, y(0) - d,\, z(0))}\right| - \left|\overrightarrow{(l_1 - x(0),\, d - y(0),\, -z(0))}\right| \\ &(\Delta d > d > \Delta d/2).\end{aligned} \tag{9}$$

In Eq. (6), we get $x(0)$ and $z(0)$ from the locations of fingertips in Section 4 and set the parameter $y(0)$ adaptively based on the finger size in the frame. As a result, we can get d from Δd by Eq. (9) by compensating the different location of F_0. Consequently, the bending angle is given by:

$$\theta(t) = 2\arccos \frac{d/2}{l(0)}. \tag{10}$$

7 KEYSTROKE LOCALIZATION

The final step is to map the finger tapping to the virtual key that is pressed by the user. When the user only uses a single finger to perform tapping gestures, we can determine the identity of the virtual key with very low cost. The identity can be determined by calculating the location of the moving fingertip during the "locating state".

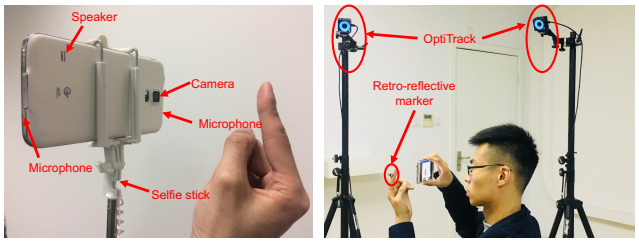
Locating the keystroke when the user uses multiple fingers to perform tapping gesture is quite challenging. This is because that more than one finger will move at the same time, even if the user only intends to use a single finger to press the key. For example, for most people, when they press their little finger, the ring finger will move together at the same time. Therefore, both the video and the ultrasound will detect multiple fingers moving at the same time. To determine the exact finger that is used for pressing, we use the depth information measured in Section 6.3. The finger used for pressing the key always has larger bending angle than other moving fingers. Therefore, we calculate the bending angle for all moving fingers in the view, using the geometric model as shown in Figure 10(a). The finger with the largest bending angle is determined as the pressing finger. Once we confirm which finger is the pressing finger, we use the same methods as in the single-finger case to locate the keystroke.

8 EXPERIMENTAL RESULTS

8.1 Implementation and Evaluation Setup

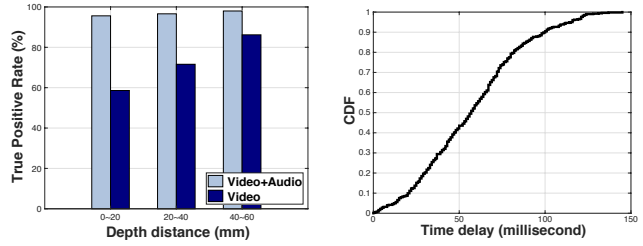
We implemented our system on both the Android and MacOS platforms. On the Android platform, our implementation works as an APP that allows the user to tap in the air in realtime on recent Android devices, *e.g.*, Samsung Galaxy S5 with Android 5.0 OS. For the video capturing, due to the limitations in hardware, the maximum video frame rate is 30 fps. To save the computational resource, we set the video resolution to 352×288 and the average video frame rate under this setting is 25 fps. We emit continuous wave signal of $A\cos 2\pi f t$, where A is the amplitude and f is the frequency of the sound, which is in the range of $17 \sim 22\,kHz$. For audio capturing, we chose a data segment size of 512 samples in our implementation, which has time duration of $10.7ms$ when the sampling rate is $48kHz$. We implemented most signal processing and computer vision processing algorithms as C/C++ functions using Android NDK to achieve better efficiency. We used the opening library OpenCV C++ interfaces in computer vision processing when implementing on the Android platform. On the MacOS platform, we implemented the system using the camera on the MacBook and streamed the audio signal using a smartphone. The MacOS-based implementation uses C++. On the laptop, both the video and virtual keyboard are displayed in real time on the screen. The user operates in front of the lap top screen. To obtain the ground truth in our user study, we also captured the user movement by a 120 fps camera. The high speed video was processed offline and manually annotated to serve as the ground truth. Due to the speaker placement (near the ear, instead of facing forward) and SDK limitations on commercial AR devices, we are unable to implement our system on existing devices, such as the Hololens [22]. Instead, we use a cardboard VR setup as shown in Figure 1(b) in our case study.

We conducted experiments on Samsung Galaxy S5 smartphone, using its rear speaker, two microphones and the rear camera in both office and home environments, as shown in Figure 11. Experiments were conducted by eight users, who are graduate students with the age of $22 \sim 26$ years. Five out of the eight users have prior experiences on using VR/AR devices. The users interacted with the phone using their bare hands behind the rear camera without wearing any accessory. The performance evaluation process lasted 90 minutes with 18 sessions of five minutes. There is a five minutes break between two sessions. If not specified, the smartphone was fixed on a selfie stick during the experiments.



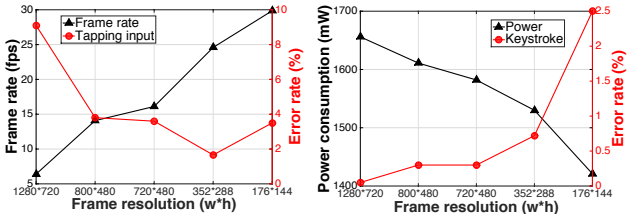
(a) Selfie stick setup (b) OptiTrack setup

Figure 11: Experimental setup



(a) Sensitivity for different tapping depths (b) Latency reduction

Figure 12: Finger tapping detection accuracy



(a) Frame rate and tapping input error rate (b) Power consumption and keystroke localization error rate

Figure 13: Impact of video resolution

8.2 Evaluation Metrics

We evaluated our system in four aspects. First, we evaluated the finger tapping detection accuracy using three metrics: True Positive Rate (TPR), False Positive Rate (FPR), and False Negative Rate (FNR). The TPR is the ratio of detected finger tappings to the number of finger tappings performed by the user; The FPR is defined as the ratio of falsely detected finger tappings to the number of decisions made by our system while the user is not performing a finger tapping; The FNR is the ratio of missed finger tappings to the number of finger tappings performed by the user. In this evaluation, we collected 2,000 finger tappings performed by eight users using the smartphone. Second, we evaluated the impact of video resolution on the performance of our system in real-time system. Third, we evaluated the latency and power consumption of our system, when the real-time system is running on a smartphone. Fourth, we performed two case studies: 1) DolphinBoard: in-the-air text input and 2) DolphinPiano: AR piano based on the finger bending angle. We evaluated the TPR of different users under different environments and the feedback based on different bending angles.

8.3 Finger Tapping Detection

Our system can robustly detect finger tappings with different tapping depths. We evaluate FNR for finger tappings with different depths. The ground truth depth distances are measured by OptiTrack [28], a high-precision motion capture and 3D tracking system. As shown in Figure 11(b), We place the retro-reflective marker on the index finger of the volunteer to achieve 120 fps 3D trace of the finger when they are performing the test. Figure 12(a) shows the detection accuracy for different tapping depths. Since it is hard for volunteers to control their fingers to move for such a small distance, we only test on three different tapping depth bins. Our system achieves 95.6%, 96.6%, and 98% TPR, for tapping depths of $0 \sim 20$, $20 \sim 40$, and $40 \sim 60mm$, respectively, while video based scheme only achieves TPR of 58.6% for the $20mm$ case. The key advantage of introducing the ultrasound is that it can reliably detect gentle finger tappings with a depth of $0 \sim 20mm$. Based on the ground truth captured by OptiTrack, our phase-based depth measurement achieves an average movement distance error of $4.32mm$ (SD = $2.21mm$) for 200 tappings.

Our system achieves an average FPR of 1.6% *and FNR of* 1.4% *for gentle finger tappings.* We evaluate FPR/FNR for gentle finger tappings with bending angle of 30 degrees. The single video camera based finger tapping detection has FPR of 1.2% and FNR of 41.8% for gentle finger tappings. The video-only scheme has much higher FNR because it cannot reliably detect gentle finger tappings. On the contrary, pure audio-based scheme has average FPR of 28.2% and FNR of 2.4%. The higher FPR for audio based scheme is because ultrasound often raises false alarms for other tappings of finger motions, such as finger movements. By combining the video with the audio, we take advantage of both of them to achieve low FPR and FNR at the same time.

On average, the finger tapping detection latency of our system is $57.7ms$ *smaller than the video-based schemes, which is equivalent to two frames in the 30 fps camera.* Figure 12(b) show the Cumulative Distribution Function (CDF) of the interval between the time that our system detects the tapping and that the video-based scheme detects the tapping, for 500 finger tappings. For 80% of the instances, our system can detect the finger tapping $33.5ms$ earlier than video-based schemes, which is equivalent to one frame in 30 fps camera.

Based on experimental results, we choose a video resolution of 352×288 *in our Android implementation.* Most mobile devices support different video resolutions, from 1280×720 to 176×144. The video resolution has different impact on various performance metrics, including frame rate, energy consumption, keystroke localization accuracy, and the tapping input FNR. A higher video resolution, such as 1280×720, often leads to lower frame rate, due to the hardware constraints of the video camera and the computational cost for higher video resolution. As shown in Figure 13(a), our Samsung Galaxy S5 can only support a video stream rate of 10 fps when the resolution is 1280×720. Higher video resolution also leads to higher energy consumption. We use Powertutor [47] to measure the power consumption under different video resolutions and the result is shown in Figure 13(b). We observe that there is a sharp drop in power consumption for the lowest resolution of 176×144, due to the sharp decrease in the computational cost.

(a) Audio thread

	Down conversion	PVE	Tapping detection	Total
Time	$6.455ms$	$0.315ms$	$0.036ms$	$6.806ms$

(b) Video thread

	Hand detection	Fingertip detection	Frame playback	Total
Time	$22.931ms$	$2.540ms$	$14.593ms$	$40.064ms$

(c) Control thread

	Keystroke localization	Virtual key rendering	Total
Time	$0.562ms$	$10.322ms$	$10.884ms$

Table 2: Processing time

	CPU	LCD	Audio	Total
Idle	$30 \pm 0.2mW$	/	/	$30 \pm 0.2mW$
Backlight	$30 \pm 0.2mW$	$894mW \pm 2.3$	/	$924 \pm 2.0mW$
Video-only	$140 \pm 4.9mW$	$895 \pm 2.2mW$	/	$1035 \pm 4.0mW$
Our scheme	$252 \pm 12.6mW$	$900 \pm 5.7mW$	$384 \pm 2.7mW$	$1536 \pm 11.0mW$

Table 3: Power consumption

However, low resolution of 176×144 cannot support accurate keystroke localization, as shown in Figure 13(b). The probability that our system gives a wrong keystroke location raises from nearly zero to 2.5%, when we decrease the resolution from 1280×720 to 176×144. Figure 13(a) shows the overall tapping input FNR, which is defined as the ratio of missed and wrongly identified keys to the total number of keys pressed. We observe that neither the highest nor the lowest resolution has a low tapping input error rate. High video resolution of 1280×720 has FNR of 9.1% due to the low video frame rate, which leads to higher latency in response. Low video resolution of 176×144 has FNR of 3.5% due to the higher error rate in keystroke localization. Therefore, to strike a balance between the latency and the keystroke localization error, we choose to use video resolution of 352×288, which gives a input error rate of 1.7%.

8.4 Latency and Power Consumption

Our system achieves a tapping response latency of 18.08*ms on commercial mobile phones.* We measured the processing time for our system on a Samsung Galaxy S5 with Qualcomm Snapdragon $2.5GHz$ quad-core CPU. Our implementation has three parallel threads: the audio thread, the video thread, and the control thread. The audio thread processes ultrasound signals with a segment size of 512 data samples (with time duration of $10.7ms$ under $48kHz$ sampling rate). The processing time for each stage of the audio thread for one data segment is summarized in Table 2. We observe that the latency for the audio process to detect a finger tapping is just $6.806ms$. The video process performs hand detection, fingertip detection, and video playback. At the resolution of 352×288, the processing latency is $40.06ms$ and our system achieves an average frame rate of 24.96 fps. The control thread performs the keystroke localization and renders the updated virtual keyboard. It has a latency of $10.88ms$. As these three threads run parallelly, the slowest video thread is not in the critical path and we can use the result of previous frames in the other two threads. Therefore, once the audio thread detects the finger tapping, it can evoke the control thread immediately and the total latency between keystroke and rendering of the virtual keyboard is $6.81ms + 10.88ms = 17.69ms$.

User	Gender	Age	Hand Length	Hand width
User1	Male	23	19.0cm	10cm
User2	Male	23	17.8cm	9.3cm
User3	Female	22	14.5cm	7.5cm
User4	Male	25	17.2cm	9.2cm
User5	Male	26	17.5cm	9.4cm
User6	Male	24	18.5cm	10.2cm
User7	Male	24	17.9cm	9.8cm
User8	Male	24	18.2cm	9.5cm

Table 4: Participants information

We use the Powertutor [47] to measure the power consumption of our system on the Samsung Galaxy S5. To measure the power consumption overhead of different individual components, we measured the average power consumption in 4 different states for 25 minutes with 5 sessions of 5 minutes: 1) idle, with the screen off, 2) backlight, with the screen displaying, 3) video-only scheme, with the video-based scheme on, 4) our system, with both the ultrasound and video scheme on. As shown in Table 3, more than 68% of power consumption comes from LCD and CPU which are essential for traditional video-only virtual display applications. Compared to the video-only scheme, the additional power consumptions introduced by our scheme for CPU and Audio are $112mW$ and $384mW$, respectively, which means more than 77% additional power consumption comes from the speaker hardware. Overall, we measured a significant power consumption overhead of 48.4% on commercial smartphones caused by our scheme. One possible future research direction could be further reducing the power consumption of the audio system.

8.5 Case Study

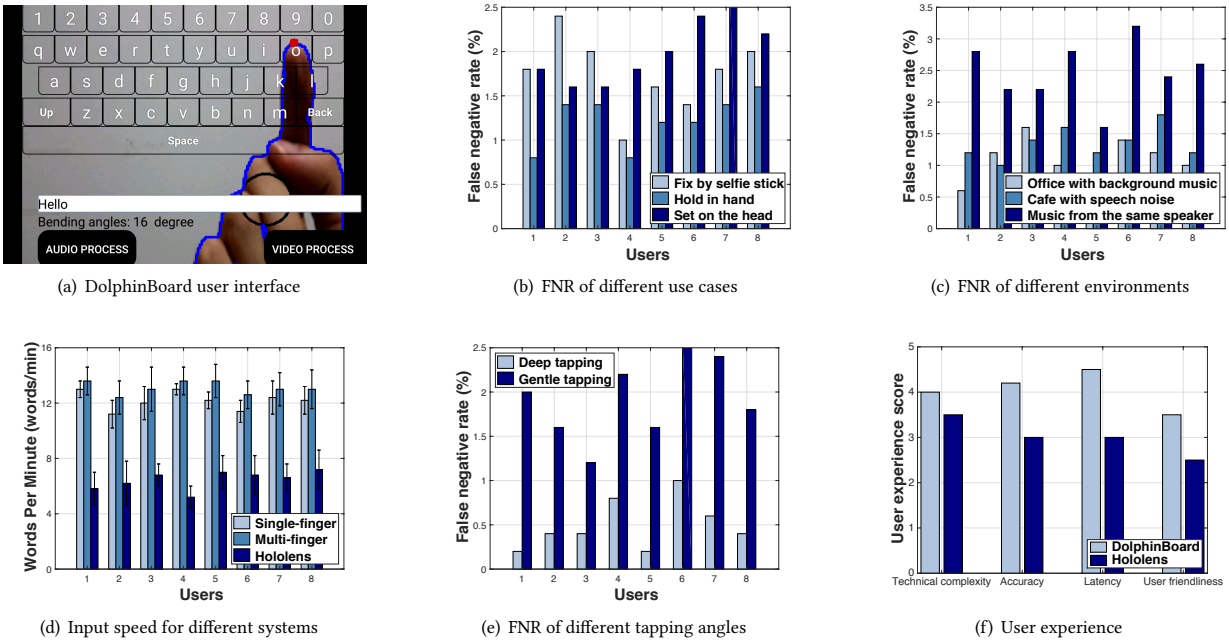
We used our system to develop different applications in AR/VR environments. In order to further evaluate the performance of our system, we conducted two different case studies using real-world settings. As our systems use both visual information and sound reflections for target locating, just as Dolphins, we name the applications as DolphinBoard and DolphinPiano.

8.5.1 DolphinBoard: In-the-air text input. In this case study, the task of DolphinBoard is to enable text input by tapping-in-the-air mechanism. This study aims to evaluate the detect error rate of different users under different environments and the tapping speed.

User interface: Figure 14(a) shows the user interface of DolphinBoard. Users move their finger in-the-air and locate the virtual key on the virtual display to be tapped. The QWERTY virtual keyboard is rendered on the top of the screen with a size of 1320×528 pixels. We set the size of keys as 132×132 pixels for most of the experiments.

Testing Participants: We invited eight graduate student volunteers to use our applications. We marked these users as User $1 \sim 8$. All of them participated in the 90 minutes performance experiments before the use case study. The evaluation of DolphinBoard lasted 20 minutes per person, where users are asked to type a 160-character sentence for text input speed test. Note that a larger hand may generate a stronger echo of the ultrasound signal. Thus, we measured the hand size of each participant as shown in Table 4.

Performance evaluation: *DolphinBoard achieves finger tapping detection error of less than 1.76% under three different use cases.* To evaluate the usability of DolphinBoard, we invited eight users to



(a) DolphinBoard user interface

(b) FNR of different use cases

(c) FNR of different environments

(d) Input speed for different systems

(e) FNR of different tapping angles

(f) User experience

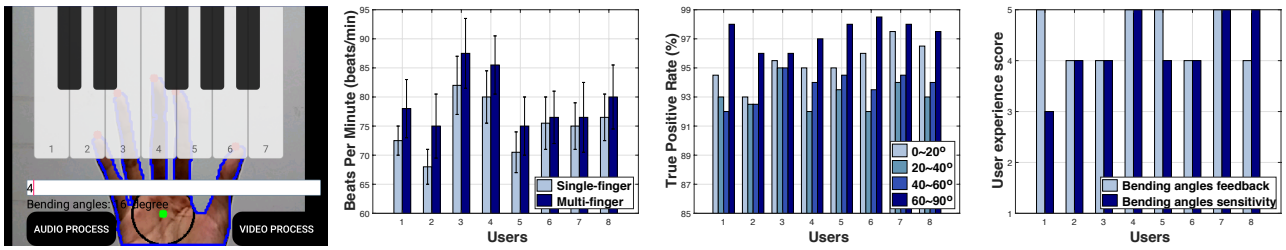
Figure 14: DolphinBoard: In-the-air text input evaluation

input text under three different use cases. The first use case is to hold the smartphone in hand and type behind the phone. The second use case is to fix the smartphone on a selfie stick so that the phone is more stable than in the first use case. The third use case is to put the smartphone in a cardboard VR set worn on the head of the user as shown in Figure 1(b). Each of the users performs 500 taps under the three different user cases. As shown in Figure 14(b), the average FNR for tapping detection are 1.75%, 1.23%, and 2.03%, respectively. This shows that DolphinBoard is robust under small interfering movements of the hands and head. The hand/head movement interferences in use case 1 and 3 only introduce a small increase in the FNR for less than 0.8% compared to the case that the device is fixed.

DolphinBoard is robust to noises and achieves low FNR in three different noisy environments. To evaluate the robustness of DolphinBoard, we asked users to perform finger tapping in three different noisy environments, including a cafe with $60dB$ speech noise, an office with $65dB$ background music, and playing $65dB$ music from the same speaker that is used for playing the ultrasound. In all of these three environments, there are other people walking around. As shown in Figure 14(c), the average FNR for finger tapping detection in the three different environments are 1.1%, 1.35%, and 2.48%, respectively. Note that the ultrasound signals can be mixed with other audible signals. Our system can support the use case that users are using the speaker to play music while performing tapping detection using the ultrasound. When music is played on the same speaker used by DolphinBoard, the intensity of the ultrasound is actually reduced due to the contention of the dynamical range on the speaker. However, DolphinBoard still achieved a low FNR of 2.48% in this challenging scenario.

The text input speed of DolphinBoard is 12.18 (SD=0.85) WPM and 13.1 (SD=1.2) WPM for single-finger and multi-finger inputs, respectively. The text input speed was evaluated using the metrics of Words Per Minute (WPM), which is defined as the number of 5-character words that the user can correctly enter for a duration of one minute. As a reference, the typing speed on DolphinBoard is about two times that on Hololens, on which the users achieve about 6.45 WPM on average as shown in Figure 14(d). The single-finger input speed on DolphinBoard is limited by the finger tapping speed. When we ask the users to tap continuously on the same key, the average tapping speed is about 98 taps per minute, which converts to about 19.6 WPM. While our method supports two-hand tracking, users can only input text with a single hand due to the limited viewing angle of mobile devices. As a result, multi-finger input does not significantly increase the typing speed. The layout of the QWERTY keyboard used in the current design of DolphinBoard could also be a limitation of the text input speed. The users still need to move their hands during typing, which limits the speed. With a better design of a dynamic virtual keyboard, the typing speed of DolphinBoard could be further increased. It is also possible to use the depth information provided by DolphinBoard to activate different virtual keys. As shown in Figure 14(e), the average FNR for DolphinBoard to detect gentle tappings and deep tappings are 0.5% and 1.93%, respectively. Therefore, DolphinBoard can reliably detect different types of tappings and use this information to build better keyboard layouts. For example, the gentle finger tappings could be used for inputting the lower-case letters and the deep finger tappings could be used for inputting the capitalized letters.

User experience evaluation: We evaluated the user experience using 10 participants via questionnaire surveys, including 1)



(a) DolphinPiano user interface (b) Tempo for DolphinPiano (c) TPR of different feedback (d) User experience

Figure 15: DolphinPiano: AR piano evaluation

technical complexity, 2) accuracy, 3) latency and 4) user friendliness (5=strongly positive, 1=strongly negative). Users were asked to type using DolphinBoard and Hololens for 20 minutes per person. Figure 14(f) shows the results. For the technical complexity, some participants have some negative evaluation, this is mainly because DolphinBoard only needs user to move their finger to the expected location instead of moving their head like Hololens. For the accuracy, most of the participants hold positive/strongly positive attitudes because DolphinBoard is able to detect small finger tappings. For the latency, most of the people have positive/strongly positive evaluation since ultrasound-assisted methods shorten the detecting duration effectively. For the user friendliness, the tapping-in-the-air mechanisms are acceptable for most of users. In addition, most participants (9/10) are in favor of the idea of different tapping depths to input lower and upper case letters.

8.5.2 DolphinPiano: AR piano based on the finger bending angle. This study aims at providing feedback based on different bending angles of finger tappings. In this case study, we develop DolphinPiano, which enables users to play the piano in-the-air. For physical pianos, the loudness of the notes depends on the strength of the keystroke: the greater the velocity of a keystroke, the greater the force of the hammer hitting the strings, and the louder the sound of the note produced. Since our system is able to measure the micro movement of the finger, DolphinPiano simulates different levels of force being applied to keys based on the bending angle.

User interface: As shown in Figure 15(a), a piano keyboard with one octave is rendered on the screen. This keyboard is translucent so that users can locate their fingertips to the virtual key on the screen. White keys and black keys have size of 150×640 and 100×320 in pixels, respectively. When users press the key, the application plays the corresponding note with different volume based on the finger bending angle.

Testing Participants: Eight people participated in case study 1 were also invited to use the DolphinPiano application. Since they were familiar with DolphinBoard, they were asked to play a simple piece of music (Jingle Bells) without any training. Each person uses our DolphinPiano application for 20 minutes.

Performance evaluation: *The tempo of DolphinPiano achieves 75.0 (SD=3.88) BPM (beats per minute) and 79.3 (SD=5.31) BPM for single-finger and multi-finger inputs, respectively.* To evaluate the music tempo, eight users were asked to play music on DolphinPiano application as fast as possible. As shown in Figure 15(b), tapping on DolphinPiano is faster than on DolphinBoard, this is mainly because that the keys in DolphinPiano are larger than those in text input which means shorter movement time according to Fitts' Law [19].

DolphinPiano provides accurate feedback for the bending angle of fingers. To evaluate the accuracy of bending angle feedback, we asked the participants to play the piano with different bending angles. Four levels of bending angles were used in the experiement, *i.e.*, $0 \sim 20$, $20 \sim 40$, $40 \sim 60$, $60 \sim 90$ degrees. Participants did not know the ground truth of their bending angles, they could only adjust their movement by the feedback of DolphinPiano. As shown in 15(c), DolphinPiano achieved an average accuracy of 95.4%, 93.2%, 93.8%, and 97.4% TPR for different bending angles, respectively. Participants had a good grasp of bending angle feedback after a short training period.

User experience evaluation: We evaluated the user experience of eight participants via the questionnaire surveys, including 1) feedback accuracy and 2) angle sensitivity (5=strongly positive, 1=strongly negative). Figure 15(d) shows the results. All the participants were positive to bending angle feedback. With the help of bending angle feedback, they could play just like on a physical keyboard. For bending angle sensitivity, they were surprised that DolphinPiano could measure the bending angle by using one camera on the mobile devices. Although there was no tactile feedback, they could play smoothly with the help of different levels of audio and visual feedbacks.

9 CONCLUSIONS

In this paper, we make three key contributions. First, we propose combining ultrasound sensing information and vision information to achieve fine-grained bending angle measurements for in-air finger tappings. Second, we design a tapping-in-the-air scheme with high accuracy and low latency. Third, we implemented and evaluated our system using commercial smartphones without any hardware modification. Our experimental results show that our system achieves high accuracy, low latency and provides feedback based on different bending angles of finger tappings. One limitation of our solution is that its audio system incurs considerable overhead in power consumption. One possible solution for this is to design low-power speakers that are specialized for emitting ultrasounds.

ACKNOWLEDGMENTS

We would like to thank our shepherd Yuvraj Agarwal and anonymous reviewers for their valuable comments. This work is partially supported by National Natural Science Foundation of China under Grant Numbers 61472185, 61472184, 61373129, 61502229, 61672353, and 61472252, the National Science Foundation under Grant Numbers CNS-1421407, Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program.

REFERENCES

[1] Heba Abdelnasser, Moustafa Youssef, and Khaled A Harras. 2015. WiGest: A ubiquitous WiFi-based gesture recognition system. In *Proceedings of IEEE INFOCOM*.

[2] C Albitar, P Graebling, and C Doignon. 2007. Robust Structured Light Coding for 3D Reconstruction. In *Proceedings of IEEE ICCV*.

[3] Kamran Ali, Alex X. Liu, Wei Wang, and Muhammad Shahzad. 2015. Keystroke Recognition Using WiFi Signals. In *Proceedings of ACM MobiCom*.

[4] Amazon. 2017. Alexa. https://developer.amazon.com/alexa. (2017).

[5] Gunilla Borgefors. 1986. Distance transformations in digital images. *Computer Vision Graphics & Image Processing* (1986).

[6] Bo Chen, Vivek Yenamandra, and Kannan Srinivasan. 2015. Tracking Keystrokes Using Wireless Signals. In *Proceedings of ACM MobiSys*.

[7] Ke-Yu Chen, Daniel Ashbrook, Mayank Goel, Sung-Hyuck Lee, and Shwetak Patel. 2014. AirLink: sharing files between multiple devices using in-air gestures. In *Proceedings of ACM UbiComp*.

[8] Xiang Anthony Chen, Julia Schwarz, Chris Harrison, Jennifer Mankoff, and Scott E. Hudson. 2014. Air+touch:interweaving touch & in-air gestures. In *Proceedings of ACM UIST*.

[9] Patricia Ivette Cornelio Martinez, Silvana De Pirro, Chi Thanh Vi, and Sriram Subramanian. 2017. Agency in mid-air interfaces. In *Proceedings of ACM CHI*.

[10] Yan Cui, Sebastian Schuon, Derek Chan, Sebastian Thrun, and Christian Theobalt. 2010. 3D shape scanning with a time-of-flight camera. In *Proceedings of IEEE CVPR*.

[11] Google. 2016. Google Project Soli. https://www.google.com/atap/project-soli/. (2016).

[12] Claire C Gordon, Cynthia L Blackwell, Bruce Bradtmiller, Joseph L Parham, Jennifer Hotzman, Stephen P Paquette, Brian D Corner, and Belva M Hodge. 1989. 2010 Anthropometric Survey of U.S. Marine Corps Personnel: Methods and Summary Statistics. *Anthropometric Survey of U.s.army Personnel Methods and Summary Statistics* (1989).

[13] Sidhant Gupta, Daniel Morris, Shwetak Patel, and Desney Tan. 2012. Soundwave: using the doppler effect to sense gestures. In *Proceedings of ACM CHI*.

[14] Masakazu Higuchi and Takashi Komuro. 2015. Multi-finger AR Typing Interface for Mobile Devices Using High-Speed Hand Motion Recognition. In *Proceedings of ACM CHI*.

[15] Intel. 2017. Intel Realsense. http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html. (2017).

[16] Bryce Kellogg, Vamsi Talla, and Shyamnath Gollakota. 2014. Bringing gesture recognition to all devices. In *Proceedings of Usenix NSDI*.

[17] Kourosh Khoshelham. 2011. Accuracy analysis of kinect depth data. In *ISPRS workshop Laser Scanning*.

[18] Jian Liu, Yan Wang, Gorkem Kar, Yingying Chen, Jie Yang, and Marco Gruteser. 2015. Snooping Keystrokes with mm-level Audio Ranging on a Single Phone. In *Proceedings of ACM MobiCom*.

[19] I. Scott Mackenzie. 1992. Fitts' Law as a Research and Design Tool in Human-Computer Interaction. *Human-Computer Interaction* (1992).

[20] Pedro Melgarejo, Xinyu Zhang, Parameswaran Ramanathan, and David Chu. 2014. Leveraging directional antenna capabilities for fine-grained gesture recognition. In *Proceedings of ACM UbiComp*.

[21] Microsoft. 2014. Microsoft Kinect. http://www.microsoft.com/en-us/kinectforwindows/. (2014).

[22] Microsoft. 2016. Microsoft Hololens. http://www.microsoft.com/microsoft-hololens/en-us. (2016).

[23] Robert B Miller. 1968. Response time in man-computer conversational transactions. In *Proceedings of ACM AFIPS*.

[24] Mark R Mine, Frederick P Brooks Jr, and Carlo H Sequin. 1997. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *Proceedings of ACM SIGGRAPH*.

[25] Leap Motion. 2015. Leap Motion. https://www.leapmotion.com/. (2015).

[26] Taichi Murase, Atsunori Moteki, Noriaki Ozawa, Nobuyuki Hara, Takehiro Nakai, and Katsuhito Fujimoto. 2011. Gesture keyboard requiring only one camera. In *Proceedings of ACM UIST*.

[27] Rajalakshmi Nandakumar, Vikram Iyer, Desney Tan, and Shyamnath Gollakota. 2016. FingerIO: Using Active Sonar for Fine-Grained Finger Tracking. In *Proceedings of ACM CHI*.

[28] OptiTrack. 2018. OptiTrack. http://www.optitrack.com. (2018).

[29] Nobuyuki Otsu. 1975. A threshold selection method from gray-level histograms. *Automatica* (1975).

[30] S. L. Phung, A. Bouzerdoum, and D. Chai. 2005. Skin segmentation using color pixel classification: analysis and comparison. *IEEE Transactions on Pattern Analysis & Machine Intelligence* (2005).

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

[32] Rajesh PN Rao. 2013. *Brain-computer interfacing: an introduction*. Cambridge University Press.

[33] Anthony Santella, Maneesh Agrawala, Doug DeCarlo, David Salesin, and Michael Cohen. 2006. Gaze-based interaction for semi-automatic photo cropping. In *Proceedings of ACM CHI*.

[34] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. 2015. Kinect range sensing: Structured-light versus Time-of-Flight Kinect âŸ. *Computer Vision and Image Understanding* (2015).

[35] Jie Song, Gábor Sörös, Fabrizio Pece, Sean Ryan Fanello, Shahram Izadi, Cem Keskin, and Otmar Hilliges. 2014. In-air gestures around unmodified mobile devices. In *Proceedings of ACM UIST*.

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

[37] Junjue Wang, Kaichen Zhao, Xinyu Zhang, and Chunyi Peng. 2014. Ubiquitous keyboard for small mobile devices: harnessing multipath fading for fine-grained keystroke localization. In *Proceedings of ACM MobiSys*.

[38] Wei Wang, Alex X. Liu, and Ke Sun. 2016. Device-free gesture tracking using acoustic signals. In *Proceedings of ACM MobiCom*.

[39] Yan Wang, Jian Liu, Yingying Chen, Marco Gruteser, Jie Yang, and Hongbo Liu. 2014. E-eyes: In-home Device-free Activity Identification Using Fine-grained WiFi Signatures. In *Proceedings of ACM MobiCom*.

[40] Teng Wei and Xinyu Zhang. 2015. mTrack: High-Precision Passive Tracking Using Millimeter Wave Radios. In *Proceedings of ACM MobiCom*.

[41] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. 2013. Analysis of the accuracy and robustness of the leap motion controller. *Sensors* (2013).

[42] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. 2015. ATK: Enabling Ten-Finger Freehand Typing in Air Based on 3D Hand Tracking Data. In *Proceedings of ACM UIST*.

[43] Yafeng Yin, Qun Li, Lei Xie, Shanhe Yi, Ed Novak, and Sanglu Lu. 2016. CamK: a Camera-based Keyboard for Small Mobile Devices. In *Proceedings of IEEE INFOCOM*.

[44] Sangki Yun, Yi-Chao Chen, Huihuang Zheng, Lili Qiu, and Wenguang Mao. 2017. Strata: Fine-Grained Acoustic-based Device-Free Tracking. In *Proceedings of ACM Mobisys*.

[45] Chi Zhang, Josh Tabor, Jialiang Zhang, and Xinyu Zhang. 2015. Extending Mobile Interaction Through Near-Field Visible Light Sensing. In *Proceedings of ACM MobiCom*.

[46] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. 2017. DolphinAttack: Inaudible voice commands. In *Proceedings of ACM CCS*.

[47] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. 2010. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of IEEE CODES+ISSS*.

[48] Chenyin Zhao and Songqing Fan. 2002. Measurement of Fingers. *Progress of Anatomical Sciences* (2002).