Turning a Mobile Device into a Mouse in the Air

Sangki Yun, Yi-Chao Chen, Lili Qiu
The University of Texas at Austin
{sangki,yichao,lili}@cs.utexas.edu

ABSTRACT

A mouse has been one of the most successful user interfaces due to its intuitive use. As more devices are equipped with displays and offer rich options for users to choose from, a traditional mouse that requires a surface to operate is no longer sufficient. While different types of air mice are available in the market, they rely on accelerometers and gyroscopes, which significantly limit the accuracy and ease of use.

In this paper, we develop a system that can accurately track hand movement to realize a mouse. A unique advantage of our scheme is that it achieves high tracking accuracy using the existing hardware already available in the mobile devices (*e.g.*, smart phones and smart watches) and equipment to be controlled (*e.g.*, smart TVs). More specifically, our approach 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 device's initial position using its movement trajectory. Based on the information, we continuously track the device's new position in real time. This is feasible because many devices, such as smart TVs, PCs, and laptops, already have multiple speakers. When only one speaker is available, we can leverage the frequency shift of sound along with the phase of received WiFi signal to enable tracking. 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.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous

General Terms

Algorithms, Measurement, Performance

Keywords

Doppler Effect; Tracking; Accelerometer; Gyroscope

1. INTRODUCTION

Motivation: 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. More and more devices in the future, such as Google Glasses, baby monitors, and new generation of home appliances, will all desire mouse functionalities, which allow users to choose from a wide variety of options and easily click on different parts of the view.

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?

Challenges: In order to enable the air mouse capability, 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 [37]. Some of them even provides motion control and gesture recognition using inertial sensors, such as accelerometers and gyroscopes [34, 35]. Existing accelerometers are 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.

A few recent works track RFID tags in centimeter-level [40, 41, 46], but they require multiple special RFID readers, each with 4 antennas. Using external devices, such as depth sensor (*e.g.*, Kinect [1]) and IR sensor (*e.g.*, Wii [2]), incurs significant additional cost and limits the types of devices that can be controlled.

Our approach: In this paper, 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 1 shows an example of our system, where a mobile device with a microphone serves as a mouse for a smart TV with two

MobiSys'15, May 18–22, 2015, Florence, Italy.

http://dx.doi.org/10.1145/2742647.2742662.

Figure 1: Enabling accurate device tracking using the Doppler effect of the acoustic signal.

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 address these challenging issues in the following way. 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. Finally, we generalize our approach to handle the equipment that has only one speaker along with another wireless device (*e.g.*, WiFi). In this case, we use the frequency shift from the inaudible acoustic signal and the phase of received WiFi signal to derive the distance of the mobile device from the speaker and WiFi transmitter. We apply the same framework to continuously track the device in real time as before.

We implement a prototype on a smart phone and a desktop PC. We perform detailed experiments and user studies to demonstrate the feasibility and effectiveness of real-time tracking. Our evaluation reveals the limitations of accelerometer based tracking. Meanwhile, 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 [41] that requires an array of 4 antennas. Compared to Tagoram [46] with unknown track, AAMouse is 7 times more accurate. In addition, we show that AAMouse also provides better user experience than Gyroscope based approach, which is widely used in commercial air mouse and smart TV remote controllers, due to its more intuitive use.

Paper outline: The rest of this paper is organized as follows. Section 2 uses measurement to show accelerometers are not sufficient to provide accurate position information. We describe our approach in Section 3, and present implementation details in Section 4. We evaluate its performance in Section 5. We review related work in Section 6, and conclude in Section 7.

2. 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.

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 [18, 26]. 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.

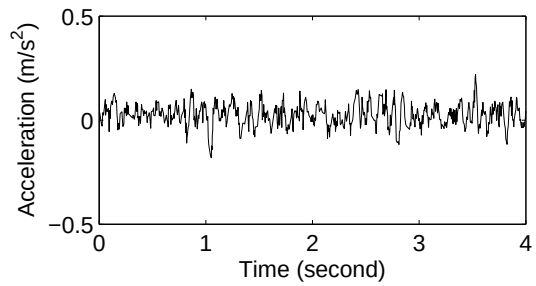
To demonstrate these issues, we observe that the acceleration measured by the accelerometer inherently includes gravity, which is approximately 9.8 m/s^2 on earth. When the device lies perfectly flat, gravity only affects the acceleration of Z-axis, so X and Y-axis have zero accelerations. However, while 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.

One way to remove gravity from the measured acceleration is to apply a high-pass filter [11]. 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.

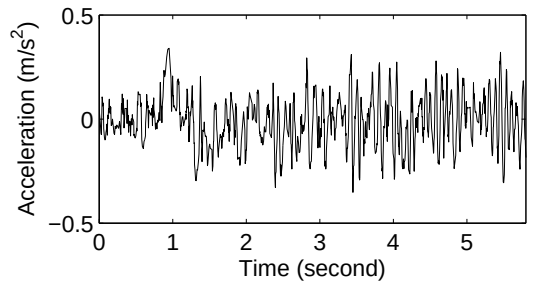
Next we perform the measurement of linear acceleration in Android devices. We use Google NEXUS 4 and Samsung Galaxy 3. Their accelerometers come from InvenSense [15], which is one of the few accelerometer manufacturers that provides accelerometers for latest iPhones and many android phones, since InvenSense MEMS sensors are considered the most accurate.

Our experiments have 3 users. 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.

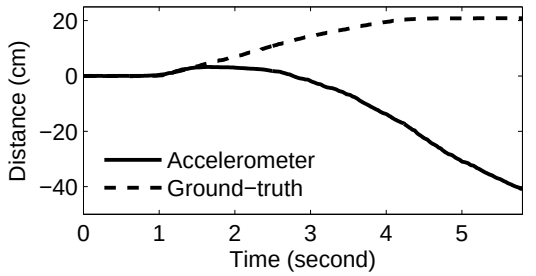
Figure 2(a) shows a snapshot of the acceleration while a user is holding the device. We only plot Y-axis acceleration for ease of viewing. While a user is holding the device, the average magnitude



(a) Acceleration while the device is not moving



(b) Acceleration while the device is moving



(c) Moving distance tracked by accelerometer

Figure 2: Comparison of acceleration while device is moving and not moving. The device's position tracked by an accelerometer significantly deviates from the ground truth.

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 2(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 2(a) and (b), it is hard to tell exactly when the device starts moving and when it stops. 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 2(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 2(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.

3. OUR APPROACH

This section presents our approach to device tracking.

3.1 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 [24]. 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:

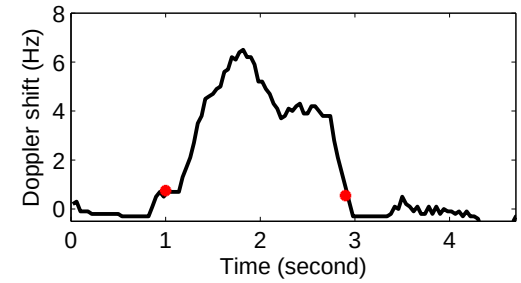
$$v = \frac{F^s}{F} c. \tag{1}$$

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.

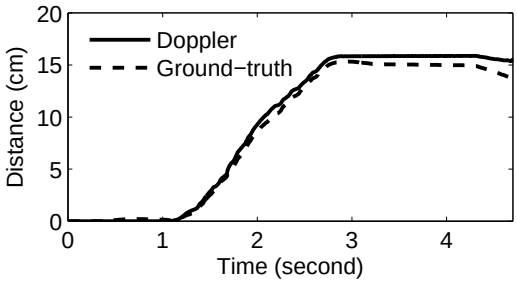
The Doppler effect is observed in any wave, including RF and acoustic signal. We use acoustic signal to achieve high accuracy due to its (i) narrower bandwidth and (ii) slower propagation speed. Its narrower bandwidth makes it easy to detect 1 Hz frequency shift than that in RF signals (*i.e.*, 44.1 KHz in acoustic signals versus 20 MHz in WiFi). In order to improve the accuracy of detecting frequency shift in WiFi signals, WiSee [30] 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 $\frac{1\times 346.6}{17000} \approx 0.02m/s = 2cm/s$. In comparison, when the RF center frequency is 2.4 GHz, the resolution is $\frac{1\times 3\times 10^8}{2.4\times 10^9} = 0.125m/s = 12.5cm/s$, which is around 6 times as large. This implies that for the same movement, the Doppler shift of the acoustic signal is 6× that of RF signal, which allows us to more accurately measure the movement speed.

Moreover, 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 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. Figure 3 shows the Doppler shift and the moving distance over time estimated by Equation 1. The device starts moving at 1 second and stops at 2.8 second. Unlike acceler-



(a) Doppler shift



(b) Moving distance

Figure 3: The Doppler shift and the moving distance estimated based on it. The tracking error is less than 1 cm. The red dots in Figure 3(a) represent the start and end of the movement.

ation measurement in Figure 2, 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(b), the maximum tracking error is only 0.7 cm.

Based on the above concept, we develop the following system, as illustrated in Figure 1, 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. 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) [13]. 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 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 [27]. 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.

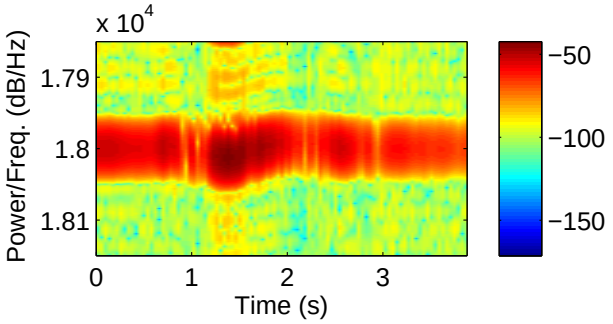
To address this issue, we use Short-term Fourier Transform (STFT), which is used to analyze the change of spectrum over time [42]. It uses fewer data samples than that required by FFT. The missing values of the input is filled with zeros. Then the FFT output has desired resolution. However, this alone 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 [16].

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 4(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.

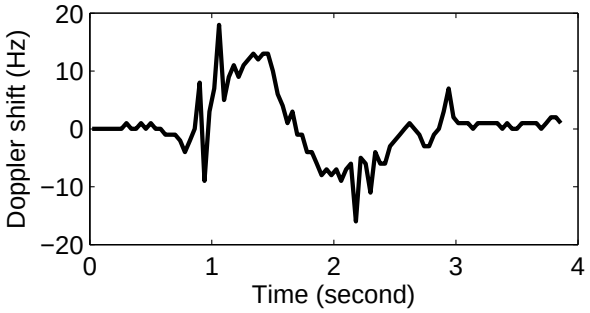
3.3 Tracking the Position

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 3.5 and 3.6.

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. Let t_s be the sampling interval in which we estimate the frequency shift. Our evaluation



(a) Spectrogram



(b) Doppler shift

Figure 4: Frequency domain analysis of the recorded audio and the Doppler shift.

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 [10]. 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 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 5, 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]:

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

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.

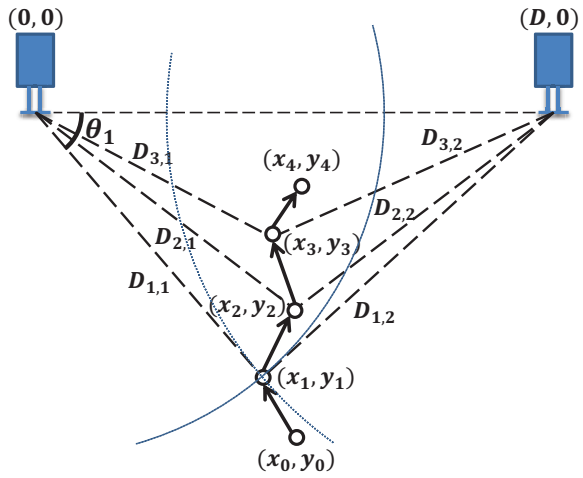


Figure 5: Tracking the position based on the Doppler shift. Whenever a new Doppler sample arrives, we calculate the distance from the speakers, and estimate the new position by finding the intersection of the two circles.

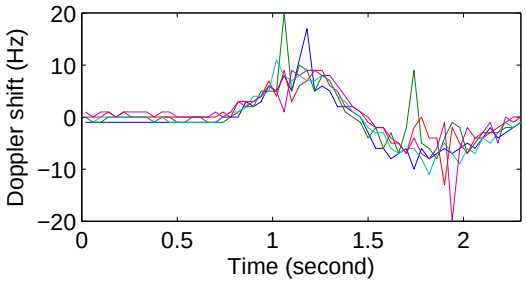
In the next Doppler sampling interval, we measure $F^s_{2,1}$ and $F^s_{2,2}$, calculate $D_{2,1}$ and $D_{2,2}$, and derive (x_2, y_2) from it. This process is repeated until the device stops moving. To minimize the impact of errors in the frequency shift estimation, we filter out the frequency shift below 1 Hz and use the remaining frequency shifts to estimate the speeds and distance.

3.4 Improving the Accuracy

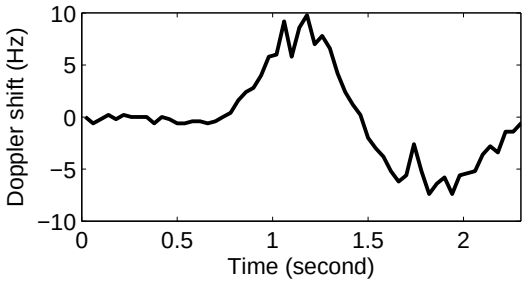
To achieve high accuracy in device tracking, it is crucial to accurately estimate the Doppler shift. However, measuring it from a single sound wave may not be reliable. The accuracy of estimating the Doppler shift in part depends on SNR of the received signal. Due to frequency selective fading, SNR varies across frequencies. To enhance robustness, we send 1-Hz sound tones at different center frequencies, and 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 5.1, we allocate 10 sound tones for each speaker.

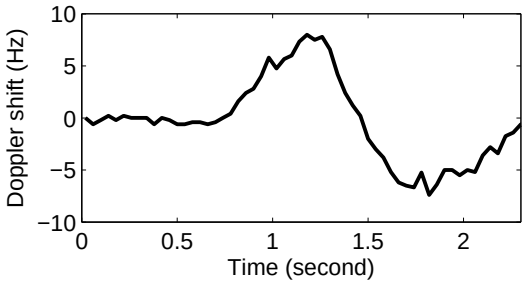
The next question is how to take advantage of the measurements at multiple frequencies to improve the accuracy. One approach is to apply Maximal Ratio Combining (MRC) technique used in the receiver antenna diversity, which averages the received signal weighted by the inverse of the noise variance. It is known to be optimal when the noise follows a Gaussian distribution [44]. 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



(a) Doppler shift measured from 5 tones



(b) Doppler shift after MRC without outlier removal



(c) Doppler shift after MRC with outlier removal

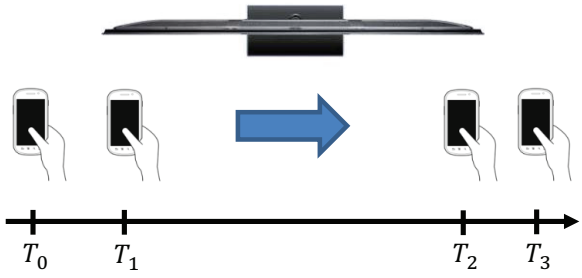
Figure 6: Improving the accuracy of the Doppler shift estimation using MRC and outlier removal.

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.

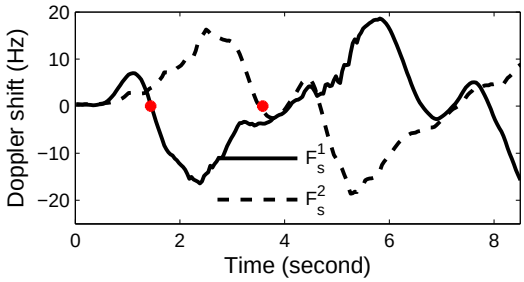
Figure 6 shows the raw Doppler shift measurements and the result after MRC with and without outlier removal. It shows that the Doppler estimation after outlier removal yields more smooth output and likely contains smaller errors. In Section 5.1, we show that the outlier removal improves the accuracy by 26% when 5 sound tones are used.

3.5 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.



(a) Calibration process.



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

Figure 7: Illustration of the calibration process. We measure the distance between two speakers by estimating T_1 and T_2 (*i.e.*, the time it gets closest to the left and right speakers, respectively) and the speed during T_1 and T_2 using the Doppler shift. The red dots in Figure 7(b) represent T_1 and T_2, where $T_1 = 1.48$ and $T_2 = 3.58$.

We propose a Doppler based calibration method. It only takes a few seconds for a user to conduct calibration. As shown in Figure 7(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 7(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 3.1, 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 7), 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, 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.

3.6 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 [33, 9]. 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 3.3, 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 4 for further details of this choice.

In Section 5.1, 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 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.

3.7 Controlling a Device with One Speaker

So far we have assumed the equipment to be controlled has two speakers so that we can estimate the distance from these speakers to track the mobile device. All Smart TVs have two speakers. Most laptops (*e.g.*, Lenovo X series, Acer Aspire S3, Dell XPS 13) have two speakers. Some recent laptops (*e.g.*, Dell Studio 17) have 3 speakers to offer better multimedia experience. Three speakers provide more anchor points and allow us to track in a 3-D space or further improve tracking accuracy in a 2-D space.

In this section, we further extend our approach to handle the devices that have only one speaker. We assume it has another wireless device (*e.g.*, WiFi or bluetooth). Then we use the Doppler effect of acoustic signal from the speaker along with the RF signal from the wireless device to enable tracking.

We use the same framework as described above for tracking. The new issue is how to estimate the distance between the mobile device and the wireless device on the equipment. We use the following known relationship from the phase rotation of the received RF signal to compute the distance:

$$\phi_{t1} = -mod(\frac{2\pi}{\lambda} d_{t1}, 2\pi)$$
$$\phi_{t2} = -mod(\frac{2\pi}{\lambda} d_{t2}, 2\pi)$$

where ϕ_{t1} and ϕ_{t2} denote the phase of the received signal at the mobile device at time $t1$ and $t2$, respectively, and d_{t1} and d_{t2} are their respective distances. This enables us to track the new distance from the RF source by

$$d_{t2} = (\frac{\phi_{t2} - \phi_{t1}}{2\pi} + k)\lambda + d_{t1}. \quad (2)$$

k is an integer and set to 0 here, since our sampling interval of RF phase is 10 ms and it is safe to assume we move less than one wavelength during a sampling interval.

One of the challenges in RF phase based tracking is accurate measurement of the received signal phase. In particular, the carrier frequency offset (CFO) between the sender and the receiver causes the phase to change over time even if the receiver is not moving. A few recent works, such as MegaMIMO [32, 47], show that it is feasible to track the phase change over time with precise CFO estimation. To simplify our implementation, we connect a sender and a receiver with the same external clock in our experiment to guarantee that they have no frequency offset. We estimate the phase of the receiver while the sender is continuously sending 1 MHz wide OFDM symbols.

As in Section 3.3, if we know the device's initial location and the distance between the speaker and RF source, we can track the position by finding the intersections of the two circles whose radii are the distance measured by the Doppler shift and RF phase tracking, respectively. We can again apply particle filter to address the issue that the device's initial position is unknown. We can adopt similar calibration to measure the distance between the speaker and RF source by detecting the speaker's position based on the change in the sign of the Doppler shift (*i.e.*, going from positive to negative) and detecting the RF source's position based on the change in the phase of the received RF signal (*i.e.*, going from decreasing phase to increasing phase).

4. 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 3. 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

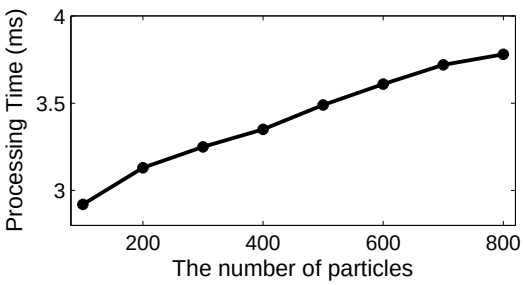


Figure 8: Processing time with a varying number of particles.

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 8, 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 [17, 18].

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 [36]. 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 [5] and smart TV motion controllers [34, 35]. 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 [7]. 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).

5. 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.

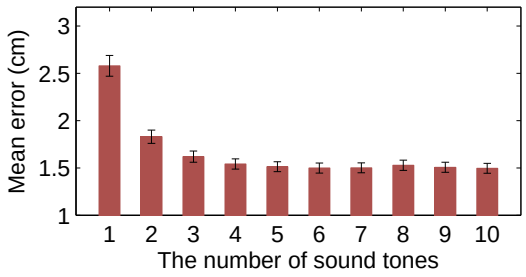
For our experiment, we implement a program that displays a pointer and the trajectory of the device movement in real-time so that users can see how well the pointer moves as they intend. To emit sound tones in inaudible frequency range, we play a wave format audio file with specific frequency tones that 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 5.1.

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.

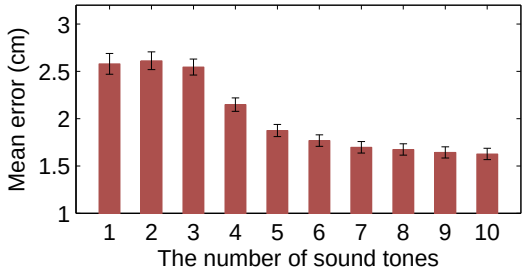
5.1 Micro benchmark

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

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.



(a) With outlier removal.



(b) Without outlier removal.

Figure 9: AAMouse trajectory error with different numbers of sound tones. The numbers of sound tones in both figures are the numbers before outlier removal.

Figure 9(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 9(b) is the mean trajectory error without the outlier removal introduced in Section 3.4. 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 3.5. The actual distance between the speakers is 0.9 m.

Figure 10 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 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 11, the tracking error increases slightly with the increasing error in the

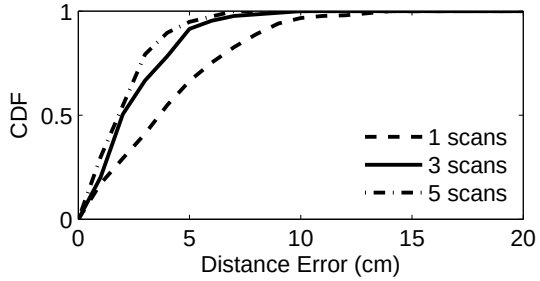


Figure 10: CDF of the speaker distance measurement result with different numbers of scannings.

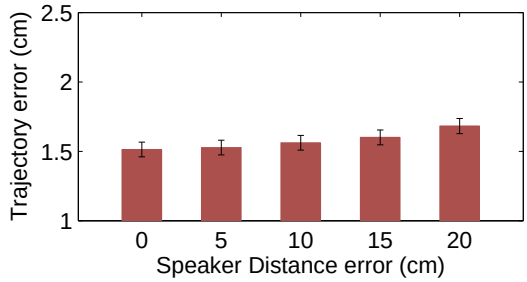


Figure 11: Trajectory error as the error of the speaker distance varies.

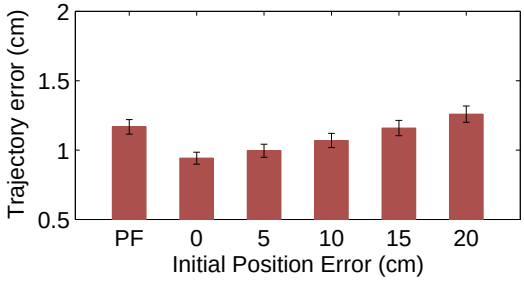


Figure 12: Trajectory error as the error of the device initial position varies.

speaker distance estimation. The speaker distance estimation error of 10 cm increases the mean error by 0.05 cm (*i.e.*, from 1.51 to 1.56 cm). In Section 3.3, 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 11.

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 3.6. 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 12 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.

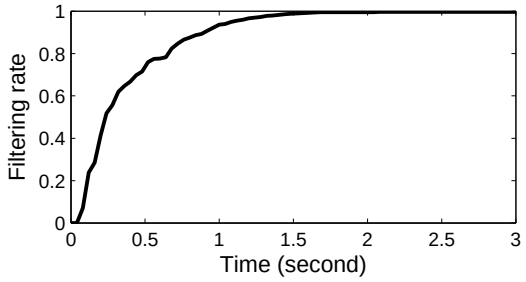


Figure 13: Particle filter convergence time.

Particle filter convergence: Figure 13 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.

5.2 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 5.1. Figure 14 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.

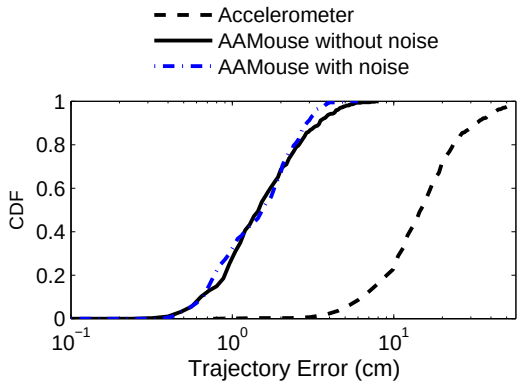


Figure 14: CDF of trajectory error.

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 [20], 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" [25]

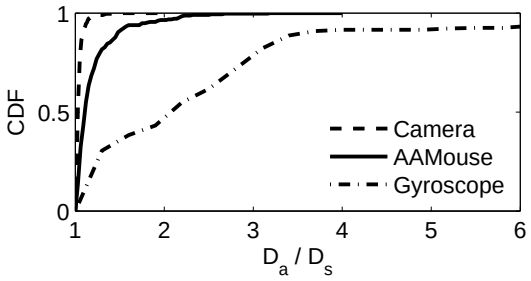


Figure 15: CDF of unnecessary movement to reach the target.

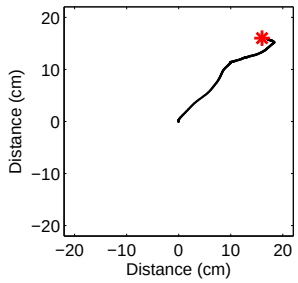
as the background noise. As shown in Figure 14, the accuracy remains the same. This is because the frequency of human voice and many music genres, such as Jazz, Pop and Classic, do not exceed 10 KHz. Even for some music with clangs and artificial sounds, such as rock and electronic music, their frequency 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 [39], 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.

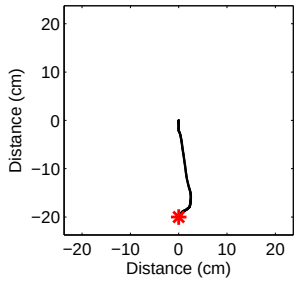
We compare AAMouse, gyroscope, and camera-based tracking, which all succeed in touching the target every time since the users are required to change the pointer position until the target is reached. We do not compare with the accelerometer based scheme since its error is too large: despite significant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

Figure 15 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 14 and 15, 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, 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

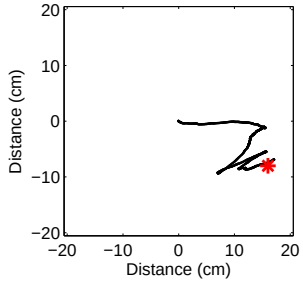
make several attempts before finally touching the pointer, which increases R.



(a) AAMouse.



(b) Camera based tracking.



(c) Gyroscope based tracking.

Figure 16: Trajectory of AAMouse, camera and gyroscope based tracking while the users are trying to reach the target point. R of each scheme corresponds to the 80th percentile.

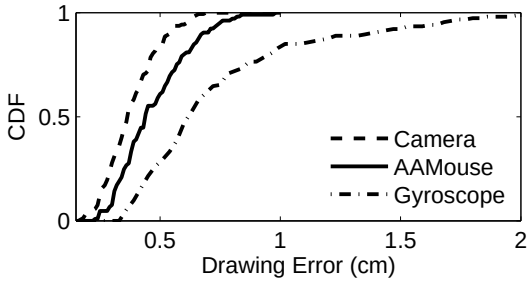
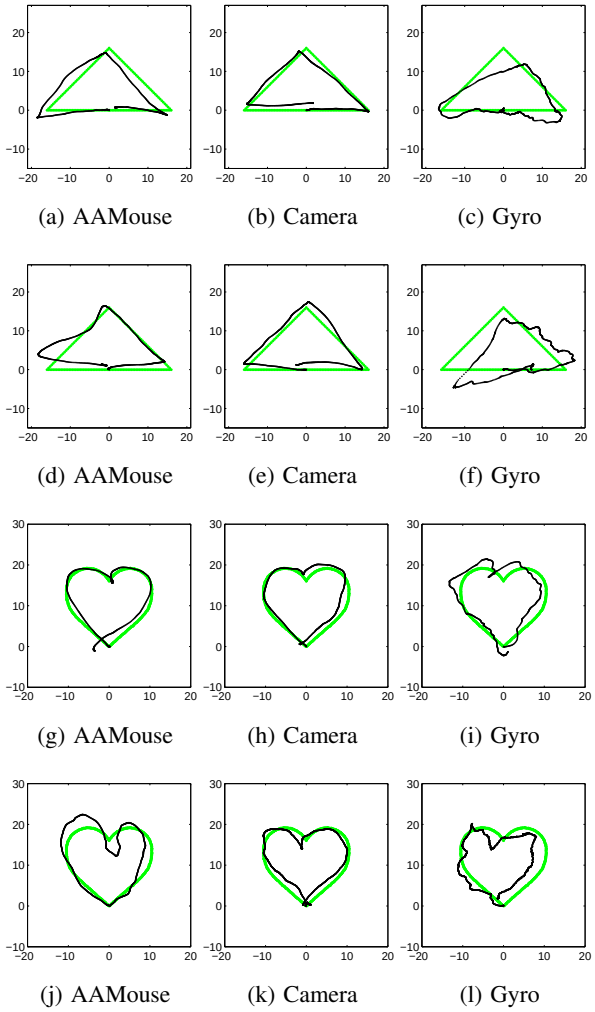


Figure 17: CDF of drawing error.

Figure 16 shows example trajectories when the user is trying to reach the target under the 80th percentile R. As we can see,



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

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

(g) AAMouse (h) Camera (i) Gyro

(j) AAMouse (k) Camera (l) Gyro

Figure 18: Figures drawn by AAMouse, camera and gyroscope based tracking. (a),(b),(c), (g), (h), (i) and (d),(e),(f), (j), (k), (l) correspond to the median and 80th percentile errors of each scheme, respectively. The units of both the x-axis and y-axis are centimeters.

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 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 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. Figure 18 shows

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 Figure 18 (d),(e),(f), (j), (k) and (l), 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.

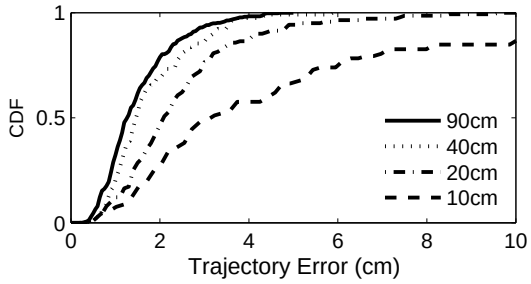


Figure 19: CDF of trajectory error as the speaker distance varies.

Impact of the speaker distance: Next we evaluate the impact of the speaker distance on the accuracy of AAMouse. In Section 3.3, 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 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 [23].

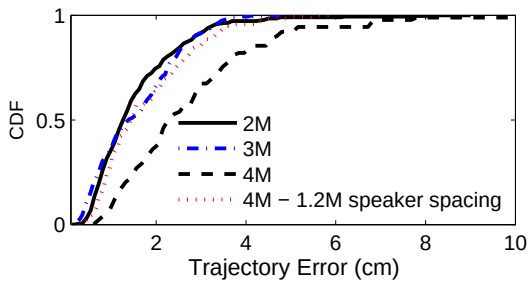


Figure 20: CDF of trajectory error as the distances between the speakers and the microphone varies.

Range experiment: Figure 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 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.

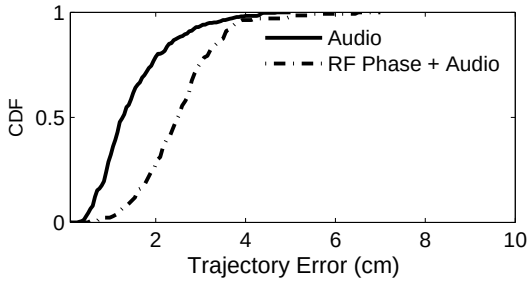


Figure 21: CDF of trajectory error while RF phase is used for tracking as well as Doppler.

Using RF signal phase: We evaluate the tracking accuracy when RF signal phase is used for tracking along with the Doppler shift estimation from one speaker. We use a USRP node with an antenna to replace a right speaker, and keep the distance between the left speaker and the USRP transmitter at 90 cm. While our implementation uses USRP to derive distance estimation, it is possible to apply the approach on the commodity WiFi card [17]. Figure 21 shows CDF of the trajectory error, and confirms the feasibility of tracking using the Doppler shift and the phase of RF signal. The median error increases from 1.4 cm to 2.5 cm because RF phase is more easily affected by multi-path fading and environmental change such as human movement. Even while the device is not moving, the phase may change due to movement of other human bodies (besides hand movement). In comparison, while the Doppler shift is also affected by other human body movement, we use the peak frequency and ignore the other frequency shifts, which enhances its robustness.

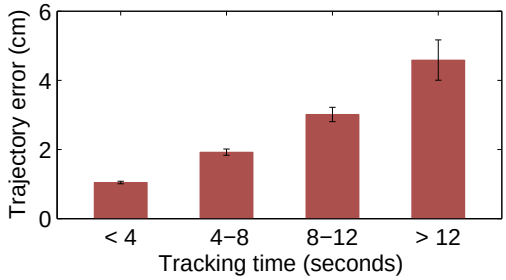
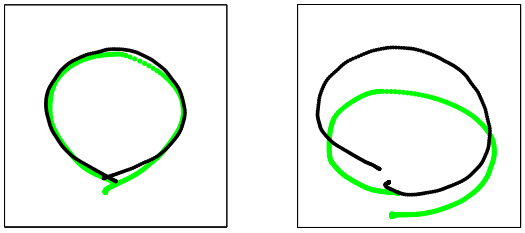


Figure 22: Trajectory error with different tracking time.

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 22 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 23 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 23 (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 23: Compare the circles drawn by AAMouse (black) and the ground-truth (green).

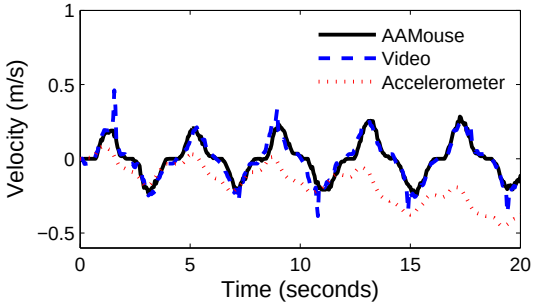


Figure 24: Velocity while moving the device backward and forward.

From our experience, we find the user is mainly concerned about instantaneous 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 24, 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.

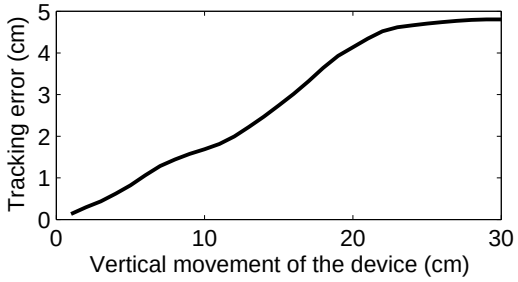


Figure 25: Tracking error as the device is moving vertically.

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 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 25 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.

6. RELATED WORK

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.*, [29, 48,

31]). Different from these ranging based works, we focus on continuous trajectory tracking to enable mouse functionalities. [38] 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 [45], 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 [40, 41, 46]. They require special RFID readers with many antennas. RF-IDraw [41] uses two RFID readers, each with 4 antennas, and Tagoram [46] 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: [22] 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 [33] 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 [28, 4]. E-gesture [28] uses gyroscope and accelerometer for gesture recognition, but it is not applicable to device tracking. [4] 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. Our evaluation result shows that AAMouse achieves much higher accuracy than the accelerometer based tracking.

Using Doppler for motion and device tracking: WiSee [30] 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 [12, 8, 6, 39] use the Doppler shift of the audio signal for gesture recognition rather than device tracking. DopLink [6] enables the interaction among devices using the Doppler shift, and Spartacus [39] finds the direction of the device movement using the Doppler shift and pairs it with another device that moves in the same direction. Swadloon [14] 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. [19] 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 [43]. 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* [5] in the market. These are usually used as remote controllers for laptops and PCs. Advanced remote controllers for smart TVs, such as Roku 3 [34], Samsung Smart Remote [35], and LG magic motion controller [21], 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, 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.

7. CONCLUSION

In this paper, 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.

Acknowledgements

We are grateful to Shyam Gollakota and anonymous reviewers' insightful comments.

8. REFERENCES

[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] S. Agrawal, I. Constandache, S. Gaonkar, R. Roy Choudhury, K. Caves, and F. DeRuyter. Using mobile phones to write in air. In *Proc. of ACM MobiSys*, pages 15–28, 2011.

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

[6] M. T. I. Aumi, S. Gupta, M. Goel, E. Larson, and S. Patel. Doplink: Using the doppler effect for multi-device interaction. In *Proc. of ACM UbiComp*, pages 583–586, 2013.

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

[8] K.-Y. Chen, D. Ashbrook, M. Goel, S.-H. Lee, and S. Patel. Airlink: Sharing files between multiple devices using in-air gestures. In *Proc. of ACM Ubicomp*, 2014.

[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of Robotics and Automation*, volume 2, pages 1322–1328, 1999.

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

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

[12] S. Gupta, D. Morris, S. Patel, and D. Tan. Soundwave: using the doppler effect to sense gestures. In *Proc. of the SIGCHI*, pages 1911–1914, 2012.

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

[14] W. Huang, Y. Xiong, X.-Y. Li, H. Lin, X. Mao, P. Yang, and Y. Liu. Shake and walk: Acoustic direction finding and fine-grained indoor localization using smartphones. In *Proc. of IEEE INFOCOM*, 2014.

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

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

[17] S. Kumar, D. Cifuentes, S. Gollakota, and D. Katabi. Bringing cross-layer MIMO to today's wireless lans. In *In Proc. of ACM SIGCOMM*, August 2013.

[18] S. Kumar, S. Gil, D. Katabi, and D. Rus. Accurate indoor localization with zero start-up cost. In *Proc. of ACM MobiCom*, pages 483–494, 2014.

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

[20] C.-H. Lee, J.-L. Shih, K.-M. Yu, and H.-S. Lin. Automatic music genre classification based on modulation spectral analysis of spectral and cepstral features. *IEEE Transactions on Multimedia*, 11(4):670–682, 2009.

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

[22] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proc. of UbiComp*, pages 421–430, 2012.

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

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

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

[26] H. Nyqvist and F. 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.

[27] A. V. Oppenheim, R. W. Schafer, J. R. Buck, et al. *Discrete-time signal processing*, volume 2. Prentice-hall Englewood Cliffs, 1989.

[28] T. Park, J. Lee, I. Hwang, C. Yoo, L. Nachman, and J. 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.

[29] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan. BeepBeep: a high accuracy acoustic ranging system using COTS mobile devices. In *Proc. of ACM SenSys*, 2007.

[30] Q. Pu, S. Gupta, S. Gollakota, and S. Patel. Whole-home gesture recognition using wireless signals. In *Proc. of ACM MobiCom*, 2013.

[31] J. Qiu, D. Chu, X. Meng, and T. Moscibroda. On the feasibility of real-time phone-to-phone 3D localization. In *Proc. of ACM MobiSys*, 2011.

[32] H. S. Rahul, S. Kumar, and D. Katabi. MegaMIMO: scaling wireless capacity with user demands. In *Proc. of ACM SIGCOMM*, pages 235–246, 2012.

[33] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen. Zee: zero-effort crowdsourcing for indoor localization. In *Proc. of ACM MobiCom*, 2012.

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

[35] CES 2014: Samsung shows off new gyroscopic remote control. `http://www.digitalversus.com/tv-television/ces-2014-samsung-shows-off-new-gyroscopic-remote-control-n32491.html`.

[36] K. Seifert and O. Camacho. Implementing positioning algorithms using accelerometers. *Freescale Semiconductor*, 2007.

[37] CES 2014 trends: New remotes and interfaces to make smart TVs actually usable. `http://spectrum.ieee.org/tech-talk/consumer-electronics/audiovideo/ces-2014-trends-getting-smart-tv-under-control`.

[38] A. Smith, H. Balakrishnan, M. Goraczko, and N. Priyantha. Tracking moving devices with the cricket location system. In *Proc. of ACM MobiSys*, 2005.

[39] Z. Sun, A. Purohit, R. Bose, and P. Zhang. Spartacus: spatially-aware interaction for mobile devices through energy-efficient audio sensing. In *Proc. of ACM Mobisys*, pages 263–276, 2013.

[40] J. Wang and D. 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.

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

[42] P. 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.

[43] C. A. Wingrave, B. Williamson, P. D. Varcholik, J. Rose, A. Miller, E. Charbonneau, J. Bott, and J. LaViola. The wiimote and beyond: Spatially convenient devices for 3d user interfaces. *IEEE Computer Graphics and Applications*, 30(2):71–85, 2010.

[44] G. Woo, P. Kheradpour, D. Shen, and D. Katabi. Beyond the bits: Cooperative packet recovery using physical layer information. In *Proc. of ACM MobiCom*, 2007.

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

[46] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu. Tagoram: Real-time tracking of mobile RFID tags to high precision using cots devices. In *Proc. of ACM MobiCom*, 2014.

[47] S. Yun, L. Qiu, and A. Bhartia. Multi-point to multi-point MIMO in wireless LANs. In *Proc. of INFOCOM Mini-Conference*, April 2013.

[48] Z. Zhang, D. Chu, X. Chen, and T. Moscibroda. Swordfight: Enabling a new class of phone-to-phone action games on commodity phones. In *Proc. of ACM MobiSys*, 2012.