UbiTouch: Ubiquitous Smartphone TouchPads using Built-in Proximity and Ambient Light Sensors

Elliott Wen, Winston Seah, Bryan Ng
School of Engineering and Computer Science,
Victoria University of Wellington
jiaqi.wen, winston.seah, bryan.ng@ecs.vuw.ac.nz

Xuefeng Liu, Jiannong Cao
Department of Computing,
The Hong Kong Polytechnic Univerisity
csxfliu, csjcao@comp.polyu.edu.hk

ABSTRACT

Smart devices are increasingly shrinking in size, which results in new challenges for user-mobile interaction through minuscule touchscreens. Existing works to explore alternative interaction technologies mainly rely on external devices which degrade portability. In this paper, we propose UbiTouch, a novel system that extends smartphones with virtual touchpads on desktops using built-in smartphone sensors. It senses a user's finger movement with a proximity and ambient light sensor whose raw sensory data from underlying hardware are strongly dependent on the finger's locations. UbiTouch maps the raw data into the finger's positions by utilizing Curvilinear Component Analysis and improve tracking accuracy via a particle filter. We have evaluate our system in three scenarios with different lighting conditions by five users. The results show that UbiTouch achieves centimetre-level localization accuracy and poses no significant impact on the battery life. We envisage that UbiTouch could support applications such as text-writing and drawing.

INTRODUCTION

Smart devices provide various interaction interfaces, among which, touchscreens are the most prevalent one. Generally, a larger touch screen leads to better user experiences. Nevertheless, numerous smart devices are shrinking in sizes to improve portability, which results in new challenges for user-mobile interaction through minuscule touchscreens. Particularly, it renders handwriting input or drawing on those smart devices cumbersome. This difficulty motivated various researchers to explore alternative interaction technologies. Notable systems, e.g. RF-IDraw [24], TypingRing [17], Okuli [30], and OmniTouch [9], have been able to achieve very good tracking accuracy. However, they rely heavily on peripheral devices such as extra RFID tags and cameras, which may significantly limit the portability of smart devices.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UbiComp '16, September 12 - 16, 2016, Heidelberg, Germany

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4461-6/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2971648.2971678

To address the portability issue, researchers have attempted to take advantage of built-in sensors to achieve the same purpose. For example, Finger-in-Air [16] utilizes a smartphone camera to estimate users' finger gestures. UbiK [25] leverages microphones on a smartphone to detect keystroke locations, enabling text-input on a sheet of paper where a keyboard outline is printed. UbiK [25] is solely designed for distinguishing different keys a user presses. It may not support applications that require continuous finger tracking, for instance, handwriting input or drawing. Despite the promising results obtained, vision-based systems [16] require the camera module to be continuously switched on. This could drain the battery quickly and its performance may be negatively impacted by the low-light environment.

Nevertheless, these successful applications shed light on our research and we pose the following question: "Is it possible to utilize built-in low-power sensors to constantly localize and track a user's finger, enabling touchpad-like input experience?". We answer this question by introducing UbiTouch, a novel system that achieves the goal of augmenting smartphones with virtual touchpads by leveraging built-in smartphone sensors. Specifically, the smartphone's proximity sensor (PS) and ambient light sensor (ALS) are utilized to sense the movement. Meanwhile, the microphone and gyroscope sensors are used to detect touch actions such as tapping and dragging. These sensors are readily available in almost every smart device and of low-power consumption.

The key challenge for UbiTouch lies in achieving fine-grained localization and tracking using sensors that are originally designed to provide coarse-grained information. An ALS is typically used to measure the luminance of ambient lighting [13]. Readings from this sensor are not directly related to a finger's movement. Similarly, a PS is designed to detect the presence of nearby objects and merely reports binary distance values representing "near" or "far" to smartphone operating systems [7], which does not benefits the tracking. In this paper, we take a closer examination of the internal structures and underlying mechanisms of the sensors [2] and uncover the potential for these sensors to be used to achieve our goal.

Specifically, we implement UbiTouch as a system with three key components: finger tracking, touch action de-

tection, and run-time calibration. We introduce a finger localization framework that adapts Curvilinear Component Analysis for mapping the raw sensor data to two-dimensional coordinates. It then estimates the finger movement trajectories by utilizing a particle filter which integrates the historic finger locations to boost the tracking accuracy. We also propose a touch action detection algorithm to capture touch tapping and dragging events on ordinary surfaces by applying hypothesis testing techniques on audio signals from microphones. To provide reliable results, we leverage the built-in gyroscope to combat the environment noises. Additionally, we design a runtime calibration mechanism that takes advantage of extra run-time user feedback data to re-calibrate the tracking algorithm and combat minor variation of background light conditions.

We consolidate the above techniques and implement a prototype system in an Android smartphone (LG Nexus 5). We then carry out experiments in three scenarios corresponding to different lighting conditions and by five individuals. The results show that in general conditions, UbiTouch can detect and localize finger positions with a mean localization error of 1.53 cm and a standard deviation of 0.42 cm. When UbiTouch works in conjunction with a hand-writing recognition application, it achieves a character recognition rate of 79%, comparable to a physical touchpad. Another advantage of this system is that it poses no significant impact on battery life of the smart phone.

The main contribution of this paper is to demonstrate the potential of built-in ambient light and proximity sensors to extend a smartphone touchscreen with virtual touch input. The principles and methods explored in this paper offer an alternative solution for interacting with smart devices.

RELATED WORK

In this section, we review past work on exploring new interaction technologies and position our contribution of providing a virtual touchpad without any extra sensors.

Extending smartphones' input space

One problem with touchscreens is that the users' fingers occupy valuable input space. Moreover, this problem aggravates when devices are shrinking in size. To tackle this issue, a great number of research work shift the interaction away from the touchscreen to nearby areas by augmenting smartphones with extra hardware such as keyboards [11], touchpads [4], cameras [9], and other specially-designed sensors [29, 18]. Despite the remarkable achievement, these system inevitably degrades devices' portability. Considering that smartphone platforms include increasingly sophisticated sensors, researchers start investigating the potential input approaches supported by the built-in smartphone sensing capabilities.

Built-in sensor based input approaches

A great number of research works utilize the built-in smartphone sensors such as inertial, compass, microphone, and camera sensors to provide alternative input approaches. Research work [10] leverages inertial sensors and touch input to provide motion-enhanced touch gestures. MagiTact [12] utilizes built-in compass sensor to extend interaction space of small mobile devices. Systems [20] detect users' tap events by analyzing the audio signals captured by microphones. LucidTouch [26] enables users to control the applications by touching the back of the device using computer vision techniques. More recently, wireless technologies have drawn a great amount attention of researchers. Some of them managed to extend smartphones input by utilizing wireless signals such as WiFi and RFID. For instance, Wigest [1] enables users' gesture input to smart devices by sensing the WiFi signal strength.

Virtual touch surface

Among various interaction technologies, touching is still most straightforward and natural approach for smart devices. A number of researchers managed to augment smart devices with virtual touch surfaces. For instance, RF-IDraw [24] relies on a RFID ring to infer the finger movement, allowing drawing in the air. TypingRing [17] leverages a ring equipped with motion sensors to trail a finger, enabling handwriting text-input experience. Okuli [30] augments smartphones with a LED light array to capture the finger movement, serving as a virtual trackpad for the user. Besides, Canesta [21] and OmniTouch [9] achieve the same goal by utilizing external cameras and image process techniques.

Despite of the promising results achieved, these systems inherit some limitations. Systems that rely on the extra devices may severely degrade the portability of the smart devices. More importantly, they also incur extra costs for the users. Recently, some researchers attempt to achieve finger tracking by purely leveraging the smartphone built-in sensors. For instance, Finger-in-Air [16] leverages smartphone cameras to detect the movement of a finger. UbiK [25] takes advantage of microphones on a smartphone to detect keystroke locations, serving as a virtual keyboard. However, the vision-based system [16] continuously turning on cameras may drain the battery swiftly and the performance may be negatively affected by low-light environment. As for the UbiK [25], it is solely built to distinguish among different keys a user presses, which may not be able to support applications such as handwriting text-input and drawing that require continuous finger movement tracking.

To address these issues, we propose a novel system UbiTouch which uncovers the potential of the built-in smartphone sensors to extend smartphones with virtual touchpads. Unlike existing works, it achieves continuous finger movement tracking while does not rely on any external devices or cameras.

SYSTEM OVERVIEW

UbiTouch extends touch-screen based devices with an external virtual touchpad on a common surface (e.g., wooden desktop). Similar to a conventional touchpad, it accepts general touch actions including tapping and dragging.

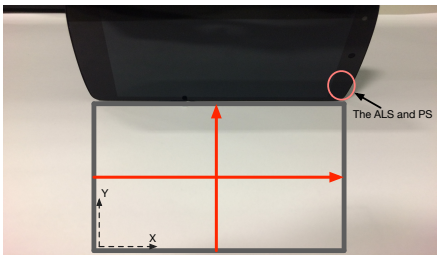

Figure 1: Typical system deployment.

Fig. 1 demonstrates the typical use case of UbiTouch. For the purpose of illustration, a 7 cm $\times$ 13 cm rectangle zone is marked on a desktop, serving as a virtual touch area. An off-the-shelf Android smartphone (LG Nexus 5) is then placed vertically on the top border of the zone, enabling the ALS and PS to sense the movement inside the virtual touch area. Before running UbiTouch, an initial training is needed, whereby the user moves his finger along some trajectories instructed by the system to generate some training data. UbiTouch then learns a mapping function from the training data through a neural network. It then uses the function to convert the subsequent sensor readings into two-dimensional coordinates, enabling continuous tracking.

In this research, our design goal is for UbiTouch to work in a portable, accurate, and robust manner. UbiTouch should only rely on smartphone built-in sensors to achieve centimetre-scale finger tracking and touch action detection. Meanwhile, UbiTouch should be resilient against minor changes of the ambient light. To achieve these goals, UbiTouch adopts a work-flow shown in Fig. 2. It can be seen that the architecture of UbiTouch comprises the following three major components: (i) finger tracking, (ii) touch action detection, and (iii) run-time calibration and adaptation.

A PEEK AT TRAILING A FINGER USING ALS AND PS

In this section, we first provide some rationales about how raw readings from ALS and PS sensors could be utilized to track a user's finger. Then we conduct some preliminary experiments to determine the utility of the ALS and PS information.

Figure. 3(b) demonstrates the underlying mechanisms of the ALS. It can be seen that the ambient light sensor consists of two photodiodes (CH0 and CH1) in two separate locations. The first photodiode is sensitive to both visible and infrared light while the second photodiode is primarily sensitive to infrared light. The fusion of the two readings is used to estimated luminance of the visible light. Assuming the settings of background light sources (i.e., positions and brightness) remain unchanged, movement of an object or finger may alter the path in which the light propagates. It then affects the intensity of the light received by the two photodiodes. Therefore, abundant information can be derived on how an object moves from the light intensity readings.

Meanwhile, the proximity sensor depicted in Fig. 3(c) also provides extra movement information. A LED emitter broadcasts a certain number of infrared pulses. The pulses strike a nearby object (which is a user's finger in our scenario) and get reflected to a receiver. The receiver counts the number of pulses and estimates the distance based on a simple principle: the farther the object is, the fewer the number pulses received, since the infrared light has to travel in a longer path and loses more energy. It is easy to see that variation of the pulse number has a strong link with the object movement, which may assist in our goal.

We start with some simple tests to determine the utility of the ALS and PS information. We first setup the experiment environment which complies with the UbiTouch's typical use case shown in Fig 1. All experiments are conducted in an ordinary office environment where the settings of background light sources (i.e., positions and luminance) do not vary.

For the purpose of illustration, we specify a coordinate system for the area where the origin is located at the lower-left corner and two axes are marked by the dash lines in the figure. To understand how the movement of a finger affects the readings of the ALS and PS, we instruct a user to move his finger along the trajectories in red. The trajectories are designed to be either parallel to the axis 'X' or axis 'Y' such that we can better observe the variation trend by altering one variable each time.

During the movement, we record five types of readings: (1) luminance (LUX); (2) distance measurement (a binary value representing 'far' or 'near'); (3) raw sensor reading from the photodiode CH0; (4) raw sensor reading from the photodiode CH1; (5) count of pulses received by the PS. The first two kinds of data can be easily retrieved via the sensor API provided by the Android platform. However, the system blocks the access to the remaining three types of data. To overcome this constraint, a patch is applied on the OS kernel, which enables us to bypass the Android framework and access the raw readings from the hardware directly. Specifically, the patch first replaces the original proprietary PS and ALS driver with an open-source one [3]. The new driver is designed to stream the data of registers in the hardware to user-space applications through SYSFS interfaces [15]. The UbiTouch application continuously polls the samples from the driver and process them. In our implementation, we set the polling frequency for the PS and ALS to 100Hz, which is the maximum frequency the hardware can support [2].

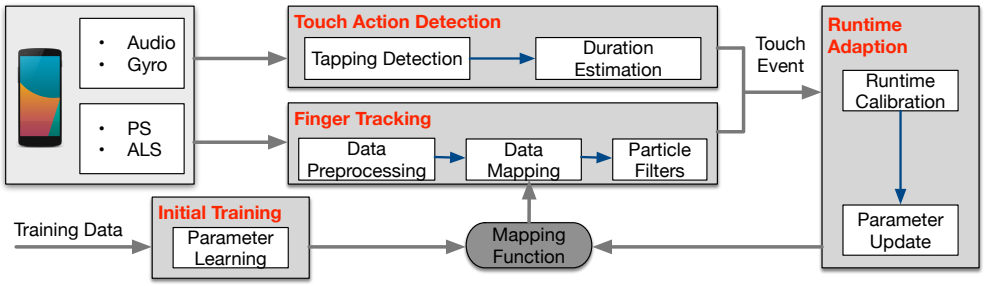

Figure 2: System Logic Flow for UbiTouch

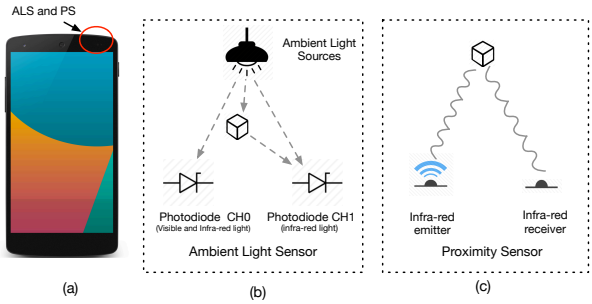

Figure 3: (a) shows the positions of the PS and ALS on a smart phone. (b) and (c) show the underlying mechanisms of the ALS and PS respectively.

We display these readings for the two trajectories in Fig. 4(a) and Fig. 4(b) respectively. From top to bottom, the sub-figures demonstrate how the luminance, binary distance measurements, CH0 readings, CH1 readings, and pulse counts change over the finger's positions. It can be seen that, regardless of the moving directions, there is no obvious link between the movement and the luminance. We analyze the possible causes and find that the resolution for the noisy luminance readings is somewhat low because the values are integers and only vary between 20 to 26 in our experiments. It renders the luminance an unsuitable feature for our goal.

As for the distance measurements, the binary values will turn to one when the finger moves within a certain radius (around 5cm) of the PS. Nevertheless the binary information is still coarse-grained to realize accurate localization and tracking.

We now turn our attention to the three remaining types of data. The first observation we can easily identify is that the readings for CH1 and CH0 have a strong correlation with the finger's movement. Specifically, as the finger moves closer to the ALS, the readings decrease. Another important observation is that the pulse counts also have a strong link with the movement. When a finger is getting further away from the PS, the pulse number received significantly decreases.

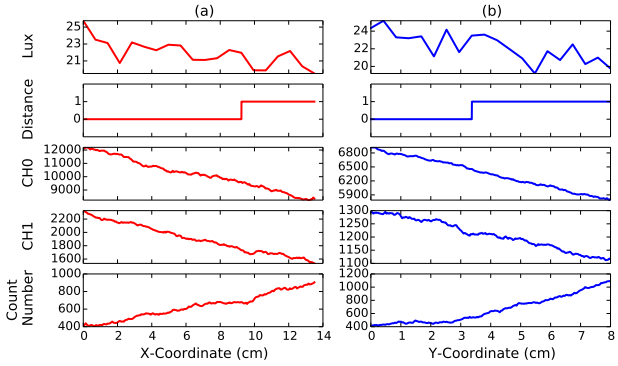

Figure 4: Recorded sensor readings during the movement along the aforementioned trajectories. The readings have been interpolated and outliers have been filtered out.

It can be seen that these data are location-dependent and have the potential to enable localization and tracking. We explore the practical solutions to obtain finger locations in the next section.

OBTAINING FINGER LOCATIONS FROM RAW SENSOR DATA

Let tuple $s_t : (r_t^0, r_t^1, c_t)$ denote the readings for the photodiode CH0, photodiode CH1, and pulse count at time t. Let $l_t : (x_t, y_t)$ denote the finger position, which is the coordinate on the aforementioned virtual touch area. To locate the finger from the raw sensor data, we use a mapping function f that maps the s_t into l_t.

Ideally, a well-chosen deterministic mapping model that takes the settings of the background light sources and placement of the smartphone into consideration will achieve best performance. However, as the background environment and smartphone deployment vary from time to time, it would be fairly difficult and laborious for users to decide the numerous parameters of the model when they are using the system. Thus, we seek a model that automatically tunes its parameters and adapts to different environments.

In this paper, we adopt a nonlinear mapping model called Curvilinear Component Analysis (CCA) [6]. The CCA is a self-organizing neural network which has the ability to learn the mapping between a high-dimensional space to a low-dimensional one. Mathematically speaking, CCA attempts to train a network (i.e. a mapping function) to optimize the following criterion that explicitly measures the preservation of the pairwise distances (denoted by E):

$$E = \sum_{j=1}^{N}\sum_{i=1}^{N}(\delta(s_i, s_j) - d(l_i, l_j))^2 F(d_{l_i l_j}, \lambda) \qquad (1)$$

where N denotes the total number of the training data. The $\delta(s_i, s_j)$ is the curvilinear distance in the input space and $d(l_i, l_j)$ is the Euclidean distance in the output space between the i-th and j-th data points. The factor F weighs the contribution of each pair in the criterion, which is usually implemented as the Heaviside step function:

$$F(d(l_i, l_j), \lambda) = \begin{cases} 0, & \text{if} \quad d(l_i, l_j) - \lambda < 0 \\ 1, & \text{if} \quad d(l_i, l_j) - \lambda \geq 0 \end{cases} \qquad (2)$$

where λ is a neighboring radius and set slightly larger than the maximum curvilinear distance measured in the input data set (i.e., $\max(\delta(s_i, s_j)|i < j \leq n)$).

The rationale behind this method is that if two readings are quite similar to each other, the points they are mapped into should also have a small distance. Conversely, a huge difference between sensor readings indicates a large distance between the points. Thus, this method attempts to obtain a mapping function f by minimizing the difference between the pairwise distances in the input space and output space.

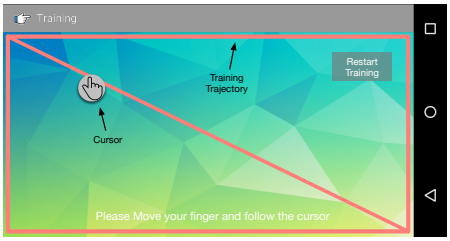

Figure 5: The application for gathering training data.

To apply this approach, we collect some training data as a series of s_t and the corresponding l_t. Thus, a data gathering application shown in Fig. 5 is built to facilitate this process. The application first requests the user to input the size of the touch area and displays a trajectory along which a cursor gradually moves in a certain speed. The user is then requested to move his finger inside the touching area on the desktop, imitating the movement indicated by the cursor.

The user may repeat the process for several times in order to achieve a good synchronization between the finger movement and the cursor motion. In this manner, the coordinates of the cursor on the smartphone screen could be transformed to the coordinates of the finger on the touch area by a simple scaling transformation. Compared with single point calibration, our calibration process can obtain far more training data. The training process typically lasts one minute.

Before feeding the training data to the neural network, we notice that the training data contain many outliers and heavy noises. Three preprocessing tasks are used to clean the training data.

Outlier Removal. We find that there are some abrupt changes of the data that are obviously not caused by the finger movement. Fig. 6(a) shows an example of the training data and it can be seen that there are some significant abrupt change in the l_0 and l_1 around the positions 2, 3 and 6.5. There are likely to be outliers and should be eliminated.

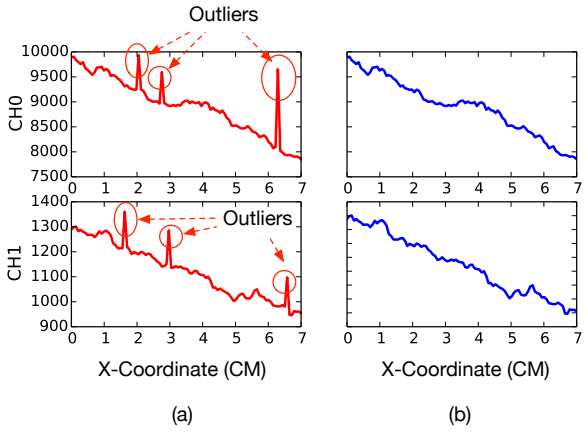

Figure 6: (a) The original CH0/CH1 data. (b) The CH0/CH1 data after the outliers are removed using the Hampel filter.

To achieve that purpose, we first attempt the widely-used Standard Deviation (SD) method. However, this approach highly relies on the mean and standard deviation which are extremely sensitive to the presence of outliers, hence does not perform well. Therefore, we utilize a more sophisticated outlier-removal algorithm called the Hampel Identifier [14]. The algorithm declares any data points not within the range $[\mu - \gamma * \sigma, \mu + \gamma * \sigma]$ as outliers, where u and σ denote the mean and the median absolute deviation of the sequence respectively, while γ is a constant and the typical value is 3 for general cases. Fig. 6(b) demonstrates the results for outlier removal. It can be seen that the Hampel Identifier eliminates the outliers and preserves the desire CH0 and CH1 responses.

Interpolation. We program the application to poll the sensor data from the hardware every 10ms (i.e., 100Hz). However, we cannot guarantee that a reading is retrieved at the desired frequency, because the Linux kernel used by typical smartphones does not provide real-time task

scheduling. A detailed look at the data shows that sampling jitters appear frequently and the sampling latency can be up to 150 ms when the smartphone is having a high computation load. Therefore, the data must be interpolated before further processing. Here, we apply linear interpolation at the data to retrieve a re-sampled series with the desired sampling frequency of 100Hz.

Noise Filtering. The final step of pre-processing is to suppress the heavy noises in the data series. In our implementation, we utilize the wavelet filter proposed in [23] to smooth away the high-frequency noises. Specifically, we apply 4-level 'db4' wavelet transform on sensor data and only use the approximation coefficients to 'reconstruct' the filtered signal. Fig 7 demonstrates the data sequences before and after filtering. It can be seen that the noisy signal is getting much cleaner.

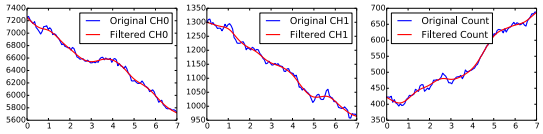

Figure 7: The effect of noise filtering.

The filtered data is fed to the CCA neural network. In the training phase, a tunable parameter called learning rate (denoted by $\alpha \in [0, 1]$) decides the convergence speed of the learning process. Here, we adopt a linear learning rate function which linearly decreases the rate with time.

Improving Tracking Accuracy via a Hidden Markov Model

In the previous section, we attempt to obtain the finger locations from the sensor readings by utilizing a CCA network. It simply maps the sensor reading s_t into a location l_t, without taking previous readings into consideration.

This method does not take advantage of an important observation: a user typically moves his finger smoothly, thus the finger position at a consecutive moment is likely not far away from the current location. The observation indicates that the finger movement can be modeled by a Hidden Markov Model (HMM) shown in Fig. 8. The model contains a set of hidden state variables (i.e., finger positions) and observable variables (i.e., sensor measurements). The current position l_t depends on the previous location l_{t-1} according to the probabilistic transition model $p(l_t|l_{t-1})$. The sensor measurement s_t can be regarded as a stochastic projection of the hidden state l_t generated via the probabilistic observation model $p(s_t|l_t)$. We now want to sequentially estimate the values of the hidden location l_k, given the values of the observation process $s_0, \cdots, s_k$, at any time step k, which follows the posterior density:

$$p(l_k|s_0, s_k, ..., s_k). \tag{3}$$

It can be seen that this model enables us to further improve the tracking accuracy by incorporating the historic sensor measurements.

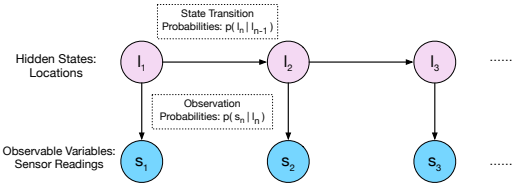

Figure 8: HMM-based Movement Modeling.

To estimate the hidden states, we adopt an approach called Particle Filter [19] which can efficiently deal with a large number of possible states in our scenarios. The algorithm is described in three steps:

1. Initialization:
   - For $i = 1, ..., M$, randomly initialize l_0^i, and set $t = 1$.
2. Importance sampling step:
   - For $i = 1, ..., M$, sample $\hat{l}_t^i \sim p(l_t|l_{t-1}^i)$.
   - For $i = 1, ..., M$, evaluate the importance weights by $\hat{w}_t^i = p(s_t|\hat{l}_t^i)$.
   - Normalize the importance weights.
3. Selection step:
   - Re-sample with replacement M particles $(l_t^i; i = 1, ..., M)$ from the $(\hat{l}_t^i; i = 1, ..., M)$ according to the importance weights.
   - Set $t \leftarrow t + 1$ and return to the step 2.

This algorithm requires two probabilistic models: the transition model $p(l_t|l_{t-1})$ and the observation model $= p(s_t|l_t)$.

We first try to compute the transition model $p(l_t|l_{t-1})$. Based on the aforementioned observation, we can safely assume the user moves his finger in a reasonable speed not exceeding a constant value V_{max}. We now express the transition model as follow:

$$p(l_t|l_{t-1}) = \begin{cases} 0, & \text{if} \quad d(l_t, l_{t-1}) - V_{max} * T < 0 \\ \frac{1}{2\pi V_{max}^2 T^2}, & \text{if} \quad d(l_t, l_{t-1}) - V_{max} * T \geq 0, \end{cases} \tag{4}$$

where T is the sampling period (10ms). V_{max} here is set to $0.05m/s$ based on the previous finger touch research [5]. The intuition behind the transition model is that a finger located at l_{t-1} can uniformly move to any points within the circle whose center and radius are l_{t-1} and $V_{max} * T$ respectively.

Next we have to figure out the observation model $p(s_t|l_t)$. Noting that we have obtained a mapping function f that

maps a sensor measurement s_t to a position l_t in the previous section, we could obtain the following equation:

$$s_t = f^{-1}(l_t) + W, \tag{5}$$

where f^{-1} is the reverse mapping function that projects l_t to s_t and can be obtained from the CCA network. W denotes the measurement noise and has a Gaussian distribution, that is $W \sim \mathcal{N}(0, \Sigma)$ and $\Sigma = \text{diag}(\sigma^2)$. The σ is a standard deviation and set to 10 which is measured from the real sensory data.

Therefore, the measurement model can be expressed as:

$$p(s_t|l_t) = C * \exp\left((s_t - f^{-1}(l_t))^T \Sigma^{-1} (s_t - f^{-1}(l_t))\right), \tag{6}$$

where C is a constant and does not affect the final results due to normalization in the importance sampling step.

It should be noted that, the algorithm's time complexity is linear with respect to the number of samples M. Obviously, the more samples, the better the accuracy, so there is a trade-off between speed and accuracy. To find an optimal value of M for the system, we adopt a strategy described in [22] which dynamically adjusts the number of samples and is adaptive to the available computational resources.

TOUCH ACTION DETECTION

The system is now able to accurately track the finger movement. However, to mimic a virtual touchpad, the system requires to detect two typical touch actions including tapping and dragging.

Our system realizes the detection by leveraging a smartphone built-in microphone. Tapping on a desktop typically generates audible stroke sounds. Similarly, dragging gestures request the user to rub on a desktop, producing constant audio noises as well. These signals could be received by the sensor and facilitate the touch action detection.

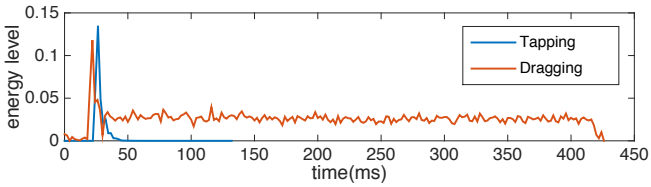

Figure 9: Energy level of the audio signals for tapping and dragging. It is obtained via short-team average on the audio signal with a moving window.

Fig 9 shows the energy level of the audio signals received when a user is tapping and dragging on a desktop. The audio energy generated by a tapping has a pattern similar to an impulse response. It starts with a outstanding peak in the beginning and fades away in a short period. Regarding the dragging action, it can be divided to two sub-actions: initial tapping and subsequent rubbing. It incurs a peak at the very beginning and causes a long lasting disturbance to the audio signals. It can be seen that the first step for our detection algorithm is to identify the tapping event.

In our implementation, we utilize a hypothesis testing approach called the generalized likelihood ratio test (GLRT) [27] to determine whether there is a tapping event. We store the incoming audio energy readings for 100ms in a First-In-First-Out buffer and keep testing the following two hypotheses on the data. The null hypothesis H_0 is that there is no tapping on the desktop and hence the average of the energy level should remain stable. Thus, we have

$$H_0 : u(t) = \mu_0 + w(t) \tag{7}$$

The variable $u(t)$ is the mean of the energy level before the time t. The μ_0 denotes the average energy and the variable $w(t)$ is the Gaussian noise with zero mean value and unknown variance σ_0.

The alternative hypothesis H_1 is that there is a tapping such that the mean of the energy level will significantly increase due to the tapping impulse. Thus, we have

$$H_1 = \begin{cases} u(t) = \mu_0 + w(t) & \text{if } t < t_c \\ u(t) = \mu_1 + w(t) & \text{if } t \geq t_c \text{ and } \mu_1 \neq \mu_0, \end{cases} \tag{8}$$

where the t_c represents the time when the tapping occurs, μ_1 denotes the new average energy after the tapping. Note that u_0, σ_0, μ_1, and t_c are unknown parameters.

Let θ_0 denote $\{\mu_0, \sigma_0\}$, and θ_1 denote $\{\mu_0, \sigma_0, \mu_1, t_c\}$. GLRT tests the two hypotheses by

$$\frac{\max_{\theta_1} p(u \mid H_1, \theta_1)}{\max_{\theta_0} p(u \mid H_0, \theta_0)} \underset{H_1}{\overset{H_0}{\lessgtr}} \gamma, \tag{9}$$

where $p(u \mid H_1, \theta_1)$ is the probability density function of u under hypothesis H_1 and θ_1. The $p(u \mid H_0, \theta_0)$ is defined in a similar manner. The r is the threshold and set to 100 which works well in general cases.

However, the energy based detection system is likely to be disturbed by ambient noises. According to the field test, human voices or sudden random burst of sound nearby can frequently mis-trigger the detection system. To remove the false positives, we utilize a cross-checking approach that leverages a smartphone's gyroscope.

The algorithm takes advantage of an observation that a tapping on a desktop generates vibrations which could be easily captured by the gyroscope, while the ambient noises do not. Thus, whenever a tapping event is detected by the audio signal, we further check the presence of the vibrations in the gyroscope signals. To detect the vibrations, we adopt the GLRT algorithm in a similar manner to the audio signal. This approach significantly suppresses the false alarms caused by the ambient noises.

After a tapping event is detected, we further estimate the duration of the touch action in case there is a following dragging event. The duration is estimated between the point when the tapping occurs and the point when the

power drops to below the threshold $\mu_0 + (\mu_{max} - \mu_0) * 10\%$. The μ_{max} is the maximum power level that the tapping impulse can reach. Capping the duration at the power level slightly above the normal noise energy helps the system combat the fluctuation of the noise energy floor.

It should be noted that the user is suggested to perform the touch actions gently, since a large movement of the finger may have negative impact on the finger tracking accuracy. In practice, the detection algorithm can achieve great performance when the user gently hits or rubs the desk using his nail tip.

RUNTIME CALIBRATION AND ADAPTATION

We assume that the smartphone is placed at an environment where the settings of ambient light including positions and luminance remain unchanged. However, in practice, the lighting conditions may vary slightly due to various reasons (e.g., a bulb may slightly decrease its brightness due to insufficient voltage), rendering the mapping function retrieved from the initial training outdated. In our system, rather than simply restarting the whole training phrase, we adopt a run-time calibration mechanism that enables the system to adapt to the minor change of the environment with a tiny effort. In some rare situations when the lighting conditions change drastically, the user can still manually restart the entire training phrase.

The system executes runtime adaptation by allowing the user to provide extra feedback on the localization results. When the user notice minor localization error, he could pause the tracking algorithm for a short period and manually input the correct coordinate by clicking on the right position on the touchscreen. Through this run-time calibration process, the system can harness extra training samples to adapt to the slightly varying environment. Note that we should carefully adjust the learning rate α associated with the new sample. A tiny α may render the update useless, while a huge α may amplify the noise in the sample and make the mapping function unstable.

In our implementation, we design the α based on the localization error. Specifically, if the original mapping function $\hat{f}$ maps the s_n into a coordinate $\hat{l_n}$, which has a relatively small distance to the coordinate l_n provided by the user, it means the function performs well and the update rate α should remain low. Conversely, a high update rate is selected when the distance is large. Based on the principle, we set α as:

$$\alpha_t = L(||l_t - \hat{f}(s_t)||^2) \tag{10}$$

L is a shifted logistic function that can be expressed as:

$$L(x) = \frac{1}{1 + \exp(6 - 3x)} \tag{11}$$

The logistic function converts the localization error into a value between (0,1) and holds nice properties. First, when the localization error is pretty small (less then 0.5cm), the value remain relatively low. Second, as the localization error continuously increases, the alpha rate will increase significantly. Finally, if the localization error goes beyond a certain value (3cm), the function returns a α value close to 1.

SYSTEM EVALUATIONS

In this section, we provide detailed performance evaluation of UbiTouch in terms of finger tracking accuracy, touch gesture detection accuracy, and system energy efficiency. We conducted our experiments in three common scenarios with dim, normal, and strong lighting condition by five users. Then we showcase UbiTouch's effectiveness when it works in conjunction with a handwriting recognition system.

Accuracy of Finger Tracking

There are several underlying factors that may affect the tracking performance of UbiTouch. In this section, we first carried out a baseline test in a typical usage scenario. Then we consider the effect of two crucial factors including (1) ambient lighting condition and (2) the tilted angle of the smartphone.

Accuracy for baseline experiment

In the baseline experiment, instead of running UbiTouch on an ordinary desktop, we deploy the system on top of a Macbook Pro's physical touchpad whose size is 7.5 cm × 10.5 cm. It collects the finger locations and trajectories which serve as the ground truth.

We then carried out a total of 100 static finger localization tests in two different setup: (1) tracking merely with the CCA mapping function. (2) tracking with both the mapping function and particle filter.

With only the mapping function, the average localization error is 1.69 cm with a standard deviation of 0.53 cm. The CDF shown in Fig 10 reveals that over 80% points have an error of less than 2.1 cm. When the mapping function works in conjunction with the particle filter, the mean error reduces to 1.53 cm with a 0.42 cm standard deviation. The corresponding CDF demonstrates that 80% of the error values is now less than 1.8 cm. The performance improvement is attributed to the particle filter which takes extra historic information into consideration.

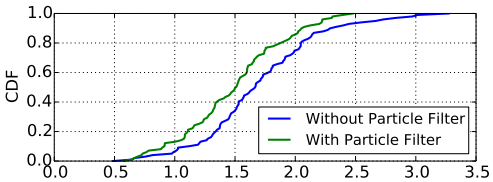

Figure 10: CDF for the localization error (cm).

Besides, the performance can also be observed through the spatial error distributions shown in Fig 11. An obvious observation is that the remote spots far away from

the PS and ALS sensor tend to possess larger error than the area around the sensors does. We investigate this phenomenon and discover that when the finger is not within the optimal detection range of the PS (10cm), the sensor readings are saturated with noises, which might severely degrade the localization performance. Noticing that the transmission power of the PS is adjustable, we may be able to increase the detection range by consuming more energy.

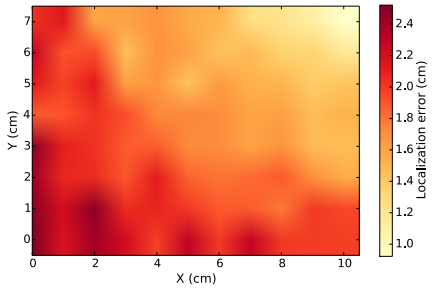

Figure 11: The spatial error distribution (interpolated). The position for the ALS and PS is around (12, 7.5).

Impact of ambient lighting condition

To measure the impact of the ambient lighting condition, we deploy our system in three typical scenarios including a storage room with a dim light bulb, an office room at campus with moderate light level, and an outdoor place with excessive sun light. Note that the settings (e.g., positions and illumination) of background light sources for these scenarios remain unchanged during our experiment. The experiments run on each scenarios, each repeated 100 times. Each time we conduct an experiment, we re-calibrate the system to achieve accurate performance measurements. Fig. 12 plots the corresponding localization accuracy. The results show that UbiTouch can maintain a localization error less than 1.54 cm across all the scenarios. It indicates that the system can perform well regardless of the illumination of the ambient lighting.

Tilted Angle

In the previous experiments, the smartphone is deployed vertically on a desk. However, in practice, some users prefer to place their phones in a slightly tilted angle for a better user experience or perspective. Hence, we carried out multiple tests under different inclined angles, which can be directly measured by the built-in orientation sensors.

Fig. 13 demonstrates how the tracking accuracy changes with the titled angle. It can be seen that the localization error mildly increases when the titled angle starts rising. It indicates that the system still performs well when the smartphone is deployed at a slightly tilted angle (less than 10 degrees). However, the slope drastically grows starting from 25 degrees and the localization error soars into xcm when the angle reaches 60 degrees.

	Library	Office	Pub
Error rate with gyro (FP/FN)	0.5% (0/0.5)	0.8% (0/0.8)	3.5% (0.3/3.2)
Error rate w/o gyro (FP/FN)	0.7% (0.3/0.4)	1.3% (0.5/0.8)	21% (15/6)

Table 1: Accuracy of tapping detection.

	Library	Office	Pub
Est. Duration(sec)	9.7	10.2	10.5

Table 2: Accuracy of touch-event duration estimation.

It may be due to that when the smartphone is significantly inclined, the AS and PLS turn into a direction far away from the finger, which has negative impact on the localization accuracy.

Accuracy of Touch Action Detection

We now evaluate the effectiveness of the tapping and dragging detection component. We carried out the test in three scenarios: a library, an office at campus, and a pub. These situations represent environments with small, normal, and extreme background noise respectively.

First, we measured the accuracy of the tapping detection. In every experiment, we instructed a user to tap on a desk to make 300 clicks, each on a randomly selected position. The tapping strength is maintained to be a moderate level which is audible to the user.

Table 2 display the experiment results for false positive (false alarm) and false negative (mis-detection) rate. We can notice that the error rates remains relatively low (3.5% in the worst case). The results indicate the tapping detection algorithm is accurate and reliable.

We conduct the experiments again without using the gyroscope. The results show that in a quiet environment including the library and office, the system performs equally well even with the gyroscope disabled. However, in a noisy pub environment, the false positive rate soars into 15%, making the system unusable. It indicates that the gyroscope can greatly suppress the false positive rate in some noisy situation and improve the performance.

Next, we evaluate the effectiveness of the duration estimation algorithm. We requested a user to perform a 10-second dragging action after a tapping. The experiments were repeated for 50 times in the aforementioned three situations. Table 2 shows the dragging duration estimation and demonstrates fair accuracy. Even in a noisy environment, the error is still below a reasonable value (0.5 seconds).

Power Consumption

UbiTouch merely leverages built-in sensors including the PS, ALS, microphone, and gyroscope which are of low power consumption. Especially, the average operating

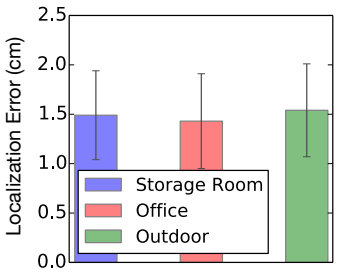

Figure 12: Localization accuracy under differnt scenearios.

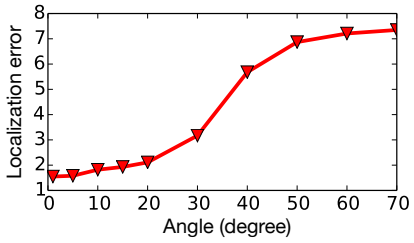

Figure 13: Localization error vs. smartphone inclined angle.

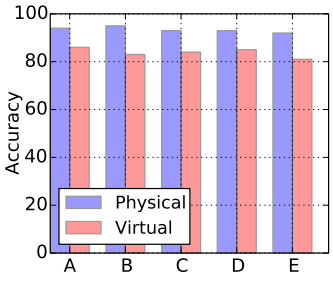

Figure 14: Recognition rates for 5 users.

current for the PS and ALS is only 176 μA [2]. We profile the power consumption of UbiTouch by utilizing a professional power monitor application called Powertutor [31]. Specifically, we measure the power cost in two situations: (1) idle with screen on (2) running UbiTouch with continuous touch input. Each situation lasts for 5 minutes. The average power consumption for each state is 6021 mW and 6930 mW respectively. Thus, UbiTouch incurs extra 15.1% power cost, that is slightly less than the power consumption of UbiK which is 18.5% [25].

Application Tests

UbiTouch can support various types of applications such as drawing and text-inputing. Here, we further evaluate the system performance by replaying the user input into the Google handwriting text-input application [8] and assessing the recognition rate. We carried out the experiments by requesting three users to input 100 random English alphabets. Fig. 15 shows two handwriting examples retrieved from the physical and virtual touchpads. Note that instead of feeding the finger trajectories to the application, we conduct some pre-processing to increase the recognition rate. Specifically, we apply a clustering algorithm called DBSCAN [28] to filter out some outliers and extract the skeleton of the input character, which is then fed into the recognition software.

Fig. 14 demonstrates the recognition rates for five users. The average recognition rate is 79%. In the worst case, the recognition rate for UbiTouch is 72%, still comparable to the accuracy of the physical touchpad which is 92%.

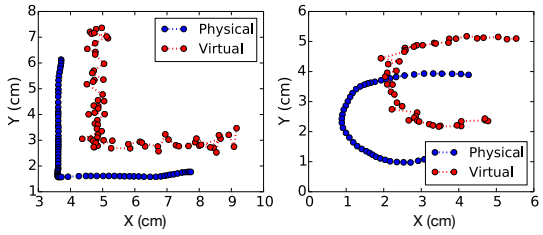

Figure 15: Handwriting examples for English characters L and C.

DISCUSSION

System Generalizability

Currently, the system is only implemented on a specific phone model. We found that smart phones from different vendors tend to equip different types of sensors, which means our system may only be able to run in a specific model till now. However, since almost every sensor could provide raw distance and luminance readings. Thus, we envision that the principles and methodologies explored in the paper could be generalizable and applicable on a number of devices.

Applicable scenarios

The prototype of UbiTouch is only applicable under two basic conditions: (1) the smartphone is placed on a desk in a vertical or slightly inclined position and remains still, and (2) throughout the usage life-cycle, the ambient lighting conditions do not vary drastically.

Regarding the first condition, users have to deploy the smartphone almost vertically, which is a somewhat unnatural position. The second condition not only requires that the positions and luminance of light sources basically remain unchanged, but also demands that there is no significant movement in the neighboring area of the user. We concede the rigidness of these conditions and envision that more sophisticated signal processing techniques will be useful for eliminating these limitations.

Finger gestures

While interacting with the system, the user is suggested to stay consistent in terms of finger gestures. A drastically change of the finger gesture (e.g., finger rotation/tilting) may negatively affect the tracking accuracy. We expect some pre-processing jobs to remove the impact of the gestures in the future.

CONCLUSION

In this paper, we present UbiTouch, a prototype system that enhances smartphones with virtual touchpads through the built-in smartphone sensors. We demonstrate that UbiTouch achieves centimeter-level localization accuracy and poses no significant impact on battery life of a smartphone.

REFERENCES

1. H. Abdelnasser, M. Youssef, and K. A. Harras, "Wigest: A ubiquitous wifi-based gesture recognition system," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1472–1480.
2. Avago, "Avago 9930 integrated proximity and ambient light sensor," http://www.avagotech.com/products/.
3. Avago-Developer, "Avago 9930 linux driver," https://github.com/CyanogenMod/android_kernel_lge_hammerhead/blob/cm-13.0/drivers/misc/apds993x.c.
4. P. Baudisch and G. Chu, "Back-of-device interaction allows creating very small touch devices," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2009, pp. 1923–1932.
5. X. Bi, Y. Li, and S. Zhai, "Ffitts law: modeling finger touch with fitts' law," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2013, pp. 1363–1372.
6. P. Demartines and J. Hérault, "Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets," *Neural Networks, IEEE Transactions on*, vol. 8, no. 1, pp. 148–154, 1997.
7. Google, "Environment sensor for android," http://developer.android.com/guide/topics/sensors/sensors_environment.html.
8. Google-input, "Google handwriting input," https://play.google.com/store/apps/details?id=com.google.android.apps.handwriting.ime.
9. C. Harrison, H. Benko, and A. D. Wilson, "Omnitouch: wearable multitouch interaction everywhere," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 441–450.
10. K. Hinckley and H. Song, "Sensor synaesthesia: touch in motion, and motion in touch," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 801–810.
11. S. Hiraoka, I. Miyamoto, and K. Tomimatsu, "Behind touch, a text input method for mobile phones by the back and tactile sense interface," *Information Processing Society of Japan, Interaction 2003*, pp. 131–138, 2003.
12. H. Ketabdar, K. A. Yüksel, and M. Roshandel, "Magitact: interaction with mobile devices based on compass (magnetic) sensor," in *Proceedings of the 15th international conference on Intelligent user interfaces*. ACM, 2010, pp. 413–414.
13. N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140–150, 2010.
14. H. Liu, S. Shah, and W. Jiang, "On-line outlier detection and data cleaning," *Computers & chemical engineering*, vol. 28, no. 9, pp. 1635–1647, 2004.
15. R. Love, *Linux kernel development*. Pearson Education, 2010.
16. Z. Lv, A. Halawani, M. S. Lal Khan, S. U. Réhman, and H. Li, "Finger in air: touch-less interaction on smartphone," in *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 2013, p. 16.
17. S. Nirjon, J. Gummeson, D. Gelb, and K.-H. Kim, "TypingRing: A wearable ring platform for text input," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 227–239.
18. I. Oakley and D. Lee, "Interaction on the edge: offset sensing for small devices," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2014, pp. 169–178.
19. B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: Particle filters for tracking applications*. Artech house Boston, 2004, vol. 685.
20. S. Robinson, N. Rajput, M. Jones, A. Jain, S. Sahay, and A. Nanavati, "Tapback: towards richer mobile interfaces in impoverished contexts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 2733–2736.
21. H. Roeber, J. Bacus, and C. Tomasi, "Typing in thin air: the canesta projection keyboard-a new method of interaction with electronic devices," in *CHI'03 extended abstracts on Human factors in computing systems*. ACM, 2003, pp. 712–713.
22. A. Soto, "Self adaptive particle filter," in *IJCAI*, 2005, pp. 1398–1406.
23. J. D. Villasenor, B. Belzer, and J. Liao, "Wavelet filter evaluation for image compression," *Image Processing, IEEE Transactions on*, vol. 4, no. 8, pp. 1053–1060, 1995.
24. J. Wang, D. Vasisht, and D. Katabi, "RF-IDraw: Virtual Touch Screen in the Air using RF Signals," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 235–246.
25. J. Wang, K. Zhao, X. Zhang, and C. Peng, "Ubiquitous keyboard for small mobile devices: harnessing multipath fading for fine-grained keystroke localization," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014, pp. 14–27.

26. D. Wigdor, C. Forlines, P. Baudisch, J. Barnwell, and C. Shen, "Lucid touch: a see-through mobile device," in *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, 2007, pp. 269–278.

27. A. S. Willsky and H. L. Jones, "A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems," *Automatic Control, IEEE Transactions on*, vol. 21, no. 1, pp. 108–112, 1976.

28. I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

29. N.-H. Yu, S.-S. Tsai, I.-C. Hsiao, D.-J. Tsai, M.-H. Lee, M. Y. Chen, Y.-P. Hung *et al.*, "Clip-on gadgets: expanding multi-touch interaction area with unpowered tactile controls," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 367–372.

30. C. Zhang, J. Tabor, J. Zhang, and X. Zhang, "Extending mobile interaction through near-field visible light sensing," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 345–357.

31. L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 105–114.