Mobile Motion Gaming: Enabling a New Class of Phone-to-Phone Action Games on Commodity Phones

Zengbin Zhang, David Chu, Xiaomeng Chen, and Thomas Moscibroda

Abstract—Mobile gaming is a big driver of app marketplaces. However, few mobile games deliver truly distinctive gameplay experiences for ad hoc collocated users. As an example of such an experience, consider a sword fight dual between two users facing each other where each user's phone simulates a sword. With phone in hand, the users' thrusts and blocks translate to attacks and counterattacks in the game. Such Phone-to-Phone Mobile Motion Games (MMG) represent interesting and novel gameplay for ad hoc users in the same location. One enabler for an MMG game like sword fight is continuous, accurate distance ranging. Existing ranging schemes cannot meet the stringent requirements of MMG games: speed, accuracy, and noise robustness. In this work, we design `FAR`, a new ranging scheme that can localize at 12 Hz with 2-cm median error while withstanding up to 0-dB noise, multipath, and Doppler effect issues. Our implementation runs on commodity smartphones and does not require any external infrastructure. Moreover, distance measurement accuracy is comparable to that of Kinect, a fixed-infrastructure motion capture system. Evaluation on users playing two prototype games indicate that `FAR` can fully support dynamic game motion in real time.

Index Terms—Acoustic localization, mobile gaming, mobile motion games, smartphones

1 Introduction

Mobile gaming constitutes a large and fast growing industry. Estimates on the total worldwide market size differ, but typical numbers are in the order of $10 billion for 2011 [1]. Furthermore, continued growth in the underlying forces—smartphone and tablet sales, mobile Internet subscribers and app downloads—all point to a bright future for the industry.

The landscape of today's mobile games is rich and varied. However, one commonality among existing multiplayer games is that they invariably require the players to be physically passive, look at the screen, and conduct game action via interaction with the screen. The success of Nintendo Wii or Xbox Kinect in console gaming on the other hand has demonstrated a demand for much more interactive games in which gameplay directly involves the user's physical activity. In this paper, we report on the development of a novel class of *Phone-to-Phone Mobile Motion Games* (MMG) that achieve a similar level of physical interactivity. These games are characterized by the fact that the *position, location, orientation, or movement of the phone is an integral part of game play*. In some cases, the phone may even serve as the equivalent of a Wii-stick, and players may not even need to look at the screen while playing. However, in contrast to Wii or Kinect, we do not rely on any external infrastructure such as a microphone array or cameras; MMG games are played purely phone-to-phone.

Consider, for example, the following *SwordFight* game. The rules are simple: Two players wield their phones, and try to attack each other. A player can attack the opponent by tapping the screen. If player A attacks, and her phone is within 20 cm of the opponent's phone, she wins. However, attacking costs energy, and an attack can only be sustained for 4 seconds before the energy has completely depleted. Energy can be regained over time when the player remains in nonattack mode. Thus, one strategy to win is for a player to survive the opponent's attack by quickly moving his phone in such a way that it maintains sufficient distance from the opponent's phone; and then counterattack while the opponent's energy is depleted.

The key enabling technology for a game like SwordFight is the ability to conduct *very fast*, *accurate*, and *robust distance measurements* between the phones, so that at any moment during play, both phones have precise distance information. Studies have shown that to sustain high-speed action games a lag of more than 100 ms decreases user satisfaction and degrades player performance, and a lag of 200 ms is unacceptable [2]. Therefore, we need to be able to conduct phone-to-phone distance measurements at a rate of at least 10 Hz. The measurements also have to be accurate—with no more than a few centimeters of error—and robust in the face of mobility, noise, and networking issues. In the absence of any external infrastructure, the combination of these three requirements constitutes a significant technical barrier on commodity phones, and no existing solution is able to meet them simultaneously.

- *Z. Zhang is with the University of California, Santa Barbara. E-mail: zengbin@cs.ucsb.edu.*
- *D. Chu is with Microsoft Research, One Microsoft Way, Redmond, WA 98052. E-mail: davidchu@microsoft.com.*
- *X. Chen is with Purdue University. E-mail: chen1058@purdue.edu.*
- *T. Moscibroda is with Microsoft Research, Asia. E-mail: moscitho@microsoft.com.*

Manuscript received 13 Sept. 2012; revised 1 Feb. 2013; accepted 14 Feb. 2013; published online 13 Mar. 2013.
For information on obtaining reprints of this article, please send e-mail to: tmc@computer.org and reference IEEECS Log Number TMCSI-2012-09-0461.
Digital Object Identifier no. 10.1109/TMC.2013.38.

Published by the IEEE CS, CASS, ComSoc, IES, & SPS

It is well known that acoustic sound can be used for distance measurements. Works such as [17], [19] have demonstrated that under ideal circumstances (e.g., no mobility) and with sufficient computation time, accurate ranging can be achieved even on commodity phones. The problem is that in an motion game scenario, the circumstances are far from ideal and the requirements substantially more challenging. First, existing ranging protocols are based on the assumption that phone positions remain static during the process of taking a measurement. For a game like SwordFight, this is not a valid assumption as human hand speed can be as high as 2 m/s. Furthermore, with two phones moving toward or apart from each other at high speed, aspects such as the Doppler effect need to be considered and dealt with. Second, acoustic ranging requires the use of expensive cross-correlation algorithms to determine the precise time-of-arrival of the sound signal. Cross-correlation algorithms are computationally intensive and cannot be run without modification on phones at sufficient speed to enable a SwordFight game. Third, not only *computation*, but also the *communication* (acoustic tone exchanges, protocol handshakes, etc.) incurs a fundamental and significant delay.

In this paper, we address these challenges in a systematic manner. Our first contribution is to enable real-time distance measurement by replacing the standard computationally intensive cross-correlation algorithm for finding sound peaks with a more sophisticated and efficient multistage algorithm. Our algorithm employs autocorrelation to fundamentally reduce computational complexity while preserving accuracy by targeting cross correlation to a very narrow search window. Our second contribution is to employ a new pipelined streaming execution strategy that overlaps protocol communication and algorithm computation. While overlapping communication and computation is a well-known technique, in our case, this overlapping comes with a twist—we overlap the *sound waves* of the ranging protocol in addition to the typical networking data packets. It turns out that both pipelining and streaming are critical to realize real-time measurements. Our third contribution is to understand and tackle the practical sources of measurement error during motion gaming for increased robustness. The first set of robustness optimizations addresses mobility and the effects of Doppler shift on the underlying ranging protocol. The second set of optimizations addresses environmental robustness from effects such as ambient noise, multipath, and acoustic tone loss.

Combining these techniques, we develop FAR—a Fast, Accurate, and Robust localization system that enables two potentially fast moving phones to keep accurate distance estimates to each other. Distance measurements can be taken at a rate of 12 Hz with 2-cm median error while withstanding up to 0-dB noise (e.g., players or spectators talking while playing), multipath (e.g., as encountered in small rooms), and the Doppler effect. To practically demonstrate the system's ability to enable novel gaming concepts, we develop two prototype MMG games: SwordFight and ChaseCat.[1] In both isolated player experiments and in-situ gameplay experiments, we find that FAR's distance measurements are comparable to Kinect, a dedicated fixed-infrastructure motion capture system. We have publicly tested our games at various locations with real players; and our experience shows that the games are fun and intuitive to play. We anticipate that with the high-speed ranging API we provide in this paper, many more MMG games can be developed.

2 Assumptions and Requirements

Enabling a game like SwordFight requires a distance ranging subroutine that allows two phones to keep accurate and up-to-date distance information between each other even in the face of noise and high mobility. This poses a unique set of challenges:

- *Phone-to-Phone*. MMG games should be playable everywhere at any time. We do not rely on any external infrastructure beyond the two phones.
- *Commodity phones*. We want MMG games to run on commodity hardware and OSs. This implies handling issues of on-phone sensor sampling rates, computation capacity, or audio playing and recording capabilities, whether they arise from the hardware, (typically closed source) audio driver, or OS.
- *Measurement Frequency*. To create the sensation of continuous real-time distance information, the ranging infrastructure must sample user movement as frequently as possible, at a rate of 10 Hz or more [2].
- *High Accuracy*. Gameplay relies on the high accuracy of distance measurements even at a high degree of phone mobility. Specifically, we aim for a target accuracy of within 2-3 cm up to a normal range of human social interaction of up to approximately 3 m with line of sight.
- *User Mobility*. In a game like SwordFight, players should be able to play without artificial restrictions on their body or hand movement. The underlying ranging infrastructure must be capable of supporting the speed of natural human hand movement (up to 2 m/s) [8].
- *Practical in Most Environments*. As it is natural for players and spectators to talk during a game, and because a game may be played indoors or close to walls, the underlying distance measurement framework must be robust against the impact of high ambient noise levels, multipath effects, and tone loss.

3 Background

In this section, we review the well-known time-of-flight acoustic ranging principle that we share in common with prior phone-based acoustic localization work [17], [19], and we outline why these existing approaches are unsuited for enabling Mobile Motion Gaming.

3.1 Acoustic Distance Measurement

The essential idea is to have two phones A and B play and record known audio tones one after another. These tones are often based on some form of pseudorandom sequences.

[1] See http://research.microsoft.com/mobile-motion-gaming for a video illustrating the gameplay of SwordFight.

Each phone records its own emitted tone as well as the tone originating from the remote phone. Suppose A emits a tone first and records this tone's arrival at its microphone at time t_{A1}. B records the arrival of this tone at time t_{B1}. Next, B emits a tone, which is recorded at the microphones of B and A at times t_{B2} and t_{A2}, respectively. The distance d between the phones can now be calculated as

$$d = \frac{1}{2} \cdot c \cdot [(t_{A2} - t_{A1}) - (t_{B2} - t_{B1})], \tag{1}$$

where c is the speed of sound.

Existing schemes [17], [19] follow a traditional execution strategy of *Play/Record then Compute and Exchange* (see Fig. 2a). The figure shows that the two phones start the *Recording* step at the same time, and then send out the tones one after another (*Tone Playing* step). Guard periods are inserted prior to and following each tone to guarantee that the tones do not overlap, and that the *Recording* is complete.

Upon completion of the recording, the next step is for each phone to determine the exact local time when each tone was received. This is done by applying a *cross-correlation* algorithm to each of its recorded sound samples (*Computation* step). Cross correlation is a standard signal processing technique that searches for the best match between a recorded sample and a reference signal (the best match is indicated by a sharp peak in the cross correlation). It has been widely used in various ranging systems, for example, [4], [6], [10], [17], [19], [20], [25] and is computed as

$$CC(t_0) = \frac{\sum_{t \in W} [X(t) - \overline{X(t)}][T(t - t_0) - \overline{T(t - t_0)}]}{\sqrt{\sum_{t \in W} [X(t) - \overline{X(t)}]^2 \sum_{t=1-L}^{0} [T(t) - \overline{T(t)}]^2}},$$

where $X(t)$ is the recorded sound sequence, $T(t), t \in [-L+1, 0]$ is the reference signal of length L,

$$\overline{X(t)} = \frac{\sum_{t \in W} X(t)}{L}, \quad \overline{T(t)} = \frac{\sum_{t=1-L}^{0} T(t)}{L},$$
$$W = [t_0 - L + 1, t_0].$$

For each t_0, the computation complexity is $O(|W|) = O(L)$, and thus $O(B * L)$ if we have B recorded sound samples. Finally, once the cross-correlation peaks are found at each phone, the corresponding tone arrival time stamps are exchanged between the two phones, for example, via 3G or WiFi (*Measurement Exchange* step), and the distance can be computed.

3.2 Limitations

As pertains to MMG, existing phone-to-phone acoustic localization schemes all have the same drawbacks: they are too slow to serve the needs of MMG games due to the large measurement delay, and they are not designed for highly mobile systems:

- The traditional execution strategy of *Play/Record then Compute and Exchange* is unsuited for high-frequency measurements, as it takes too much time.
- Cross correlation is well known to be computationally expensive, as it grows superlinearly in the size of the reference signal L and the recorded sound sequence T. For example, although [19] applied an energy detection module to reduce the search space in the sound sequence T, the reported time for one measurement is still 800 ms.
- Existing protocols are not designed for highly mobile systems. They assume that during the course of a measurement, the phones remain stationary and if they are not, results will be erroneous.

Finally, in addition to these more fundamental reasons, there are also purely practical system design issues that prevent us from building MMG games on top of existing ranging infrastructure:

- Commodity phones do not offer tight timing guarantees on the operation of the playing and recording controls, and thus, systematically incur large and unpredictable delays. Specifically, as we show in Section 8, there is a lag of high magnitude and high variance between the time a playback is initiated and the actual time the tone is played. For example, the lag is 60-90 ms for both Nexus One phone running Android 2.3.4 and Samsung Focus phone running Windows Phone 7.5. This caps the frequency at which phones can emit tones. Based on our empirical observations, this lag occurs so commonly among commodity smartphones that we cannot simply discount it as a software- or device-specific phenomenon.

4 FAR System Design

In this section, we introduce FAR, our fast, accurate, and robust distance measurement system that serves as an API to phone-to-phone MMG games like SwordFight. Meeting the requirements in Section 2 in the face of the limitations in Section 3.2 is not easy and drives us to a novel system design as well as numerous optimizations at both the algorithm and systems level.

The key goal of the FAR system is to systematically improve upon existing phone-to-phone ranging schemes in two directions—1) by making them *faster* and reducing delays (See Section 5); and 2) by making them more *robust* in the face of mobility (See Section 6)—while at the same time keeping the required degree of accuracy.

Fig. 1 illustrates the FAR architecture that achieves these goals. Upon initiation, each of the major stages *Playing*, *Recording*, *Computation*, and *Exchange* happen continuously. Section 5 discusses the dynamic arrangement of these stages. *Computation* contains the main algorithmic contributions of our work. It is sequenced as follows:

1. An ambient noise filter mitigates environmental noise such as from shouting, talking, and crowd noise common during gameplay.
2. One or more lightweight autocorrelators work in tandem to detect the presence of tone signatures.
3. Smoothing reduces the impact of anomalous local minima, which are the result of sound distortion played and captured by commodity hardware.
4. Multipath readings (e.g., reflection from nearby objects) are filtered out to identify the actual tone

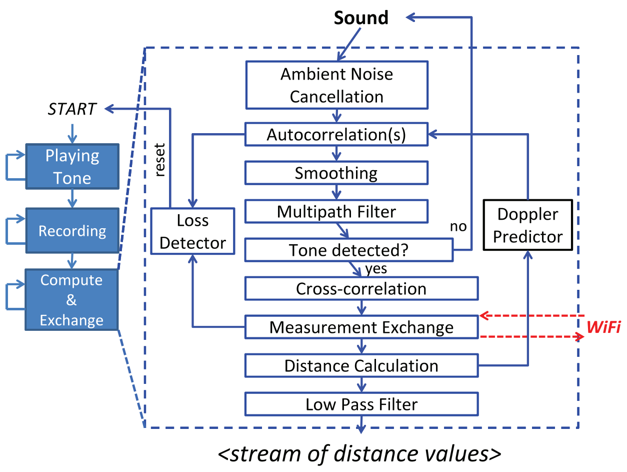


Fig. 1. FAR architecture.

received. If no tones are detected, expensive cross correlation is not engaged.

5. The cross-correlator identifies the tone reception time stamp for accurate timing information.
6. The phones exchange their measured time stamps via WiFi. Failure to receive data here (due to signal jamming or tone misdetection in the autocorrelation step) over prolonged durations results in the loss detector automatically signaling a protocol reset.
7. The exchanged data are used in distance calculation. The distance calculation also informs the Doppler predictor, which adjusts the number of autocorrelators to use in the next round. This enhances measurement accuracy at high mobility.
8. A final low-pass filter smooths the calculation before emitting the distance value to the game.

5 Fast Distance Measurements

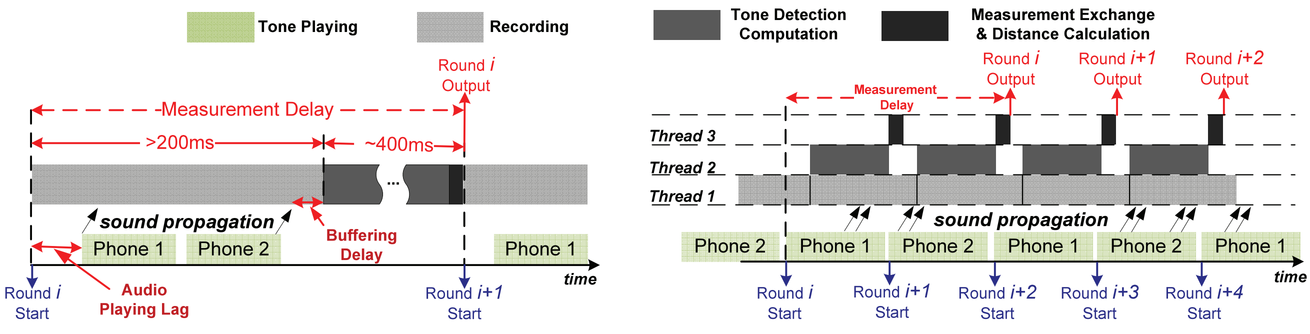
In this section, we present the set of techniques that allows FAR to conduct distance measurements at high frequency and with low lag. To see how our techniques impact lag and frequency, it is useful to again consider the traditional execution strategy as illustrated in Fig. 2a. In the traditional execution strategy, the lag of each individual measurement is comprised of the following components:

- *Tone Length*. Time spent on sending out each tone. Since one measurement requires that both phones send out tones, the measurement lag includes two tone lengths.
- *Audio Playing Lag*. Delay between the time of calling the `Play()` API and the time when the tone is actually sent out from the speaker. This delay is observed across platforms on both Android and Windows Phone.
- *Sound Propagation Delay*. Time it takes for the sound to reach the other phone over the air.
- *Buffering Delay*. Time from when the tone has been fully recorded until when the audio driver passes the recorded buffer to the application layer. The reason for the delay is that the audio driver delivers a buffer only when the buffer is full, regardless of whether the tone has arrived.
- *Tone Detection Computation Time*. Time required to compute on a recorded buffer and determine the exact time stamp when the tone arrived.
- *Measurement Exchange and Distance Calculation*. Time it takes for the two phones to exchange their time stamps, and compute the final distance measurement result using (1).

If we want to achieve measurement frequency above the 10 Hz required for MMG games, we must address these delays across the board. FAR accomplishes this through an efficient system architecture and set of core acoustic signal processing algorithms. Section 5.1 discusses how we substantially reduce the tone-detection computation time by a very efficient algorithm. Section 5.2 introduces pipelining to further improve measurement frequency, and streaming to circumvent widespread system lags. Lastly, Section 5.3 summarizes how we reduce other components of delay.

5.1 Fast Tone Detection Algorithm

As discussed in Section 2, the problem of relying upon cross correlation for tone detection is that its computation overhead is high. To find the start time of a tone of length L in a buffer of size B, a total of $O(B*L)$ multiplications are required. Our key observation is that instead of running the cross correlation directly, we can first use a computationally much more efficient *autocorrelation* primitive.



(a) Traditional Execution Strategy

(b) FAR Execution Strategy

Fig. 2. Execution strategies.

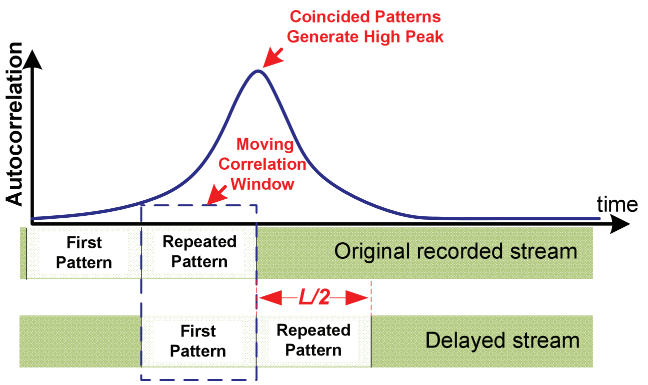


Fig. 3. Autocorrelation-based Tone Detection. When sliding the correlation window, at a certain time, the repeated pattern in the original stream will coincide with the first pattern of the delayed stream, which generates a high correlation peak.

Autocorrelation's strength is that it is much more efficient than cross correlation at finding *repeating patterns* in data sequences. However, autocorrelation does not give us an exact estimate of the tone's location because of its much flatter peak style. Therefore, after applying autocorrelation, we employ (as a second and third step) a simple smoothing algorithm and then a small-scale cross correlation in a narrow search window centered around the smoothed autocorrelation peak. We now discuss these steps in detail.

Step 1: Autocorrelation-based Tone Detection. The basic methodology of autocorrelation we use is shown in Fig. 3. Autocorrelation can only detect correlation of a signal to itself. Therefore, we let each phone send out a tone consisting of a pseudorandom sequence followed immediately by an exact copy of this sequence. To detect this tone, the receiver phone records the incoming stream $X(t)$, and then generates a second stream $Y(t)$, which is an identical copy of $X(t)$, except that it is delayed by half the tone length L, i.e, $Y(t) = X(t - L/2)$. The receiver now correlates $X(t)$ and $Y(t)$ (i.e., the incoming steam is correlated with a copied, delayed version of itself) in a moving correlation window of length $L/2$. This is done by incrementally computing for each time step t_0 the function

$$R(L/2, t_0) = \frac{\sum_{t\in W}[X(t) - \overline{X(t)}][Y(t) - \overline{Y(t)}]}{\sqrt{\sum_{t\in W}[X(t) - \overline{X(t)}]^2 \sum_{t\in W}[Y(t) - \overline{Y(t)}]^2}},$$

where $\overline{X(t)} = \frac{\sum_{t\in W} X(t)}{L/2}$, $\overline{Y(t)} = \frac{\sum_{t\in W} Y(t)}{L/2}$, and $W = t_0 - L/2 + 1, t_0]$. Higher autocorrelation values correspond to closer correlations of $X(t)$ and $Y(t)$ at t_0. The idea is that because phones send two identical sequences in each tone, at a certain time slot, one of the repeated patterns in $X(t)$ will match a first pattern in the delayed stream $Y(t)$, thus generating a large autocorrelation peak as shown in Fig. 3.

The benefit of the above method is that we can compute $R(L/2, t_0)$ in $O(1)$ time. To see this, consider that the above expression can be rewritten as follows:

$$R(L/2, t_0) = \frac{\frac{L}{2}\widehat{XY} - \widehat{X}\widehat{Y}}{\sqrt{[\frac{L}{2}\widehat{XX} - \widehat{X}\widehat{X}][\frac{L}{2}\widehat{YY} - \widehat{Y}\widehat{Y}]}},$$

where

$$\widehat{XY} = \sum_{t\in W}[X(t)Y(t)],\ \widehat{XX} = \sum_{t\in W}[X(t)X(t)],$$

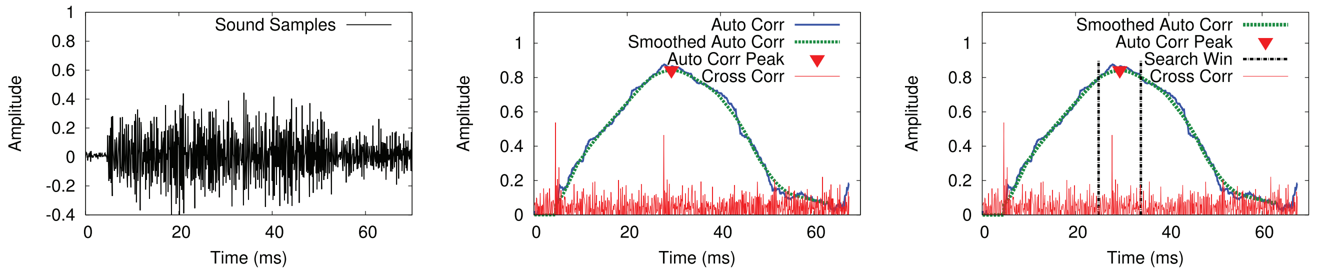
$$\widehat{YY} = \sum_{t\in W}[Y(t)Y(t)],\ \widehat{X} = \sum_{t\in W} X(t),\ \widehat{Y} = \sum_{t\in W} Y(t).$$

Note that all these variables can be computed in $O(1)$ time with a standard moving window method. Taking $\widehat{X}$ as an example, given $\widehat{X}(t_0)$, we can compute $\widehat{X}(t_0 + 1)$ as follows:

$$\widehat{X}(t_0 + 1) = \widehat{X}(t_0) + X(t_0 + 1) - X(t_0 - L/2 + 1).$$

Since $\widehat{X}(t_0)$, $X(t_0 + 1)$, and $X(t_0 - L/2 + 1)$ are all known, $\widehat{X}(t_0 + 1)$ is computed in $O(1)$. The other variables can be updated in the same way. $R(L/2, t_0 + 1)$ can, thus, be obtained in $O(1)$ time based on the variables at t_0. For a buffer of size B, this autocorrelation solution has complexity $O(B)$, which is much faster than $O(B * L)$ via traditional cross correlation.

Inaccuracy of Autocorrelation-based Tone Detection. Unfortunately, we cannot rely solely on autocorrelation to find the tone peak as the result will be inaccurate. Fig. 4b shows the output of the autocorrelation step. As can be seen, there is a noticeable offset between the autocorrelation peak and second cross-correlation peak (a sharp line). In an ideal scenario, these two peaks should perfectly align. The reason for this discrepancy is twofold. First, autocorrelation fundamentally has a much flatter peak. Intuitively, if a time slot's autocorrelation value is high, its neighboring time slots' values are also high. Second, the offset can arise



(a) Captured Sound Samples (b) Auto- and Cross-correlation Peaks (c) Peak Search in a Small Window

Fig. 4. Fast Tone Detection Algorithm. (a) Recording of received tone with repeating patterns. (b) While cross-correlation peaks are sharp, the autocorrelation peak is flat and has an offset to the pattern's actual location. (c) Cross correlation is used in a small window around the autocorrelation peak to find the precise location of the pattern.

due to signal distortion of the tone after propagation. This can result in up to 10-cm error in the final result even in quiet environments as we show in the evaluation. For these reasons, we cannot rely exclusively on autocorrelation for tone detection. Rather, we seek to combine the respective benefits of cross correlation and autocorrelation.

Step 2: Smoothing of Autocorrelation Peak. To obtain cleaner peaks, we smooth the autocorrelation curves within a 10-ms window. Note that this smoothing process is necessary as the raw autocorrelation output can have many local maxima due to environmental noise.

Step 3: Small Scale Cross correlation. We interpret the smoothed autocorrelation curve as an indicator for the location of the tone, then we search in a small window S around the autocorrelation peak using standard cross correlation to detect the precise tone location (see Fig. 4c). The computation overhead of the combined algorithm now is $O(B + |S| * L)$, which is still much smaller than $O(B * L)$, assuming that the autocorrelation indicator allows us to choose sufficiently small $|S|$.

Choosing the Search Window Size $|S|$. The above discussion reveals an interesting design tradeoff concerning the selection of the window size $|S|$. A smaller $|S|$ yields faster cross-correlation computation, but risks misdetecting a tone altogether because it is outside the searching window. This is especially true in noisy environments, where the offset between cross correlation and autocorrelation peaks can be large. Based on our empirical measurements, we set our window size to a very conservative size of 100 samples (offset by -80 to $+20$ around the detected peak), which is sufficient even at high noise. The fact that the search window is not symmetric around the detected peak is probably due to reflected signals, which may generate higher autocorrelation peaks after the direct signal's peak.

5.2 Pipelined Streaming Execution Strategy

With the preceding reductions in tone detection computation time, we next introduce FAR's *pipelined streaming execution strategy* for boosting measurement frequency. Its two key improvements upon the traditional execution strategy are that 1) it employs pipelining to increase measurement frequency by overlapping computation time with tone transmission, and 2) it employs streaming to minimize measurement lag by removing Audio Playing Lag. Fig. 2b illustrates the FAR execution strategy.

5.2.1 Pipelined Execution to Mask Computation Time

We devise a pipelined execution strategy that uses separate threads to handle *Playing, Recording, Tone-Detection Computation*, and *Measurement Exchange* (Fig. 2b). In the recording thread, the Audio Recorder periodically fills in a predefined audio buffer. For example, the minimum required buffer size for Nexus One running Android 2.3.4 is 1,024 Bytes, which corresponds to 512 sound samples. Once the buffer is filled, the computation thread processes the buffered sound samples, while the recording thread waits for the next buffer. Once computation is finished, the measurement exchange thread exchanges the detected tone arrival times with the other phone using WiFi and outputs the final distance measurement. The net effect is that tone-detection computation is now fully overlapped with the exchange of sound waves (including recording delay, playing delay, and signal propagation delay). Given that in our system, these two sets of operations take approximately equal amount of time, the pipelined execution strategy effectively doubles FAR's measurement frequency.

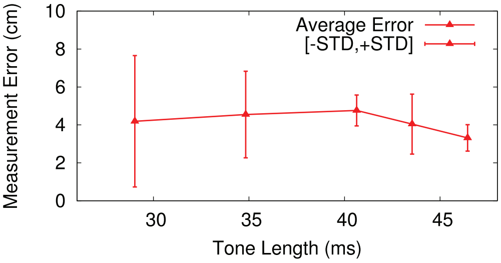


Fig. 5. Impact of different tone length. A tone length of 46.4 ms has good accuracy and low variance.

Note that when pipelining, it is of critical importance that the tone-detection computation time is tightly bounded. If the computation on one buffer is not finished before the next buffer arrives, the computation overhead will accumulate over time, resulting in buffer overflows and lost sound samples.

5.2.2 Streaming Execution to Mitigate Audio Playing Lag

Instead of blocking on computation completion as the traditional strategy does, FAR implements a streaming mode to continuously send and record tones. The `Play()` API of the Audio Driver is called at a frequency similar to the tone length, thus one tone will be sent right after another. This helps in eliminating the Audio Playing Lag from the overall measurement time. Similarly, we also implement the Audio Recorder to work in a streaming fashion, continuously recording sound samples for computation. As a result, if there is no lag between tones, measurements can theoretically be conducted at a frequency equal to $1/L$, provided tone detection is fast enough to support such a rate.

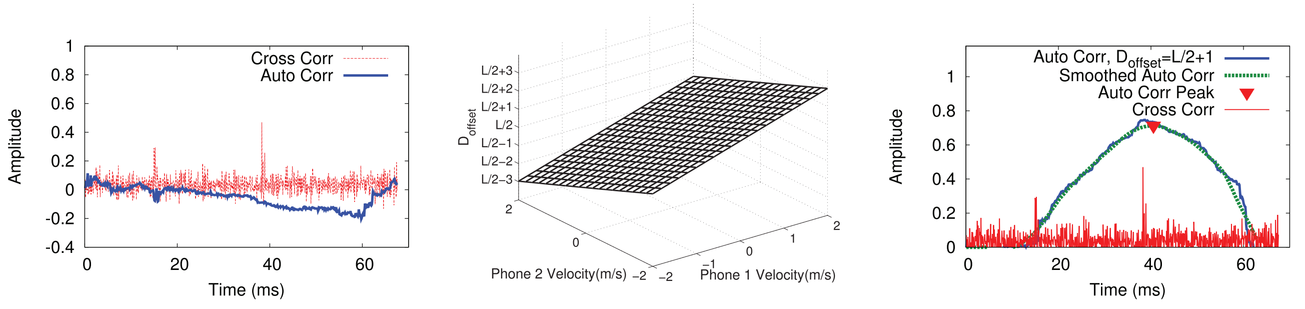
5.3 Further Optimizations

In addition to the techniques discussed in the previous sections, our system implements several additional optimizations to reduce delay.

Tone Length Minimization. Choosing the right Tone Length is important. Choosing a small value can negatively affect measurement accuracy, but choosing a large value not only adds lag to the overall execution, but also increases the complexity of the cross-correlation computation, thereby reducing the measurement frequency. We conducted extensive experiments over different tone lengths and found that using a tone length of 512 sound samples (corresponding to 46.4 ms at a 11.025-kHz sampling rate) achieves good accuracy and stability (see Fig. 5), while still allowing us to keep the measurement frequency above 10 Hz. The tone of 46.4 ms is similar to that used in [17].

Buffering Delay Minimization. The buffer is pushed to the application layer whenever it is full. The buffering delay is, therefore, a random period in $(0, B)$. To minimize this delay in FAR, we use the phone's minimum buffer size (which is 46.4 ms (1,024 Bytes) on Nexus One).

Networking Operation. We use UDP over WiFi, adding a delay of around 2-4 ms per exchange.



(a) Doppler Effect Example.

(b) Phone Velocity *vs.* D_{offset}.

(c) Peak Recovery by Changing Offset.

Fig. 6. Doppler effect and recovery. (a) The autocorrelation peak is destroyed by the Doppler effect. The interval between the two cross-correlation peaks is $L/2+1$ rather than $L/2$. (b) Given player speed is limited, L_{recv} falls into a limited range. (c) Changing the offset of the autocorrelation to $\mathrm{L}/2+1$ can detect the peak.

Tone Overlapping. One attempted optimization that we were unable to execute in practice was tone overlapping. Instead of spacing the two phone's tones over time, we tried to overlap them in time, but send them on different audio frequencies. Unfortunately, this approach failed because the local speaker's high power masks the remote phone's signal, which generates detection errors.

6 System Robustness

Besides measurement frequency and accuracy, another important challenge we need to tackle is the robustness of the system. A real-time motion game cannot be fun if there are frequent measurement errors. In this section, we tackle the possible vulnerabilities a practical system could suffer in real game play environments.

6.1 Mobility Robustness

One unique challenge to distance estimation of motion gaming is the Doppler effect caused by player movement. Doppler effect happens when there is a relative movement between the sound player and recorder. Intuitively, when two phones are moving toward each other, the sound wave arrives at the recorder earlier than expected, so it appears *compressed*. However, because the recorder is recording the sound at a constant rate, the recorded sound samples are fewer than expected, which means in the recorded version of the tone, the repeating pattern has a shorter length. Thus, the offset used in the autocorrelation calculation needs to be shortened. Similarly, when the phones are moving further away, the tone is *diluted*, and thus, a longer offset is needed.

We define the term *Doppler offset* D_{offset} as the number of samples between the two cross-correlation peaks, i.e., the locations of the repeating patterns. In a scenario without Doppler effect, the Doppler offset should be equal to the repeating pattern's length, which is exactly half of the tone length, $D_{offset}=L/2$, and the autocorrelation of $X(t)$ and $X(t-L/2)$ should be calculated to detect the tone. If Doppler shift occurs, the repeating pattern will be separated by a D_{offset} not equal to $L/2$. In these cases, the autocorrelation of $X(t)$ and its delayed stream $X(t-D_{offset})$, which is $R(D_{offset},t)$, should be calculated to find the tone.

In Fig. 6a, we show an example of a dilution, in which the tone does not show an autocorrelation peak. A careful diagnosis uncovers that the offset is $L/2+1$ rather than the expected $L/2$. This means that an extra sample has appeared in the tone because of Doppler sound dilution. Thus, $D_{offset}=L/2+1$ is appropriate in this case.

Assume Phone 1 is sending the tone, Phone 1's velocity is v_{p1} (relative to earth), Phone 2's velocity is v_{p2}, Tone Length is L (so the repeated pattern length is $L/2$), and c is the speed of sound. The delay between the repeating pattern of received sound sequence D_{offset} can be expressed as

$$D_{offset}=\frac{c+v_{p2}}{c+v_{p1}}*L/2.$$

Fig. 6b shows the relationship between D_{offset} and the relative velocity of the two phones. We assume the maximum speed of a player's hand is 2 m/s, as we have observed from our experiments that a higher speed is extremely hard to reach. This is in accord with previous work [8] on human hand functions. Since this speed is not negligible compared to the speed of sound ($c=343.2\ \mathrm{m/s}$), we can see from the figure the possible range of the Doppler offset as $D_{offset}\in[L/2-3,L/2+3]$.

Recovery from Doppler Effect with parallel autocorrelators. To recover the correct autocorrelation peaks, a simple method is to calculate the autocorrelation with the appropriate offset. As an example, in Fig. 6c, we show that we can recover the peak by calculating $R(L/2+1,t)$, rather than $R(L/2,t)$. Given that $D_{offset}\in[L/2-3,L/2+3]$, we can set up a maximum of seven parallel autocorrelators to catch any practical case of Doppler shift.

Parallel autocorrelators have both advantages and disadvantages. The advantage is that more autocorrelators increase the detection ratio. The disadvantage is that more autocorrelators lead to longer computation time. Therefore, one problem is to find the optimal number of autocorrelators $\hat{x}$ that maximizes the tones detected per unit time for a given velocity v, i.e.,

$$\hat{x}=\underset{x}{\operatorname{argmin}}[D_v(x)\max(b,T(x))],$$

where b is the communication bound, $D_v(x)$ is the detection ratio with x autocorrelators at velocity v, and $T(x)$ is the completion time for x parallel autocorrelators.

Unfortunately, each additional autocorrelator causes an additive increase in computation time. In response, we

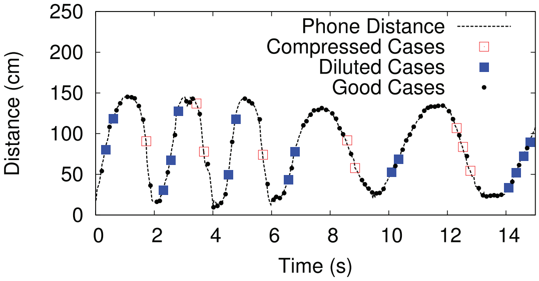


Fig. 7. Doppler effect causes sound dilution and compression errors. However, we are able to predict these errors based on recent velocity.

apply a *predictive Doppler estimate procedure*. The idea is to predict the likely Doppler shift based on the recent history of distance measurements, and only execute the autocorrelators likely to match the predicted velocity shift.

Fig. 7 shows a fast movement case with a maximum player hand speed of 2 m/s (and, thus, phone-to-phone relative speed reaches 4 m/s). We show the two phones' distance change over time, and mark the tones affected by the Doppler effect. The Doppler-shifted tones are divided into two groups: diluted cases and compressed cases. As evident from the figure, compressed cases occur when the distance is decreasing, and diluted cases happen when distance is increasing. Exceptions to this rule happen rarely.

Thus, we can use two groups of autocorrelators to deal with the two groups of Doppler effect cases: one group of autocorrelators $R(L/2-1,t), R(L/2-2,t), R(L/2-3,t)$ for compressed cases, and the other group of autocorrelators $R(L/2+1,t), R(L/2+2,t), R(L/2+3,t)$ for diluted cases. For each buffer of recorded samples, we always run the original autocorrelator $R(L/2,t)$. At the same time we predict the players' relative velocity based on the recent distance change history, then apply the appropriate group of autocorrelators, as shown in Fig. 8. We can reduce the total overhead by nearly half with this method. In fact, our evaluation suggests that almost all of D_{offset} are within $[L/2+2, L/2-2]$, and thus, only three parallel autocorrelators are needed to recover from the vast majority of Doppler shifts.

The prediction-based method might not be correct when the player is changing direction. However, the user's speed naturally slows when changing direction, which will decrease the likelihood of experiencing Doppler shifts. As shown in Fig. 7, almost all of the tones at the peaks or troughs are good cases.

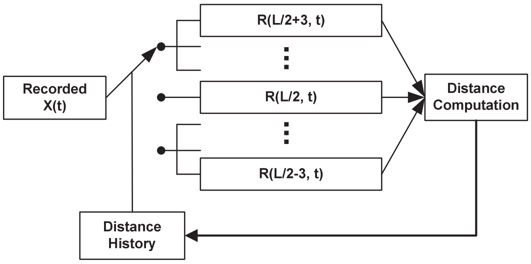


Fig. 8. Predictive parallel autocorrelators. The distance history is leveraged to predict whether sound will be diluted or compressed, and subsequently which group of autocorrelators to use.

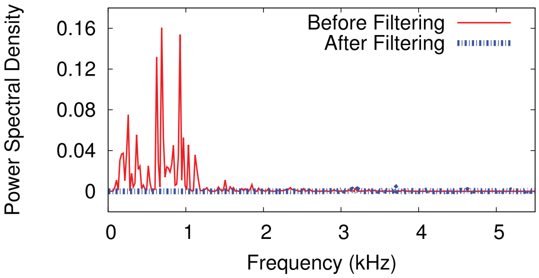


Fig. 9. The frequency map of a recording of ambient sound from a crowded area before and after filtering.

6.2 Environmental Robustness

There are several environmental factors which can impair distance measurements. We address these presently.

6.2.1 Robustness to Ambient Noise

To study the character of ambient noise, we recorded sound from both indoor and outdoor noisy scenarios. One finding is that the frequency for these sounds is usually below 2 kHz. Thus, in our system we apply a high-pass filter (a ninth order Butterworth filter) to filter out ambient noise. Fig. 9 shows the spectrum density map of a 1-minute recording of a crowded place, before and after applying our filter, respectively. It is clear that the high-pass filter functions well in eliminating ambient noise.

6.2.2 Robustness to Multipath

Multipath effects can also impact the robustness of tone detection if the players are in a relatively small room or hallway. The reflected signal may generate correlation peaks or degrade the main component. Fig. 10 shows a tone with an observable multipath effect. There is a lower autocorrelation peak at around 100 ms, which is caused by the multipath components. We observe that the multipath components usually have low power and signal quality after reflection or scattering. Note that the power threshold might not work well if there is high ambient sound. However, after applying the high-pass filter, we can efficiently reduce the ambient sound power level. In our implementation, we empirically set the power threshold to 0.02 and peak level threshold to 0.4, which we show to be sufficient to filter out the multipath components experienced when playing in actual scenarios. In our evaluation section, we conduct experiment in a small room to show the performance.

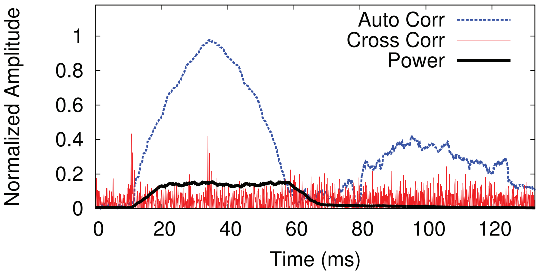
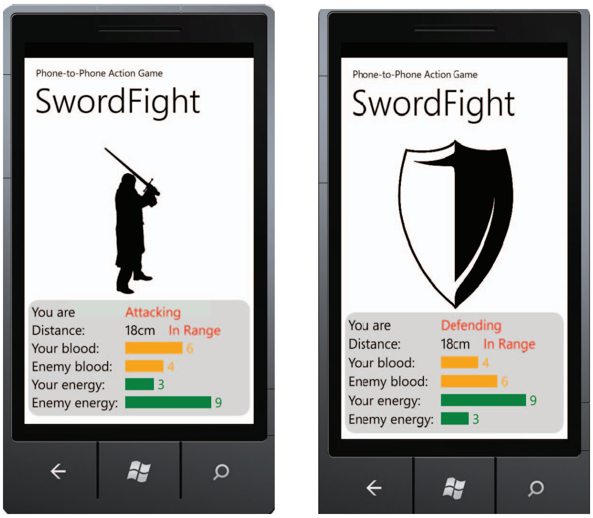


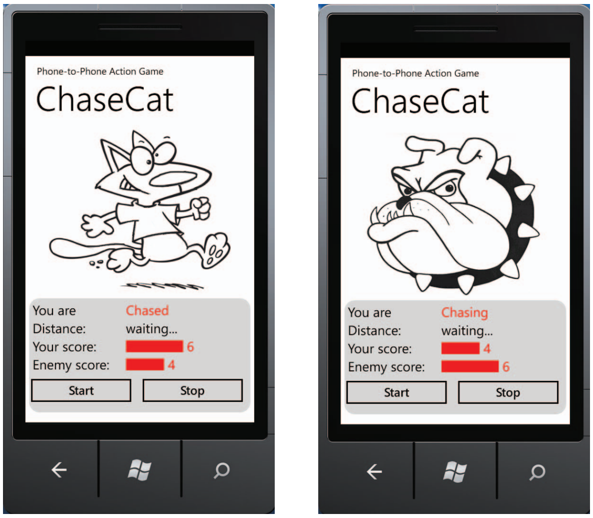
Fig. 10. A multipath component generates a smaller autocorrelation peak (at around 100 ms) after the correct peak, but with a much lower power level.


Phone-to-Phone Action Game
SwordFight
You are Attacking
Distance: 18cm In Range
Your blood: 6
Enemy blood: 4
Your energy: 3
Enemy energy: 9

Phone-to-Phone Action Game
SwordFight
You are Defending
Distance: 18cm In Range
Your blood: 4
Enemy blood: 6
Your energy: 9
Enemy energy: 3

(a) Attack (b) Block

Fig. 11. SwordFight game screenshots.


Phone-to-Phone Action Game
ChaseCat
You are Chased
Distance: waiting...
Your score: 6
Enemy score: 4
Start Stop

Phone-to-Phone Action Game
ChaseCat
You are Chasing
Distance: waiting...
Your score: 4
Enemy score: 6
Start Stop

(a) Cat (b) Cat Chaser

Fig. 12. ChaseCat game screenshots.

6.2.3 *Robustness to Occasional Tone Loss*

FAR also provides failsafe methods to deal with occasional tone losses. Since the distance calculation requires the tone measurement from both phones' computation result, it is important to keep the two measurements synchronized. In particular, if a tone is not detected, the phone should be alerted of the situation. We achieve this by a *binary coding* method: we assign a unique pseudorandom sequence for each phone. If detection is flawless, each phone will see alternating codes. Thus, if a phone receives any two consecutive tones whose codes are the same, the phone detects a missed tone condition and accounts for this in the distance calculation.

Coding beyond binary (e.g., ternary) is not necessary, because multiple consecutive missing tones are detected by simple threshold on the arrival interval between two consecutive tones. When such cases arise, the system will enter *Failsafe mode*. In Failsafe mode, one phone will send two tones separated by a much larger interval, for example, 250 ms. Both phone will then observe the large interval and reset their state.

7 Game Design

In this section, we briefly describe two novel games we have developed on FAR to date.

SwordFight. In SwordFight, the core gameplay works as described in the introduction. As an alternative to the default mode, SwordFight also offers a gameplay mode with gesture-based attacks and blocks. In this mode, a player thrusts her phone toward the other opponent to attack (phone screen parallel to earth), or holds the phone like a shield to block (phone screen perpendicular to earth). For a hit to succeed, the hit must be made within 20 cm of the opponent's phone while the opponent is not blocking. One point is awarded for a successful regular hit. A successful block is awarded two points, but continuous blocking will cause the blocker to exhaust her ability to block. The gestures are detected by the inertial measurement unit (IMU). To ensure that gestures are oriented toward the opponent, the digital compass is used. At the start of a fight round, the players are instructed to point their phones toward one another and the phones' digital compass orientations are recorded. During the fight round, the digital compass is read to ensure that Attack and Block gestures are actually oriented toward the opponent. Screenshots of SwordFight gesture-based mode are shown in Fig. 11.

ChaseCat. ChaseCat is a fast round-based game. In each round, one player is randomly chosen as the Cat and the other player is chosen as the cat chaser, the Dog. The Dog player attempts to get her phone as close to the Cat player's phone as possible, while the Cat player tries to get her phone as far from the Dog player's phone as possible.[2] A random 15-25 seconds after the round starts, players are instructed to Freeze by an audio cue. If the Cat is within 30 centimeters of the Dog during Freeze, then the Dog catches the Cat and scores a point. Otherwise the Cat escapes and scores a point. Movement during Freeze is detected by distance and acceleration measurements. Screenshots of ChaseCat are shown in Fig. 12.

A design consideration for MMG games is that, unlike traditional mobile games, players focus on each other rather than the phone screens. Therefore, timely cues are important for conveying feedback to the players. In both of our games, the iconography is large and easily glanceable. SwordFight uses the phones' vibrate mode every time a point is scored.

Note that in the case of both games, distance measurements are expected to be accurate up to several centimeters. Our evaluation results in the next section show that FAR is accurate at this resolution. The maximum distance that can be supported is a function of the maximum volume of the phone speakers and the sensitivity of microphones. The current platforms we use happen to have an effective measurement range of 2 meters, so this is the current effective limit of gameplay. Out-of-range occurrences can be detected by sequences of dropped tones and low SNR.

[2] We recommend players plant their feet during the game, but this has proven difficult to enforce.

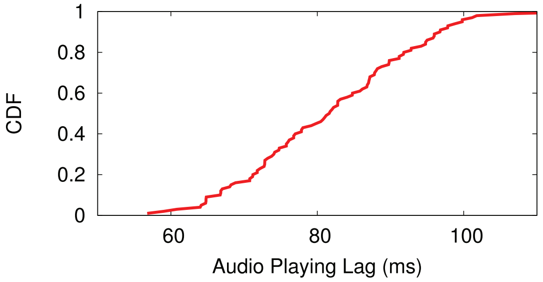


Fig. 13. Audio Playing Lag distribution.

8 Implementation

We implement both our games on two phones: Nexus One running Android 2.3.4 and Samsung Focus running Windows Phone 7.5.

System Configuration. We use two maximum length sequences (m-sequences) to generate unique tones for the two phones. Since the m-sequence is well known for its distinctive peak when applying cross correlation, it helps us to accurately identify the tone's location in the recorded sound. The sequence is comprised of 256 16-bit short integers. In the tone, the sequence is repeated once to support the autocorrelation-based detection method (Section 5.1). The frequency range of the tones are [0, 11.025 kHz] (Nexus One) and [0, 16 kHz] (Samsung Focus). The Audio Recorder samples the sound at a rate of 11.025 kHz (Nexus One) and 16 kHz (Samsung Focus), which is sufficient to support 1-cm measurement granularity. As discussed, to achieve a good balance of robustness, accuracy, and speed, we choose a tone length of 46.4 ms. As mentioned in Section 5, the buffer size of the Audio Recorder can incur a buffering delay. To minimize this delay, we use the minimum allowed buffer size in both phones: 1,024 Bytes (46.4 ms with 11.025-kHz sampling rate) for Nexus One, and 2,000 Bytes (62.5 ms with 16-kHz sampling rate) for Samsung Focus.

Randomness of Audio Playing Lag. As mentioned in Section 5.2, the Streaming Execution Strategy mitigates Audio Playing Lag. However, it turns out that the commodity devices we have encountered exhibit an additional random audio playing lag artifact. Fig. 13 shows that the audio playing lag is randomly distributed between 64 (5 percentile) and 99 ms (95 percenttile). Thus, to prevent two consecutive tones from overlapping with each other, we add a 35-ms *guard period* when playing each tone. The interval between two consecutive `Play()` calls is then $L + 35 \text{ ms} = 46.4 \text{ ms} + 35 \text{ ms} = 81.4 \text{ ms}$. In fact, given our reductions in computation and communication overhead which are no longer the bottleneck, it is this random system artifact that currently prevents us from increasing the measurement frequency from $1/81.4 \text{ ms} \approx 12 \text{ Hz}$ to a theoretical $1/L = 1/46.4 \text{ ms} = 21.6 \text{ Hz}$.

9 Evaluation

We demonstrate through our evaluation that FAR is fast, accurate, and robust, and that it is suitable for MMG games. The primary metrics are as follows:

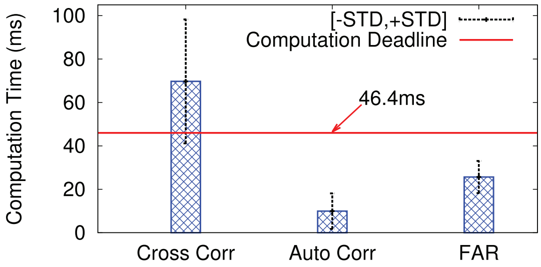


Fig. 14. Compute time needed to process a 512-sample audio recording.

- *Measurement Frequency.* Is FAR efficient enough to support fast measurement? How fast can FAR make measurements?
- *Measurement Accuracy.* How accurate are the measurements?
- *Measurement Robustness.* How well does FAR handle noise and motion?
- *Gameplay.* Do games based on FAR work in practice?

We conducted experiments with two Nexus Ones in a room measuring 14 m by 7 m by 2.5 m that typically serves as a conference room. Static measurements are verified against manual ruler measurements. For dynamic measurements, the main comparable system used is the Xbox Kinect, a dedicated fixed-infrastructure motion tracking system for console gaming. Lastly, as real-world validation, we invited public users to play SwordFight and ChaseCat on four separate occasions. The highlights of the evaluation are as follows:

- FAR achieves a measurement frequency of at least 12 Hz. In fact, FAR is fast enough to reach the limits imposed by the phone's audio driver.
- The median measurement error is 2 cm. During times of rapid movement, FAR compare favorably to Kinect for both isolated player measurements and in-situ gameplay measurements.
- Doppler effects, high noise, and multipath are effectively neutralized.
- Experiences of over 50 game players suggest that FAR is a good match for motion gaming, and that SwordFight and ChaseCat are fun to play.

9.1 Measurement Frequency

We show that FAR is efficient enough to take measurements as fast as the phone audio driver can support. We evaluate FAR's feasibility in supporting the pipelined streaming execution strategy, and then report on the achieved frequency.

Feasibility. The fundamental limitation of previous approaches is that the detection of tones requires intensive computation, and thus is not feasible within a pipelined streaming execution strategy bound by tight deadlines. Fig. 14 illustrates that FAR is able to comfortably satisfy the strict deadline of 46.4 ms corresponding to a recording of 512 samples. The traditional cross-correlation method is by far the slowest at 70 ms.[3] Missing the computation deadline

[3] Note that 70 ms is much faster than results reported in [19] because we only process a 512-sample recording, while [19] requires a much larger recording due to its traditional execution strategy.

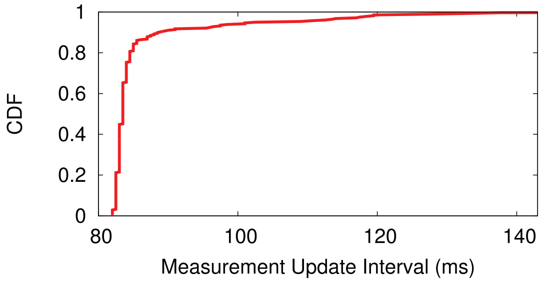


Fig. 15. Measurement Update distribution. More than 80 percent of the time, a new update arrives within 84 ms of the previous update.

incurs serious buffer overflow problems, leading to loss of sound samples and, thus, measurements.

Correlation computation can also be computed in the spectrum domain. To compare it with our algorithm, we implemented a spectrum domain correlation algorithm using a widely used Fast Fourier Transform (FFT) library.[4] Compared to FAR, cross correlation in the spectral domain still takes 150 percent more computation time, exceeding the available deadline.

Measurement Frequency. Fig. 15 shows the distribution of measurement updates for over 400 individual measurements as generated by FAR. The effective measurement frequency is 12 Hz over 80 percent of the time. As noted in Section 8, a random audio playing delay of [64, 99 ms] inhibits us from achieving a theoretical frequency of 21.6 Hz.

Measurement Lag. For each distance measurement, the lag is comprised of the following components: *Tone Length, Propagation Delay, Buffering Delay, Computation Time*, and *Measurement Exchange Delay*. For motion gaming, the distance between two phone does not exceed 2 meters. Therefore, the *Propagation Delay* is less than 6 ms. Fig. 16 shows the lag of 400 distance measurements when the phones are placed 50 cm apart. The measurements are sorted by their total delay. With the exception of the last 5 percent of cases with large network delays, the majority of measurements exhibit a total lag of around 100 ms. As expected, the delay due to the length of the tone is constant, and the network delay and propagation delay are very small. The buffering delay is uniformly distributed in (5, 46 ms).

9.2 Measurement Accuracy

Next, we juxtapose the measurement accuracy of autocorrelation (Auto Only) and FAR. We statically position the two phones in three typical phone-to-phone orientations at different distances. Fig. 17 illustrates the measurement accuracy of the two methods, each computed from over 400 experimental cases. It is clear that autocorrelation exhibits measurement error that is severe, whereas FAR's error is very low with a median of 2 cm and standard deviation of 1 cm. The figure also highlights that FAR is robust across varying phone orientations and distances.

9.3 Impact of Player Movement

We next investigate whether our system is capable of accurate measurements while players are moving. After exploring several possibilities including a studio motion capture system, we settled upon Kinect, an off-the-shelf motion capture system for the Xbox console gaming system, as the candidate comparable system. The reason Kinect is appealing is because it is commercially available, practical to use, and has already proven successful for infrastructure-based console gaming.

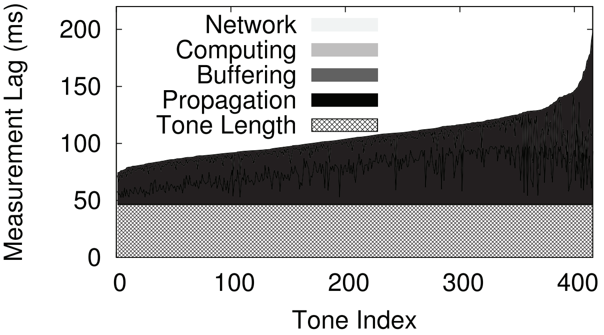


Fig. 16. Measurement lag composition. Several individual delays contribute to the time between when a tone is played and when a distance measurement is completed.

Kinect measurements were obtained with the Kinect SDK, which is a Kinect-to-computer interface that allows programmatic collection of the human skeleton coordinates as seen by the Kinect camera. We employed both isolated player experiments and in-situ real player experiments.

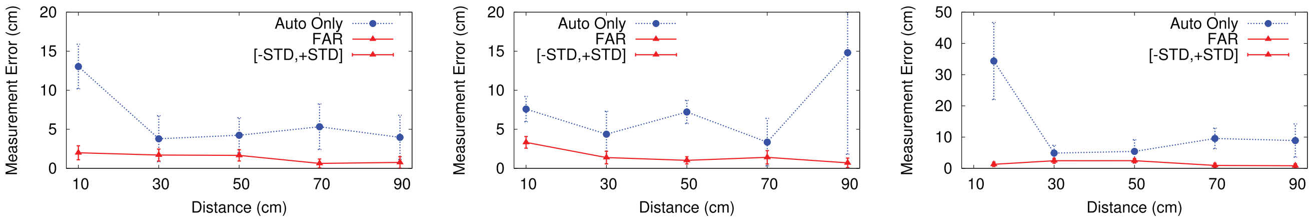
Isolated Player Movement. In isolated player experiments, we positioned the Kinect camera statically, and placed a subject in front of the camera with a phone running FAR in each hand. The subject was then asked to vary her hand position at her discretion. FAR performed phone-to-phone distance measurements while Kinect independently tracked the hands.

Fig. 18 illustrates the distance measurements simultaneously measured by Kinect and FAR for two representative traces. For the most part, FAR matches the Kinect motion path closely. In both traces, there are only five measurements out of 700 that deviate significantly from the Kinect estimate. In some instances, Kinect and FAR measurements can be offset from one another by up to 12 cm. The reason for this is actually a reflection of the Kinect SDK, which can exhibit 5-15 cm variance of the reported hand coordinate because it is ambiguous as to whether it measures distance from the base or tip of the hand. While an ideal ground truth would not have such issues, the results are encouraging enough to suggest that FAR compares favorably to a commercially successful motion gaming system.

In-Situ Gameplay Player Movement. In the in-situ gameplay experiments, six players were invited from the UC Santa Barbara Link Lab to conduct nine in-situ rounds of measurement. Players were each given a phone and asked to try out the SwordFight MMG game. During the game, FAR provided the in-game distance measurements while Kinect independently recorded their skeletal hand positions.

Fig. 19 compares the corresponding FAR and Kinect measurements for two representative traces. There are occasionally slightly more mismatches between FAR and Kinect during in-situ gameplay than during isolated player experimentation. This is due to the fact that players—in the

[4] https://sites.google.com/site/piotrwendykier/software.

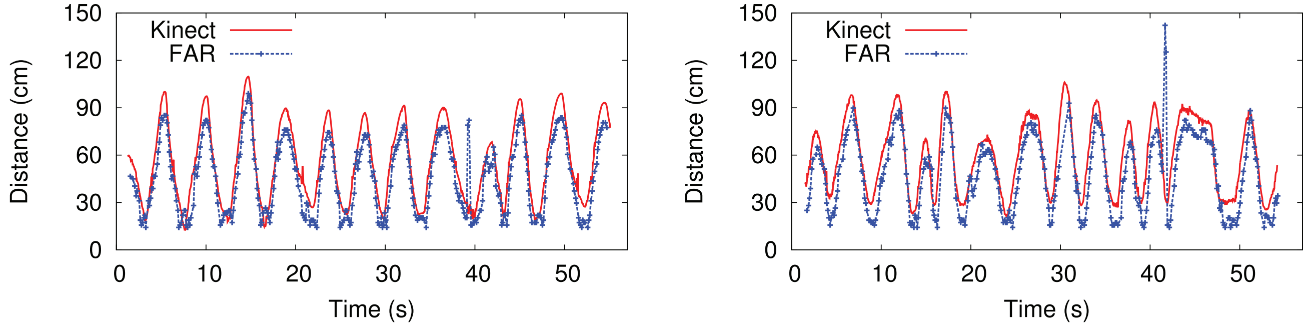


(a) Back To Back

(b) Head To Back

(c) Head To Head

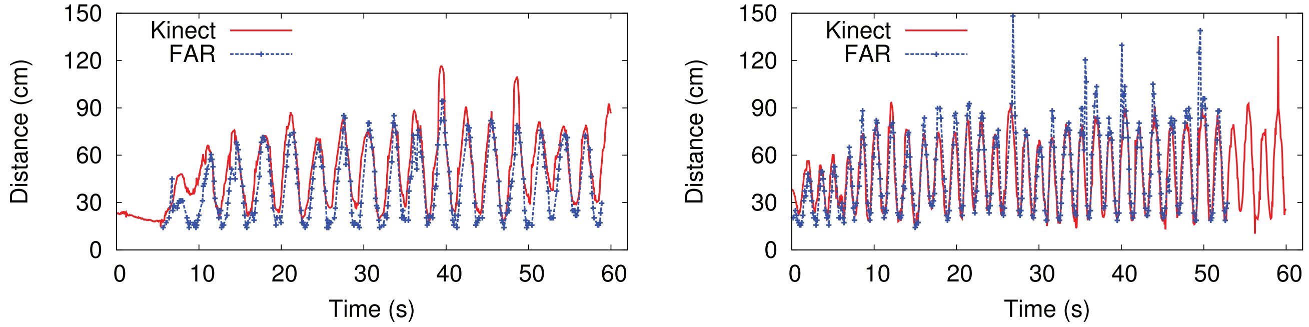
Fig. 17. Measurement accuracy in a large room.



(a) Experiment Case 1

(b) Experiment Case 2

Fig. 18. Isolated player measurements. Simultaneous distance measurements of FAR and Kinect when a single player is moving her hands freely with a phone in each hand.



(a) Experiment Case 1

(b) Experiment Case 2

Fig. 19. In-situ Gameplay measurements. Simultaneous distance measurements with FAR and Kinect of players engaged in SwordFight.

heat of competition—tend to grip or orient the phones such that mics or speakers may be temporarily blocked.

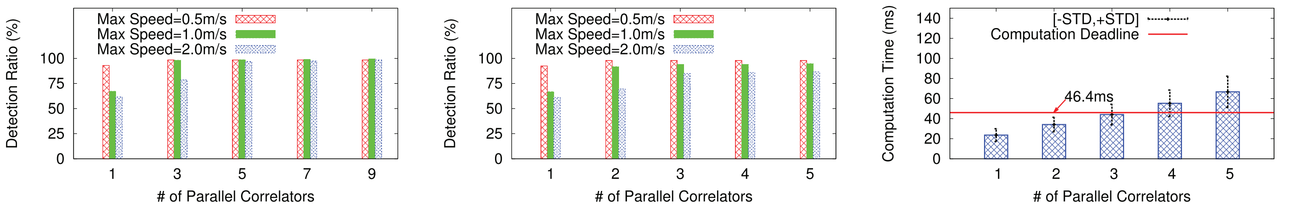
Interestingly, we have found that our system can in fact be more accurate than Kinect because Kinect cannot accurately track the hand when the player's hand is obstructed from the camera's view (e.g., by her own body or the opponent's body). These situations happen frequently in SwordFight and similar games. For example, two right-handed players facing off will often mean one player is poorly oriented toward Kinect. In these situations, FAR measures distance better because there are typically no obstructions between the two phones in the players' hands for games in which opponents are facing each other. In the in-situ experiments presented in Fig. 19, we asked that the players compensate for this Kinect limitation for the sake of experimentation: one player was instructed to play with the left hand, the other player was instructed to play with the right hand, and both players were instructed to not obstruct the line of sight between the Kinect and their phones. These patently artificial constraints on gameplay highlight why FAR may be potentially even more suitable than fixed infrastructure measurement systems for certain styles of motion gaming, despite infrastructure availability.

9.4 Measurement Robustness

We quantify FAR's effectiveness at handling three sources of measurement error: Doppler effect, multipath, and noise.

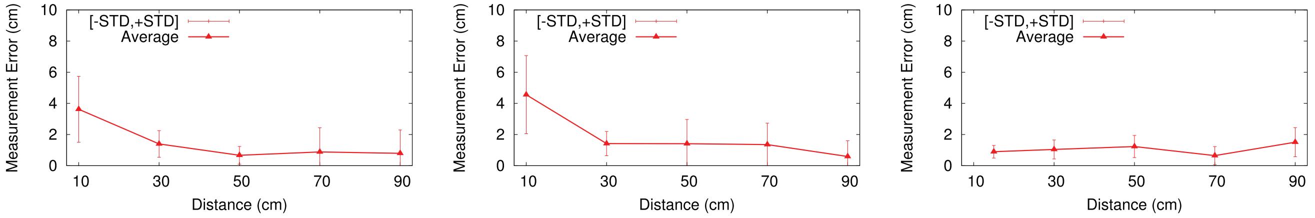
Robustness to Doppler Effect. FAR detects Doppler shifts with the use of parallel autocorrelators. Our experimental setup consisted of recording FAR and Kinect measurements simultaneously while instructing the players to swing hands at varying rates, including as fast as possible. We collected 15 motion traces, each consisting of over 400 distance measurements. We calculated a phone-to-phone relative velocity from the Kinect data (which is not affected by Doppler shift). We group motion traces according to the maximum velocity reported by Kinect.

Figs. 20a and 20b show the detection performance of parallel autocorrelators (without prediction and with prediction, respectively) for representative traces with motion



(a) Detection Ratio w/o Prediction (b) Detection Ratio w/ Prediction (c) Computation Time of Autocorrelators

Fig. 20. Parallel autocorrelators. (a) 90+ percent of the tones can be detected by five parallel autocorrelators, which corresponds to offset range $[L/2-2, L/2+2]$ (b) With predictive parallel autocorrelators, only three autocorelators are needed. 85+ percent of the tones can be detected even at 2.0-m/s hand movement speed. (c) Three parallel autocorrelators can still meet the computation deadline.



(a) Back To Back (b) Head To Back (c) Head To Head

Fig. 21. Accuracy in a small room. The impact of the reflection is not noticeable.

speeds of 0.5, 1, and 2 m/s, where the Doppler shift does occur. As expected, even one autocorrelator performs reasonably for 0.5 m/s but more autocorrelators are needed as the velocity increases. Most of the tones affected by Doppler shift can be recovered by five parallel autocorrelators as shown in Fig. 20a. This means the Doppler offset is usually within $[L/2-2, L/2+2]$.

Fig. 20b shows that after applying prediction, only three parallel autocorrelators can recover most of the errors, and achieve 85+ percent detection ratio even in a high-speed scenario of 2 m/s. The detection ratio gain for more than three autocorrelators is marginal. Fig. 20c shows the computation time required for increasing numbers of parallel autocorrelators. Three correlators can be run simultaneously while maintaining or just barely exceeding the computation deadline required for the fastest measurement frequency. Our empirical observations indicate that sustained velocities of 2 m/s or even 1 m/s are very hard for players to maintain due to physical human limits; there is a natural opportunity to detect and complete measurements after sudden bursts of fast movement. Therefore, FAR uses three parallel autocorrelators that work well in practice.

Robustness to Multipath and Ambient Noise. To test robustness to reflected sounds, we ran FAR in the smallest contained space we could find, a private room measuring 3 m by 2 m. Fig. 21 illustrates the performance in the small room with various phone orientations and distances. While the variance of measurement error is 5 cm and larger than in the original room, the measurement error is again very small, with a median error of 2 cm.

We injected an ambient sound to evaluate the antinoise property of FAR. To ensure repeatability, the ambient sound is a 1-minute recording from a very crowded area, with people talking and laughing loudly. We define the SNR of the signal as the energy of the sound recorded by the remote phone over the energy of the ambient sound. While we tested over different scenarios, we report only the performance of the back to back position result with $\mathrm{SNR}=0$ here in Fig. 22 due to the similarity of the results. As expected, the ambient sound degrades the accuracy, but the impact is manageable. Recall that $\mathrm{SNR}=0$ means a noise energy level equal to the tones', which is high.

9.5 Deployment Experiences

We have prepared SwordFight and ChaseCat for public play on multiple occasions: at the SenSys 2011 conference for any conference participant [26]; in Beijing for interns working at the Microsoft Research Asia building late at night; in Redmond for office colleagues taking a work break; and at UC Santa Barbara for officemates, friends, and family. Qualitatively, the public reception has been positive. Just as the Wii and Kinect ushered in a new class of console gameplay, SwordFight and ChaseCat introduce an element of physical action to mobile gaming that players seem to enjoy.

10 Related Work

Motion Gaming. Recently, nonmobile console gaming systems have embraced player motion-based gameplay, developing a variety of schemes for player localization. The Wii remote [16] uses an infrared camera to track the relative

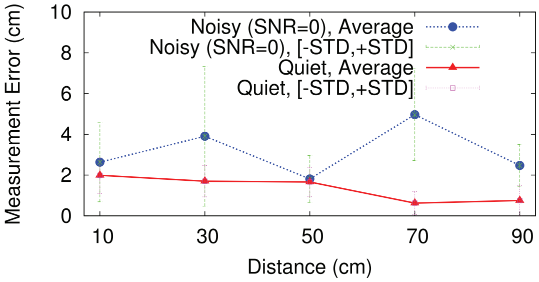


Fig. 22. Measurement accuracy when injecting heavy noise $(\mathrm{SNR}=0)$. The accuracy drops a little but can still achieve an accuracy of 5 cm in average.

position of the TV-mounted sensor bar, which emits two infrared light sources. The PS3 Move controller [24] has a light-emitting ball affixed, which is tracked by a TV-mounted video camera. The TV-mounted Kinect [15] projects an infrared mesh and tracks its time-of-flight back to the Kinect camera. These console gaming systems demonstrate the compelling nature of motion gaming, but are unfortunately tethered to fixed console-based infrastructure.

The idea of phone-to-phone motion gaming was first mentioned in the recent work of Qiu et al. [19], described in that paper's introduction as high-speed, locational, phone-to-phone (HLPP)-gaming. Qiu et al. [19] was the first to propose phone-to-phone localization as a primitive for mobile gaming. The authors developed a 3D acoustic localization system that utilizes each device's two microphones, one speaker, 3-axis accelerometers and 3-axis magnetometers to calculate coordinates of one phone in three dimensions relative to the other. However, the scheme proposed in [19] is not actually applicable to high-speed motion games such as SwordFight because the reported update interval is over 800 ms. Nevertheless, their 3D positioning algorithm may in the future serve as a complement to the high-speed ranging techniques presented in this work.

Acoustic Ranging. The authors of [17] developed an earlier method for acoustic ranging between two phones but its update rate can be arbitrarily slow because the two devices are not synchronized, and the applications are unclear.

Much research exists on infrastructure-based localization algorithms (see [11] for a survey). ActiveBat [5] and Cricket [18] are two representative infrastructure-based systems. Each is capable of achieving centimeter resolution, but dense ultrasonic infrastructure requirements are not compatible with ad hoc mobile gaming. Infrastructure-based acoustic localization includes [3], [13], [22]. These systems achieve resolution in meters, which is insufficient for action gaming. Several systems have reported achieving centimeter resolution [4], [7], [9] with custom-built hardware, which makes them less appealing in light of the pervasiveness of commodity mobile phones.

Components of our algorithm's signal processing techniques have been used previously in communication systems but not in the context of localization. Schmidl and Cox [21] proposed using autocorrelation to reduce the complexity of frame and carrier synchronization. Mason et al. [14] use a technique similar to running multiple parallel detectors for handling Doppler effects underwater, but does not consider predicting Doppler shifts. Acoustic signal processing techniques used in voice activity detection [23] should also complement our techniques for accurate and efficient detection of tones.

11 Discussion and Conclusion

In this paper, we report on the design and implementation of a new class of phone-to-phone games: MMG. The key underlying technology is a new highly accurate and real-time phone-to-phone distance measurement substrate, which provides the API on top of which games such as SwordFight and ChaseCat are built. Our experience with having people play these games is positive. They are intuitive and fun to play. However, our work here is a starting point. There remain open challenges that could affect the game experience. One issue is that continuous acoustic tones are noticeable, because mics and speakers on commodity phones that we used only support the audible frequency range. A possible approach is to embed these tones in game music. Another possible approach is to perform localization with mics and speakers in the ultrasonic range [12]. A second issue is that line-of-sight blockage between the phones degrades the measurement accuracy, and players can accidentally or purposefully block the mics and speakers. Future work includes creation of a blockage warning or cheat detection protocol. In addition, we are interested in extending the API to permit simultaneous ranging between more than two phones. Lastly, we are working on designing and prototyping additional MMG games on top of our phone-to-phone gaming API.

Acknowledgments

The authors extend their thanks to our shepherd Monica Lam and the anonymous reviewers for their feedback and suggestions, which greatly improved the paper. They would also like to thank Jian Qiu for consulting advice, and members of the UC Santa Barbara Link Lab for volunteering their time to help them collect in-situ game-play player data.

References

[1] F. Biscotti, B. Blau, J.-D. Lovelock, T.H. Nguyen, J. Erensen, S. Verma, and V.K. Liu, *Market Trends: Gaming Ecosystem,* Gartner, 2011.

[2] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003," *Proc. Third ACM SIGCOMM Workshop Network and System Support for Games (NetGames),* pp. 144-151, 2004.

[3] X. Bian, G. Abowd, and J. Rehg, "Using Sound Source Localization in a Home Environment," *Proc. Int'l Conf. Pervasive Computing,* pp. 281-291, 2005.

[4] L. Girod, M. Lukac, V. Trifa, and D. Estrin, "The Design and Implementation of a Self-Calibrating Distributed Acoustic Sensing Platform," *Proc. Fourth Int'l Conf. Embedded Networked Sensor Systems (SenSys),* pp. 71-84, 2006.

[5] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The Anatomy of a Context-Aware Application," *Wireless Networks,* vol. 8, no. 2, pp. 187-197, 2002.

[6] M. Hazas and A. Hopper, "Broadband Ultrasonic Location Systems for Improved Indoor Positioning," *IEEE Trans. Mobile Computing,* vol. 5, no. 5, pp. 536-547, May 2006.

[7] M. Hazas, C. Kray, H. Gellersen, H. Agbota, G. Kortuem, and A. Krohn, "A Relative Positioning System for Co-Located Mobile Devices," *Proc. ACM MobiSys,* pp. 177-190, 2005.

[8] L. Jones and S. Lederman, *Human Hand Function.* Oxford Univ., 2006.

[9] G. Kortuem, C. Kray, and H. Gellersen, "Sensing and Visualizing Spatial Relations of Mobile Devices," *Proc. 18th Ann. ACM Symp. User Interface Software and Technology (UIST),* pp. 93-102, 2005.

[10] M. Kushwaha, K. Molnár, J. Sallai, P. Volgyesi, M. Maróti, and A. Lédeczi, "Sensor Node Localization Using Mobile Acoustic Beacons," *Proc. IEEE Int'l Conf. Mobile Adhoc and Sensor Systems (MASS),* pp. 491-500, 2005.

[11] A. LaMarca and E. de Lara, "Location Systems: An Introduction to the Technology Behind Location," *Synthesis Lectures on Mobile and Pervasive Computing,* vol. 3, no. 1, pp. 1-122, 2008.

[12] P. Lazik and A. Rowe, "Indoor Pseudo-Ranging of Mobile Devices Using Ultrasonic Chirps," *Proc. 10th ACM Conf. Embedded Network Sensor Systems (SenSys),* pp. 99-112, 2012.

[13] C. Lopes, A. Haghighat, A. Mandal, T. Givargis, and P. Baldi, "Localization of Off-the-Shelf Mobile Devices Using Audible Sound: Architectures, Protocols and Performance Assessment," *ACM SIGMOBILE Mobile Computing and Comm. Rev.,* vol. 10, no. 2, pp. 38-50, 2006.

[14] S. Mason, C. Berger, S. Zhou, and P. Willett, "Detection, Synchronization, and Doppler Scale Estimation with Multicarrier Waveforms in Underwater Acoustic Communication," *IEEE J. Selected Areas in Comm.,* vol. 26, no. 9, pp. 1638-1649, Dec. 2008.

[15] "Xbox Kinect," *Microsoft,* http://www.xbox.com/kinect, 2013.

[16] "Nintendo Wii," *Nintendo,* http://www.nintendo.com/wii, 2013.

[17] C. Peng, G. Shen, Y. Zhang, Y. Li, and K. Tan, "BeepBeep: A High Accuracy Acoustic Ranging System Using Cots Mobile Devices," *Proc. Fifth Int'l Conf. Embedded Networked Sensor Systems (SenSys),* pp. 1-14, 2007.

[18] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The Cricket Location-Support System," *Proc. ACM MobiCom,* pp. 32-43, 2000.

[19] J. Qiu, D. Chu, X. Meng, and T. Moscibroda, "On the Feasibility of Real-Time Phone-to-Phone 3D Localization," *Proc. Ninth ACM Conf. Embedded Networked Sensor Systems (SenSys),* pp. 190-203, 2011.

[20] J. Sallai, G. Balogh, M. Maroti, A. Ledeczi, and B. Kusy, "Acoustic Ranging in Resource Constrained Sensor Networks," *Proc. Int'l Conf. Wireless Networks (ICWN),* pp. 467-474, 2004.

[21] T. Schmidl and D. Cox, "Robust Frequency and Timing Synchronization for OFDM," *IEEE Trans. Comm.,* vol. 45, no. 12, pp. 1613-1621, Dec. 1997.

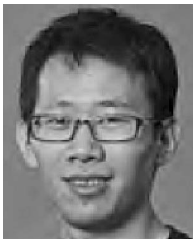
[22] J. Scott and B. Dragovic, "Audio Location: Accurate Low-Cost Location Sensing," *Proc. Int'l Conf. Pervasive Computing,* pp. 307-311, 2005.

[23] J. Sohn, S. Member, N.S. Kim, and W. Sung, "A Statistical Model-Based Voice Activity Detection," *IEEE Signal Processing Letters,* vol. 6, no. 1, pp. 1-3, Jan. 1999.

[24] "Playstation Move," *Sony,* http://us.playstation.com/ps3/playstation-move, 2013.

[25] K. Whitehouse and D. Culler, "Calibration as Parameter Estimation in Sensor Networks," *Proc. First ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA),* pp. 59-67, 2002.

[26] Z. Zhang, D. Chu, J. Qiu, and T. Moscibroda, "Demo: Sword Fight with Smartphones," *Proc. Ninth ACM Conf. Embedded Networked Sensor Systems (SenSys),* pp. 403-404, 2011.



Zengbin Zhang is working toward the PhD degree at the University of California, Santa Barbara.

David Chu is a researcher at Microsoft Research Redmond.

Xiaomeng Chen is working toward the PhD degree at Purdue University.

Thomas Moscibroda is a senior researcher at Microsoft Research Asia and a chair professor at Tsinghua University.