Input Finger Detection for Nonvisual Touch Screen Text Entry in *Perkinput*

Shiri Azenkot*, Jacob O. Wobbrock†, Sanjana Prasain*, Richard E. Ladner*

*Computer Science and Engineering | DUB Group
University of Washington
Seattle, WA 98195 USA
{shiri,ladner}@cs.washington.edu, pras5181@uw.edu

†The Information School | DUB Group
University of Washington
Seattle, WA 98195 USA
wobbrock@uw.edu

Abstract

We present *Input Finger Detection* (IFD), a novel technique for nonvisual touch screen input, and its application, the *Perkinput* text entry method. With IFD, signals are input into a device with multi-point touches, where each finger represents one bit, either touching the screen or not. Maximum likelihood and tracking algorithms are used to detect which fingers touch the screen based on user-set reference points. The Perkinput text entry method uses the 6-bit Braille encoding with audio feedback, enabling one- and two-handed input. A longitudinal evaluation with 8 blind participants who are proficient in Braille showed that one-handed Perkinput was significantly faster and more accurate than iPhone's VoiceOver. Furthermore, in a case study to evaluate expert performance, one user reached an average session speed of 17.56 words per minute (WPM) with an average uncorrected error rate of just 0.14% using one hand for input. The same participant reached an average session speed of 38.0 WPM with two-handed input and an error rate of just 0.26%. Her fastest phrase was entered at 52.4 WPM and no errors.

Keywords: Text entry, blind, mobile devices.

Index Terms: H.5.2. Information interfaces and presentation: User interfaces — *Input devices and strategies*. K.4.2. Computers and society: Social issues – *assistive technologies for persons with disabilities*.

1 Introduction

As mobile touch screen devices become increasingly ubiquitous, they are also becoming more accessible to people with visual impairments. Both Apple and Google have developed screen readers that are pre-installed on their devices, but Apple's iPhone seems to be the most popular touch screen phone among blind people. The iPhone is the only touch screen phone recommended by the National Federation of the Blind (NFB) [14], the American Foundation for the Blind [2], and the Royal National Institute of Blind People [18]. Meanwhile, Android's accessibility features require a complex setup process [2].

Although advancements in touch screen accessibility have been made, the fundamental task of text entry remains slow and error-prone. Much work has recently arisen in this area [3][4][13][16][17], highlighting the compelling need for a better input method. Bonner *et al*. [3] found that the mean text entry rate on an iPhone with VoiceOver was only 0.66 words per minute (WPM) and Oliveira *et al*. [16][17] report a mean speed of 2.1 WPM with a VoiceOver-like input method. Clearly, a better text entry method is needed for blind and eyes-free text input.

While VoiceOver interaction involves selection of virtual keys, we propose a novel multi-touch technique based on signal detection theory called *Input Finger Detection* (IFD). In this technique, a signal is input by touching the screen with several fingers where each finger represents one bit, either touching the screen or not. A user sets reference points anywhere on the screen at any time to indicate initial finger positions, and we use maximum likelihood to decide which fingers were used to input the touch points based on the reference points. We then track the reference points to ensure they accurately represent the "true" locations of the fingers.

Using IFD, we developed *Perkinput*, a nonvisual text entry method that uses the 6-bit Braille character encoding. To enter a character with two hands, a user taps the screen with up to 3 fingers from each hand. The fingers contacting the screen correspond to dots in the Braille character. When using one hand, a user enters 3 bits of a character at a time, performing two multi-point touches to enter one character (see Figure 1).

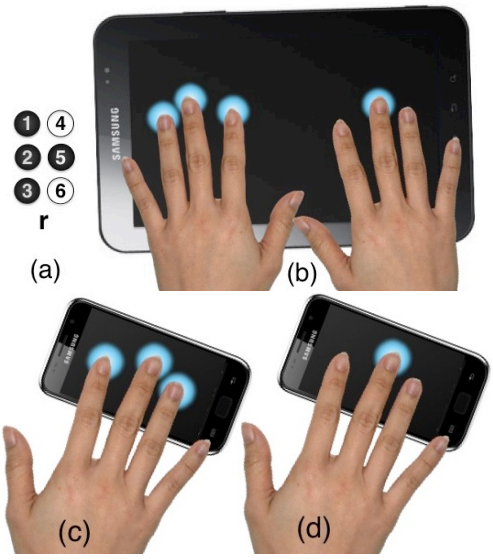

Figure 1: A user enters a character with Perkinput using one or two hands. The (a) Braille encoding for 'r' is the binary string "111010." It is entered with two hands simultaneously as shown in (b), or with one hand by touching the phone twice. The first touch (c) inputs bits 1-3 and the second touch (d) inputs bits 4-6.

Additionally, we present a novel multi-device interaction technique that uses the screen of a second device to enter text twice as fast. Two-device Perkinput enables a user to enter text into a device with a small screen using *both* hands, by using a

Graphics Interface Conference 2012
28-30 May, Toronto, Ontario, Canada

second device's screen for additional input space. Thus, instead of inputting characters with two touches by one hand, the user enters text twice as fast with both hands simultaneously.

To evaluate Perkinput, we conducted a longitudinal study with eight blind participants comparing one-handed Perkinput to VoiceOver. Study results show that one-handed Perkinput was significantly faster and more accurate, and had a higher learning rate than VoiceOver. Average entry rates during the last session were 7.26 WPM (*SD* = 2.64) with one-handed Perkinput and 4.52 WPM (*SD* = 1.51) with VoiceOver. Uncorrected error rates were low, averaging 2.13% (*SD* = 4.01) and 4.45% (*SD* = 5.73), respectively.

With further practice, one Perkinput user reached an average entry rate of 17.56 WPM (*SD* = 3.36) as part of a case study we conducted to evaluate expert performance. This user had only 0.14% (*SD* = 3.18) uncorrected errors.

We also evaluated performance of two-handed Perkinput using a tablet and two connected phones with our case study participant. Perkinput on two devices was faster than one-handed Perkinput, with an average speed of 20.4 WPM (*SD*=8.4). More practice would probably raise the average speed well beyond that of one-handed Perkinput. On a tablet, on the other hand, Perkinput was more than twice as fast as on a phone, with an average speed of 38.0 WPM (*SD* = 9.1) and an error rate of 0.26% (*SD* = 1.43).

Our main contributions are the theory of the IFD technique and its application, the Perkinput text entry method, with which we have achieved higher text entry rates than any other known nonvisual text entry method in the HCI literature. A second contribution is a novel two-device interaction technique, where two devices are used to enter text at a faster rate than with just one device. A third contribution is the empirical findings from our three studies, which demonstrate the potential of Perkinput for Braille-based text entry as well as the IFD technique in general.

2 Related Work

Several nonvisual touch screen text entry methods have been proposed in the HCI literature, some of which are based on Braille. Perkinput differs from these methods because it uses the new concept of IFD, while other methods use primarily gestures or virtual keys. We discuss the benefits of IFD over the two existing paradigms in section 4. No other work reports entry rates for blind people that are comparable to our findings. Also, we are the first to conduct a longitudinal evaluation and an evaluation of nonvisual text entry on a tablet.

One recent Braille-based text entry method is BrailleType [16][17], where the screen is split into six virtual keys representing the dots in a Braille cell. The user enters a character by selecting the regions that correspond to the raised dots in the character. Oliveira *et al*. compared BrailleType to VoiceOver input in a one-session study with 15 blind people. They found that with an entry rate of 1.45 WPM, BrailleType was significantly slower than VoiceOver, albeit more accurate.

Similarly, Frey *et al*. [4] describe BrailleTouch, a text entry method that also has six large virtual keys. The difference between BrailleType and BrailleTouch is that the latter requires the user to input all dots for a character simultaneously. To do this, the user holds the phone in a nonstandard way, with both hands in landscape orientation facing the screen away from the user. No evaluation of BrailleTouch is described.

TypeInBraille [13] is another Braille-based entry method that uses three simple gestures to input one character based on its Braille encoding. There is no evaluation of TypeInBraille.

Several nonvisual text entry methods in the literature are *not* based on Braille. No-Look Notes [3] is a text entry method with virtual keys where two keys must be selected and activated to enter one character. An evaluation with blind people compared No-Look Notes to VoiceOver, finding that entry rates were 0.66 WPM and 1.32 WPM, respectively. Sánchez and Aguayo [21] also propose a method with virtual keys but have no evaluation.

Oliveria *et al*. [17] present NavTouch, a touch screen method that uses a sequence of gestures to "navigate" through the alphabet until a letter is selected. A single-session evaluation found that NavTouch was slower than VoiceOver, with an entry rate of just 1.72 WPM, and an error rate of over 10%.

Tinwala and MacKenzie [25] and Yfantidis and Evreinov [30] also presented gesture-based entry methods but conducted evaluations with blindfolded sighted users. Their results do not apply to blind people because the gesture input ability of a blind person differs significantly from that of a sighted person [8].

Several of the text entry methods mentioned above were compared to iPhone's VoiceOver text entry method. Given reviews and recommendations from blind organizations [2][15][18] and anecdotal accounts, the iPhone seems to be by far the most common touch screen device used by blind people. Introduced in 2009, VoiceOver is the iPhone's built-in screen reader. When VoiceOver is running, a user can touch a virtual key to hear its description and double-tap or split-tap [7] to activate that key. A split-tap, initially termed a "second-tap" [7], involves touching the screen with a second finger while keeping the first in contact with the screen. VoiceOver thus enables eyes-free text entry on the iPhone's virtual QWERTY keyboard.

In addition to touch screen methods, Perkinput's chording input resembles the Twiddler, a one-handed mobile keyboard used with wearable computers. Lyons *et al*. [11] conducted a longitudinal study and found that text entry with Twiddler was much faster than with multi-tap. Although Twiddler has physical keys, Lyon *et al*. highlight the potential of chording methods for fast, eyes-free input.

3 Input Finger Detection (IFD)

Touch screen input can be modeled by transmission of information over a noisy channel [22]. With Input Finger Detection (IFD), a user first positions her hand over the screen and sets *n* reference points with, for example, a long press of *n* fingers. Then, the user transmits a message into the device by encoding the message into multi-point touches, each representing a binary sequence with *n* bits. If the *i*th finger contacts the screen for a given touch, the *i*th bit in the binary sequence is 1. Otherwise, the *i*th bit is 0. The touch is input into a device with some inconsistency, since the user will not hit the reference points exactly with every subsequent touch. This inconsistency is analogous to the message passing through a noisy channel in our model. The device receives the noisy, encoded message and the input method decodes it using our detection algorithms.

There are three sources of noise in our model:

1. *Hand repositioning*. The user may reposition a hand on a device, such that the fingers are no longer near their reference points.
2. *Touch-point inconsistency*. As with mouse clicks around a target, there is natural error that occurs when a user attempts to touch a consistent point on the screen.
3. *Hand drift*. It is likely that the user's hand drifts slowly over time as she touches the screen repeatedly.

To correct for noise caused by (1), the user must simply set new reference points that reflect the new position of her hand. For (2), we use *Maximum Likelihood* (ML) [19] to detect which finger corresponds to which point while accounting for the distribution of points around the target reference points. To minimize decoding errors caused by hand drift in (3), we *track* the reference points after each touch.

Our approach applies to one or two dimensions with signals of up to 10 bits, since people have 10 fingers and can touch a two-dimensional screen. For simplicity, we explain these concepts in the following section with examples for 3-bit encodings and one dimension only. Extending the ideas to two dimensions and longer bit encodings is straightforward.

3.1 Maximum Likelihood for Detecting Fingers

To determine which fingers touched the screen, we compare the input touch points to the reference points most recently set by the user. We assume that the user has not repositioned her hand, so the touch points for each finger are likely to be close to their reference points. There is inconsistency when touching the screen, however, and some fingers may have a greater variance than others.

We correct for touch point inconsistency by using Maximum Likelihood (ML) [19] to detect which fingers touched the screen given the input touch points, thereby determining which bits in the signal are equal to 1. We assume the distribution of touch points around their respective reference points is Gaussian, centered at their reference points.

To formalize our approach, let d be an observed data point and θ be the parameter we seek. Our hypothesis that θ is the "true value" is h_θ.

Suppose d is one point that corresponds to one reference point θ. We have three hypotheses for θ: h_1, h_2, h_3, where h_i is the hypothesis that d was input with finger i. $P(d|h_\theta)$ is the probability of observing d given h_θ and the ML detection is the value of θ that maximizes $P(d|h_\theta)$. We express $P(d|h_\theta)$ as a Gaussian distribution centered around reference point θ with variance σ^2.

$$P(d|h_\theta) = \frac{1}{(2\pi)^{1/2}\,\sigma}\exp\left[-\frac{(d-\theta)^2}{2\sigma^2}\right] \quad (1)$$

Figure 2 shows (in one dimension) the likelihood distribution of d given each hypothesis. The variance of the errors around the reference points differ, so even though the touch point is closer to reference point 2, it is *more likely* to be input by finger 1. In other words, $P(d|h_1)$ is greater than $P(d|h_2)$. The detected signal for Figure 2 is thus the binary sequence "100," since only finger 1 touched the screen.

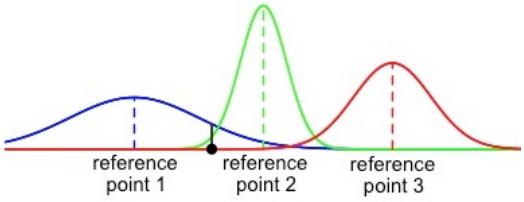

Figure 2: The curves show the likelihood distributions of a touch point input by each finger. The black dot marks the location of a touch point. The height of the curves above the touch point show that it was most likely input by finger 1, even though the point is closer to reference point 2 than to reference point 1.

In the case where the observed data has more than one point, the ML detection is the set of values for θ_j that maximizes the sum of $P(d_j|h_\theta)$ for each point j in the touch. This is valid if we assume that pointing variance is independent for points in the same touch. The set of values for θ_j must be unique and increasing, since one finger cannot touch the screen at more than one point and we assume fingers do not cross.

Figure 3 shows the likelihood distributions for three reference points and two touch points. The ML detection is {1,2}, which maximizes the sum of the likelihood of the reference points given their respective input points. The detected binary sequence is "110."

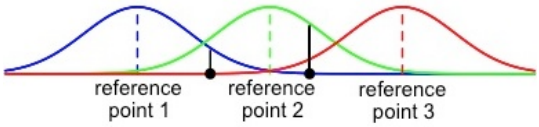

Figure 3: The black dots mark the location of 2 touch points and the black lines show the probabilities that the first point was input by finger 1 and the second point was input by finger 2, whose sum is higher than any other combination of reference points.

As Figure 2 and Figure 3 show, the variance of touches around a reference point is an important factor in IFD. Variances can be determined per finger per user by observing the distances between input points and their corresponding reference points. Adjustments can be made dynamically as well, by updating the variance with new data as the user touches the screen.

3.2 Tracking Reference Points

While our maximum likelihood detection is based on the assumption that a user keeps her fingers close to their reference points, we observe that a user tends to move her hand as she touches the screen repeatedly. To account for this movement, or drift, we *track* the reference points by adjusting them slightly with every input. This technique is based on a first-order Phase-Locked Loop [5], commonly used in electronic applications.

After we determine which fingers touched the screen, we move each reference point towards its corresponding touch point by a fraction of the distance between them. Most touches, however, were not made with all 3 fingers, so we cannot update all reference points based on their corresponding touch points alone. We aim to correct for drift of the entire hand, so we assume that the drift of individual fingers is correlated with one another. Therefore, we use the error of *each* touch point to adjust *all* reference points, but to varying degrees.

We formalize this concept with the following equation:

$$R_{n+1} = k \cdot C \cdot E_n + R_n \quad (2)$$

where R_n is the vector of reference points at time n. E_n is the vector of errors, or differences, between the touch points input at time n and their corresponding reference points.

$$R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix},\ E = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \quad (3)$$

The scalar constant k is the *adaptation coefficient* that indicates the extent to which errors are used to shift reference points. Smaller values of k reduce the effect of tracking.

Lastly, C is the *correlation coefficient matrix*, which describes the degree to which the error of one input point affects *each* reference point. As an extreme example, suppose we assume there is no correlation among the fingers as they move. The correlation coefficient matrix that reflects this assumption is:

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

When using $\boldsymbol{C}$ from Eq. 3, the tracking of a reference point $\boldsymbol{r_i}$ is only affected by an input point with finger i. If a touch did not include a point made by finger i, then $\boldsymbol{r_i}$ is not updated. The specific matrix we use in Perkinput is described in section 5.2.

4 IFD for Nonvisual Input

We discuss advantages of IFD for nonvisual input compared with the *de facto* approaches of virtual keys and gestures.

First, our approach is inherently customizable to support a user's hand size, pointing inconsistencies, and hand drift tendency. Unlike with virtual keys, reference points can be set anywhere on the screen and there are no predefined boundaries. Also, as we show with Perkinput, encodings can easily be adapted for one- or two-handed input. This is useful for different screen sizes such as phones and tablets and user abilities and preferences.

IFD is fast, since there is no need to search for virtual keys or draw gestures. With methods like VoiceOver that have many small keys, the user must first explore the screen with audio feedback until the desired key is found, then activate the key. This two step process is slower than tapping the screen. Some methods use large virtual keys [3][4][16][21] to avoid or quicken the explore-and-activate process, but as keys get larger, fewer fit on the screen. In our approach, however, there are up to 2^n possible inputs for one tap, where n is the number of reference points. Simple gestures for input are limited in the same way as large keys, since the number of possible simple gestures is small and several gestures must be input to enter one signal [13][17][30].

Finally, the algorithms we use and the chording nature of the technique make it robust to errors. The ability to reset reference points and the ML and tracking algorithms account for the three sources of noise in our model. Virtual keys have boundaries that lack tactile indications on a touch screen, so they are inherently visual. Also, complex gestures [25] are difficult for blind people to input [8]. Furthermore, input finger detection, like any chording technique, enables a user to discern through proprioception whether her input is correct. By contrast, in techniques based on a fingertip at a screen position, all inputs "feel" the same.

In the following sections, we describe an application of IFD for Braille-based text entry for blind people, showing how the theory we presented works in practice.

5 The Perkinput Text Entry Method

We begin by giving an overview of Braille and describe how we used IFD in Perkinput. For simplicity, we will hereafter refer to the "index" finger as finger 1, the "middle" finger as finger 2, and the "ring" finger as finger 3. Reference points 1, 2, and 3 correspond to those fingers, respectively.

5.1 Braille and Perkins Braillers

Braille is a tactile alphabet where each character is represented by raised dots in a 3×2 grid. The dots are numbered as shown in Figure 4. Braille is often embossed with a Perkins Brailler, a mechanical device that has 6 keys that correspond to the dots in a Braille cell (see Figure 4). To enter a character, the user simultaneously presses the keys that correspond to the dots raised in the character. People commonly learn how to type on a Perkins Brailler as they learn to read Braille.

Figure 4: A Perkins Brailler on the left and the Braille letter "p" on the right. The numbers above the keys show which keys correspond to which dots in a Braille cell.

5.2 IFD in Perkinput

Perkinput uses IFD with the Braille character encoding. This encoding offers an advantage for blind people who know Braille. For two-handed input, Perkinput uses the same dot-to-finger mapping as the Perkins Brailler. Fingers 1, 2, and 3 on the left hand enter dots 1, 2, and 3 respectively. On the right hand, fingers 1, 2, and 3 enter dots 4, 5 and 6, respectively. For one-handed input, two touches are required to enter one Braille character. During the first touch, fingers 1, 2, and 3 enter dots of the same number and during the second touch, fingers 1, 2, and 3 enter dots 4, 5, and 6. One-handed input is useful for devices with small screens such as phones. The second hand is free to hold the device or perform other tasks such as read Braille. Figure 5 shows which dots correspond to each finger for one-handed input.

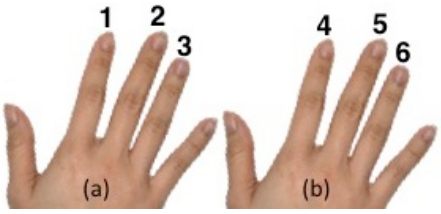

Figure 5: The numbers above the fingers show which fingers correspond to which Braille dots for (a) the first and (b) the second touch inputs with one-handed Perkinput.

A user registers reference points with a "long press." If six fingers are registered, Perkinput uses a 6-bit encoding. If only three fingers are registered, Perkinput uses the one-column-at-a-time 3-bit encoding. Reference points can be set at any time and anywhere on the screen, whether in portrait or landscape orientation. The only restriction when registering reference points is that finger 1 must be closer to the top of the phone than finger 3 in landscape orientation, or closer to the left side of the phone in portrait orientation. This enables Perkinput to determine which reference point corresponds to which finger.

For IFD, we assume each reference point has a bivariate Gaussian touch point distribution with constant and equal variance along the x- and y-axes, with a covariance of zero. This simplifies the problem of ML detection to finding the nearest reference point(s) in two dimensions. In the future, we plan to dynamically track a user's finger variance. We derived the following tracking constants through experimentation:

$$k = 0.1, \qquad C = \begin{bmatrix} 1 & 0.4 & 0.4 \\ 0.4 & 1 & 0.4 \\ 0.4 & 0.4 & 1 \end{bmatrix} \tag{5}$$

Although using the Braille encoding has advantages, it does not include encodings for all characters used in text entry. Perkinput uses swipes to represent characters such as spaces and backspaces. A two-finger swipe in any direction represents a space and a three-finger swipe in any direction represents a backspace. This involves placing the fingers on the screen and moving them while still touching the screen.

During one-handed input, we also added a gesture to input "blank" Braille columns. Some Braille characters do not have dots 4, 5, or 6 raised, Perkinput uses a one-finger swipe to indicate that no dots are raised (*i.e.*, a binary sequence "000"). For example, the letter "a" is encoded in Braille as dot 1, or the binary sequence "100000." A user enters an "a" by touching the screen with finger 1 and then swiping the screen with one finger. This maintains that

all Braille-encoded characters require two touches in the one-handed version of Perkinput.

Perkinput provides audio feedback with every touch, including entry of the first column of a character and registration of reference points. When a user enters a character, that character is spoken.

5.3 Perkinput for Two Devices

As touch screen devices become more common and less expensive, it is interesting to explore interaction techniques using two devices at once. Many people carry multiple devices, such as a phone and a music player, or a phone and a tablet. Even two of these mainstream devices are usually less expensive and easier to carry than devices with physical Braille keyboards, which cost $1,000 or more [1].

In two-device Perkinput, two BlueTooth-connected touch screen devices are used to input text into one device, which we call the *primary* device. The user inputs dots 1, 2, and 3 with the left hand on one device and dots 4, 5, and 6 with the right hand on a second device, as shown in Figure 6. We define the "primary device" as the device that initiated the BlueTooth connection and accepts input of dots 4, 5, and 6.

The "secondary device" sends the primary device information about each multi-point touch it receives, which the primary device then decodes along with its own input to determine what character was entered. The user must register reference points on each phone. The primary device identifies characters in the same way standard Perkinput decodes the two touches required to enter a character—the only difference is that the first touch is now received from the non-primary device and has a different set of reference points than the second touch.

Figure 6: With two devices, the left hand inputs dots 1, 2, and 3 and the right hand inputs dots 4, 5, and 6. The right phone is the primary phone, defined as the phone that initiates the BlueTooth connection and expects dots 4-6 as input.

Two-device Perkinput decodes a character after it receives input from each device and there is no time limit. If the user touches one device twice before touching the second device, the first input on the former device is disregarded. Thus, to indicate a "blank" column, the user inputs a swipe as she does on one-handed Perkinput.

6 Study 1: One-Handed Perkinput vs. VoiceOver

To evaluate Perkinput on a small touch screen, we compared the one-handed version to the *de facto* standard in nonvisual touch screen text entry, iPhone's VoiceOver. We conducted a longitudinal study with eight blind participants who knew Braille to evaluate performance over time.

6.1 Method

6.1.1 Participants

We conducted the study with eight blind participants (five female, three male), with an average age of 55. None of the participants were able to visually identify keys on a phone. One participant had experience with VoiceOver but the others had little to no experience with touch screen devices. All participants were proficient with reading Braille and, as is typical, entering text on a Perkins Brailler. We required Braille proficiency when recruiting participants since memorizing the Braille character encodings is a separate task from entering text. All users were also proficient at typing on a QWERTY keyboard, so they were familiar with the layout of the soft keyboard used with VoiceOver.

6.1.2 Apparatus

We developed a prototype of one-handed Perkinput and VoiceOver input for a Samsung Galaxy Phone running the Android operating system. We implemented our own version of VoiceOver standard input instead of using an iPhone to control variables related to device hardware features such as touch screen sensitivity, and quality of audio feedback (we used the same audio files as feedback for both methods). There is precedence for comparing a text entry method to a research implementation of VoiceOver in prior work [16][17].

Our prototype did not include any contractions (Grade 2 Braille) to maintain a fair comparison with VoiceOver. The same number of characters had to be entered for any given phrase on each method.

The prototypes were instrumented with mechanisms to load random phrases and log information about each input, including time and screen coordinates. Phrases were spoken by the built-in text-to-speech engine. Character names were pre-recorded sound files.

6.1.3 Procedure

The study included eight sessions, the first of which was a training session and was not included in the study results. During training, we conducted a brief interview and explained one-handed Perkinput and VoiceOver to a participant until he/she independently typed two phrases with each method.

All other sessions were 75 minutes long, where participants typed phrases for 30 minutes with each method. We used a set of short phases [29] that are representative of the English language. No phrase was longer than 27 characters. Participants were instructed to type "as quickly and accurately as possible." Each 30-minute phrase entry period was preceded by a 5-minute warm-up. Participants received a 5-minute break between methods.

Since participants were blind, we were unable to present them with phrases visually. To reduce cognitive load, we used short phrases derived from a standard set of text entry evaluation phrases [29]. The prototypes spoke a phrase and participants were instructed to repeat the phrase verbally before entering it to ensure the phrase was understood.

6.1.4 Design and Analysis

This study was a 7×2 within-subjects factorial design with factors for *Session* and *Method*. The levels of *Session* were (1-7); the levels of *Method* were (one-handed Perkinput, VoiceOver). The average number of trials run per session per method was 13.28 ($SD = 9.31$), with a maximum of 46 trials. A total of 2362 trials were conducted, of which 1363 (57.7%) were for Perkinput and 999 were for VoiceOver (42.3%).

For analyzing words per minute (WPM), a mixed-effects model analysis of variance was used [10] with fixed effects of *Session* and *Method*; *Trial* was included as a nested factor within *Session* and *Method*. In addition, *Participant* was modeled as a random effect to account for correlated measurements within subjects over

time [9]. In general, mixed-effects models preserve large denominator degrees-of-freedom but compensate with wider confidence intervals. In addition, mixed-effects models can, unlike traditional fixed-effects ANOVAs, accommodate unbalanced data such as ours, where we had a variable number of trials per session.

Both uncorrected and corrected error rates were measured [24][26], the former being errors left in final transcriptions and the latter being from backspacing during entry. As is typical, neither uncorrected nor corrected error rates were remotely normal in their distribution (Shapiro-Wilk $W > .65$, $p < .0001$; see [23]), ruling out the use of parametric analyses. Therefore, we used the nonparametric *Aligned Rank Transform* procedure [6][20][28], which enables the use of ANOVA after alignment and ranking, and which maintains the integrity of interaction effects. The same model terms were used for error rates as for WPM.

Subjective *Method* preferences were analyzed using a one-sample Pearson Chi-Square test of proportions.

To validate counterbalancing, a test of order effects was conducted utilizing an *Order* factor encoding *Method*'s order-of-presentation (*i.e.*, whether a method was first or second in a given session). *Order* was not statistically significant for WPM ($F_{1,2351} = 1.51$, *n.s.*), and neither was *Order×Method* ($F_{1,2352} = 0.00$, *n.s.*), indicating adequate counterbalancing and no asymmetric skill transfer.

6.2 Results

6.2.1 Words per Minute (WPM)

Average entry speed over all sessions for one-handed Perkinput was 6.05 WPM ($SD = 2.69$). For VoiceOver, it was 3.99 WPM ($SD = 1.65$). This difference resulted in a significant effect of *Method* on WPM ($F_{1,1880} = 224.25$, $p < .0001$). Figure 7 shows WPM graphed over sessions.

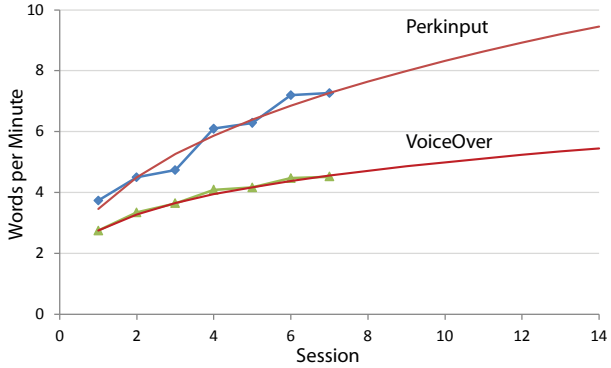

Figure 7: Entry rates as WPM over sessions 1-7, and extrapolated to session 14, for each input method. Higher is better

6.2.2 Performance over Time

We can examine how entry speeds varied over time. There was a significant effect of *Session* on WPM ($F_{6,1878} = 51.36$, $p < .0001$), as both methods improved over time. There was also a significant *Session×Method* interaction ($F_{6,1878} = 5.94$, $p < .0001$), as each method improved at a different rate over time. As can be seen in Figure 8, Perkinput improved more quickly than VoiceOver.

It is customary (see, *e.g.*, [11][12][26]) to fit power laws of the form $y = ax^b$ to entry speeds, where y is WPM, x is *Session*, a is initial speed and b reflects the learning rate. Doing so results in the following equations for each method:

$$\text{Perkinput:} \quad y = 3.4589x^{0.3812},\ R^2 = .9546 \qquad (6)$$

$$\text{VoiceOver:} \quad y = 2.7450x^{0.2600},\ R^2 = .9893 \qquad (7)$$

From Figure 7 and the above equations, we can see that Perkinput was both initially faster than VoiceOver *and* improved at a faster rate. Extrapolating to twice the sessions that were conducted gives entry speeds of 9.46 WPM for Perkinput and 5.45 WPM for VoiceOver in the 14th session.

6.2.3 Error Rates

We calculated both uncorrected and corrected error rates [24]. Note that while uncorrected error rates are at odds with entry rate, corrected errors take time to produce, and thus are *subsumed* by entry rate. A successful text entry method could make and fix mistakes "along the way" resulting in high corrected errors, but in the end, produce more accurate text in less overall time having low uncorrected errors [26]. Thus, uncorrected errors are of chief importance. Figure 8 shows uncorrected error rates by method over sessions.

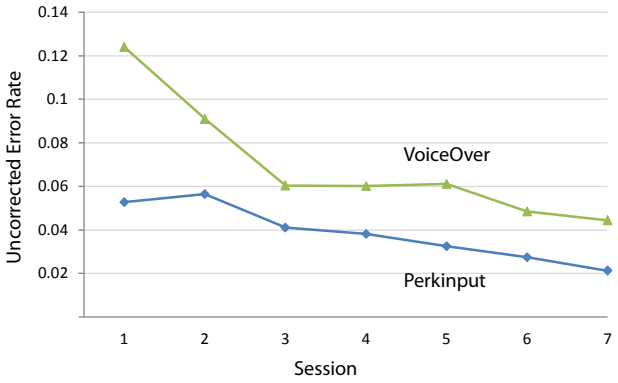

Figure 8: Uncorrected error rates over sessions 1-7 for each input method. Lower is better.

The average overall uncorrected error rate for Perkinput was 3.52% ($SD = 5.87$). For VoiceOver, it was approaching twice that, at 6.43% ($SD = 7.16$). This difference was significant ($F_{1,1882} = 91.87$, $p < .0001$). Furthermore, the overall uncorrected error rate went down significantly over time ($F_{6,1878} = 19.41$, $p < .0001$), and did so differentially by method ($F_{6,1878} = 4.51$, $p < .001$). Thus, we see that not only was Perkinput faster than VoiceOver initially and over sessions, but it resulted in fewer errors initially and over sessions as well.

Corrected errors give insight into the amount of "error fixing" taking place during entry. Despite Perkinput creating more accurate text in less overall time, it did so while making more errors and error corrections. Corrected error rates were 12.23% ($SD = 11.64$) for Perkinput and 8.32% ($SD = 10.53$) for VoiceOver. This difference was significant ($F_{1,1881} = 41.97$, $p < .0001$). Overall, corrected error rates decreased significantly over sessions ($F_{6,1878} = 17.33$, $p < .0001$), but did not do so differently for each method ($F_{6,1878} = 1.52$, *n.s.*).

6.2.4 Subjective Performance

At the end of the longitudinal study, participants were asked which method they preferred overall. In light of the performance results in favor of Perkinput, it is perhaps not surprising that 6 of 8 participants preferred this method to VoiceOver. Nonetheless, a one-sample Pearson Chi-Square test of proportions does not show

preference for Perkinput was significantly different than chance ($\chi^2_{(1,N=8)} = 2.00, p = .16$).

7 Study 2: Expert Performance with One-Handed Perkinput

We sought to assess expert performance on one-handed Perkinput after additional hours of practice by focusing on one participant who strongly preferred Perkinput to VoiceOver.

7.1 Method

We conducted a case study with one participant (female, age 59) from Study 1, who achieved the highest text entry rates with Perkinput. We refer to this participant as P5. P5 was proficient in Braille but had little experience with touch screens. We conducted six additional sessions with P5, each consisting of 30 minutes of entering phrases with Perkinput. We used a similar procedure to that of Study 1, but we did not test VoiceOver.

We used the same Perkinput prototype from Study 1 but made slight modifications according to feedback from P5. Instead of using swipes to represent "blank" columns and spaces, we used a tap by the "pinky" finger, which we call finger 4. A "blank" column is entered by tapping finger 4 and a space is entered by tapping finger 4 twice (like two "blank" columns). Also, a backspace was given the Braille encoding of dots 1, 2, 3, 4, 5, and 6, instead of a three-finger swipe. We envision different ways of inputting "blank" columns and characters that lack Braille encodings to be set by the user in a preference dialog, since some users may find it easier to use swipes than pinky-taps.

7.2 Results

Not surprisingly, P5 improved her entry rate significantly over all sessions ($F_{12,533} = 55.38$, $p < .0001$). Her speed in the 13th session was 15.96 WPM ($SD = 4.10$) over 61 trials, but this was not her maximum session speed, which occurred in the 12th session and was 17.56 WPM ($SD = 3.36$) over 66 trials. By comparison, her speed in session 1 had just been 4.79 WPM ($SD = 1.93$) over 22 trials. Thus, P5 more than tripled her entry rate after 12 sessions, or about 6.5 total hours of one-handed Perkinput usage.

P5's single fastest phrase occurred in session 11, at 22.47 WPM with no uncorrected or corrected errors (0.0%).

Figure 9 shows P5's entry rates with Perkinput from sessions 1-13 and extrapolated to session 26. For Study 1's sessions (1-7), P5 averaged 8.09 WPM ($SD = 3.60$). For the additional sessions of Study 2 (8-13), P5 averaged 14.12 WPM ($SD = 4.88$), an improvement of 74.5%. The extrapolated speed in session 26 is 24.78 WPM, which is only about 2 WPM greater than P5's fastest phrase reported from her 11th session.

The power law equation for P5 is shown below. Note the remarkably high exponent (b) of 0.6348, indicating a rapid rate of improvement for P5 compared to the average over participants of 0.3812.

Perkinput: $y = 3.1332x^{0.6348}$, $R^2 = .8435$ (8)

P5 corrected almost all of her errors during entry, so error rates for P5 were exceptionally low. P5's average overall uncorrected error rate for sessions 1-13 was 0.50% ($SD = 1.87$), compared to 3.52% for all participants for sessions 1-7. During entry, P5 made, on average, 9.16% ($SD = 9.63$) corrected errors, not too far below participants' average of 12.23% for sessions 1-7. Thus, P5 shows that Perkinput, while creating some errors during entry, enables rapid enough error correction so as to achieve noteworthy speeds with few uncorrected errors in the end.

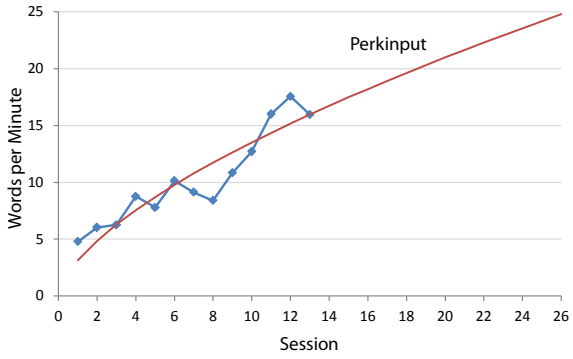

Figure 9: Entry rates for P5 with one-handed Perkinput over sessions 1-13, and extrapolated to session 26. Higher is better. Note that unlike Figure 7 the Y-axis here reaches 25 WPM, not just 10 WPM.

8 Study 3: Evaluating Two-Handed Perkinput

After evaluating one-handed Perkinput, we wanted to compare text entry rates to those achieved using two hands.

8.1 Method

We conducted another case study with P5 who already had 6.5 hours of practice with one-handed Perkinput. The procedure was similar to Study 1, but included only two sessions, the first of which was a training session and not included in the results. During the training session, P5 entered phrases with two hands for 30 minutes on a tablet and another 30 minutes on two phones. During the latter session, P5 entered phrases for 30 minutes on each device setup, preceded by 5 minutes of warm-up. As in Study 1, P5 took a 5-minute break between methods.

P5 entered text on a Samsung Galaxy tablet. For two-device input, we used two Samsung Galaxy phones.

8.2 Results

P5's speeds with each method are shown in Figure 10. Recall that the average WPM for one-handed entry from P5's *fastest* session (#12) was 17.56 WPM ($SD = 3.36$). With two devices, it was 20.43 WPM ($SD = 8.41$). With two hands on one device, it was 38.02 WPM ($SD = 9.31$). Overall, these differences were significant ($F_{2,222} = 171.92$, $p < .0001$).

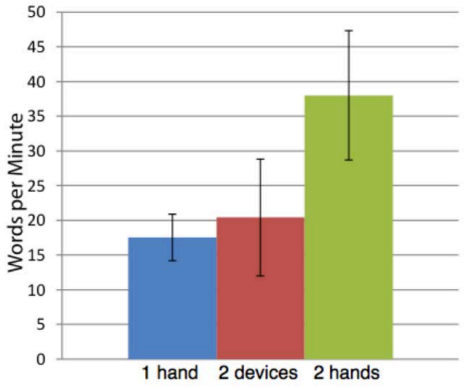

Figure 10: Entry rates in WPM for P5 with one-handed, two-device, and two-handed-one-device entry with Perkinput. Higher is better. Error bars represent ±1 SD.

Pairwise comparisons reveal that tablet entry was significantly faster than both two-device ($F_{1,222} = 149.86$, $p < .0001$) and one-handed entry ($F_{1,222} = 287.48$, $p < .0001$). Two-device entry, however, was not quite significantly faster than one-handed entry, although a trend is clear in that direction ($F_{1,222} = 3.31$, $p = .07$). Recall that the two-device data was used only once for this testing, while the data for one-handed Perkinput were taken from P5's fastest overall session.

P5's error rates with each method are shown in Figure 11. Uncorrected error rates were near zero for all three versions of Perkinput. Average uncorrected error rates were 0.14% ($SD = 0.81$) for one-handed Perkinput, 1.75% ($SD = 3.18$) for two-device Perkinput, and 0.26% ($SD = 1.43$) for two-handed Perkinput on a tablet. Corrected error rates were also low, at 3.10% ($SD = 3.86$), 6.81% ($SD = 7.86$), and 3.19% ($SD = 4.74$), respectively. Thus, it seems two-device Perkinput was more error prone than the other methods, although all methods exhibited low uncorrected error rates, making speed measurements comparable.

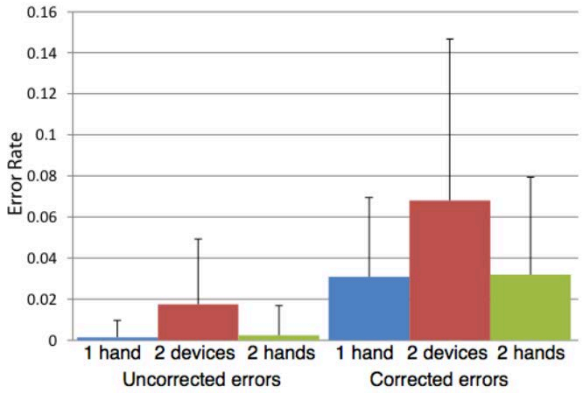

Figure 11: Error rates for P5 with one-handed, two-device, and two-handed-one-device entry with Perkinput. Lower is better. Error bars represent ±1 *SD*.

9 Discussion

In Study 1, our longitudinal study, we showed that one-handed Perkinput outperforms VoiceOver, the *de facto* eyes-free text entry method for touch screen devices, for users who are familiar with Braille. Perkinput was faster and more accurate in the first as well as the final sessions of the study, and was preferred by six out of eight participants. Surprisingly, both users who preferred VoiceOver had higher average entry rates with Perkinput, but also higher corrected error rates. The errors seemed to frustrate these participants, although they managed to correct many of them quickly.

Performance on both methods in Study 1 was worse than we had expected. Participants had difficulty remembering a phrase, remembering how much of a phrase they had entered, and remembering how to spell certain words. Also, participants often touched the screen without intending to do so—a source of noise that we did not consider in our IFD model. Some of the challenges participants faced during the study were probably a result of age (average age was 55) and lack of experience with touch screens (all but one had little to no experience). We were concerned that the lower-than-expected performance on VoiceOver was a result of our prototype not fairly representing iPhone's VoiceOver. However, VoiceOver entry speeds in related work [3][16][17] are even lower than in our findings. Our average VoiceOver entry rate for Session 1 was 2.74 WPM, slightly higher than the rate found by Oliveira *et al*. of 2.11 WPM [17].

The case studies we conducted provide insight into the potential of expert performance with one- and two-handed Perkinput. Study 3 showed that two-handed text entry with Perkinput on a tablet is dramatically faster than one-handed entry on a phone for a participant who is proficient at typing on a Perkins Brailler. In light of the high speeds achieved, we believe people may prefer to use two-handed Perkinput on a phone-sized touch screen as well. Perkinput is designed to accept input from one or two hands on any device: a user simply sets six reference points (using both hands) or three reference points (using one hand) to indicate which encoding (six or three bits) the system should expect.

It would be interesting to compare performance of two-handed Perkinput to a Perkins Brailler keyboard[1]. As with QWERTY keyboards used by sighted people, it is likely that a "hard" Perkins keyboard would be more efficient than two-handed Perkinput, a "soft" counterpart. Efficient text entry on commodity touch screens offers advantages over entry on a hard keyboard, however, since people may not want to purchase and carry additional specialized hardware.

As previously mentioned, we evaluated Perkinput with users who were already familiar with Braille. A person who does not know Braille would have to memorize the Braille character encodings as she learns Perkinput, but learning how to read Braille tactually is not required. It is likely that using VoiceOver would be faster initially, especially if the user is familiar with a QWERTY keyboard. There is no reason, however, for VoiceOver to have an advantage to Perkinput after the user has memorized the Braille character encodings. There are no known estimates of the number of blind people who know Braille, but the National Federation of the Blind reports that a declining number of children are learning Braille [14]. One participant suggested Perkinput would be a useful tool for learning Braille without specialized devices.

Perkinput can potentially be used with other encodings that, while not conforming to an existing standard, are designed to optimize speed or minimize fatigue. For example, an encoding that assigns more ergonomic finger patterns to frequent characters may be faster after a user overcomes the initial task of memorizing each character's encoding.

10 Future Work

Since Perkinput is the first application of IFD, it can be improved in many ways. We plan to dynamically estimate the variance values for each of a user's fingers and use this information for the Maximum Likelihood detection. Our choice of tracking constants was somewhat arbitrary. Instead, we plan to learn the values of the adaptation coefficient and the correlation coefficient matrix using machine learning techniques. Additionally, both Maximum Likelihood and tracking can be modified to learn from user reinforcement. In text entry, the decoding algorithm receives negative reinforcement from the user when the backspace key deletes a user's input.

Perkinput can also be improved by adding support for contracted Braille, word-level automatic corrections, and better audio feedback. For example, when a user enters a space, the previous word should be spoken to remind the user of what she just entered and reveal possible mistakes.

[1] There are no known evaluations of text entry on Perkins Braillers or similar hard keyboards.

Finally, we plan to use Input Finger Detection for other applications, such as chording text entry that is not based on Braille.

11 Conclusion

We have described a new approach to touch screen input called Input Finger Detection (IFD) and its application for Braille-based text entry in Perkinput. The three studies we conducted demonstrate the potential of IFD. Our longitudinal evaluation showed that Perkinput outperforms iPhone's VoiceOver, reaching entry rates that are far beyond those of any nonvisual text entry method in the HCI literature, with low error rates. Enabling blind people to enter text into mobile devices efficiently and accurately represents an important step forward in making mobile computing fully accessible to all.

Acknowledgements

The authors thank Rhonda Nelson and all participants. This work was supported in part by Google, Intel, by the National Science Foundation under grant IIS-0952786, and by the US Department of Education under grant H327A100014. Any opinions, findings, conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect those of any supporter listed above.

References

[1] American Foundation for the Blind. "Braille Technology." http://www.afb.org/section.aspx?DocumentID=1282. Accessed on March 6, 2012.

[2] American Foundation for the Blind. "Cell Phone Access: The Current State of Cell Phone Accessibility." http://www.afb.org/afbpress/pub.asp?DocID=aw120602. Accessed on Sept. 16, 2011.

[3] Bonner, M., Brudvik, J., Abowd, G. Edwards, K. (2010). No-Look Notes: Accessible Eyes-Free Multi-Touch Text Entry. *IEEE Pervasive Computing* '10. Heidelberg: Springer, 409--426.

[4] Frey, B., Southern, C., Romero, M. (2011) BrailleTouch: Mobile Texting for the Visually impaired. Proc. HCII '11. Springer, 19-25.

[5] Gardner, F. M. (2005) Phaselock Techniques. New York: John Wiley & Sons.

[6] Higgins, J.J. and Tashtoush, S. (1994). An aligned rank transform test for interaction. *Nonlinear World 1* (2), 201-211.

[7] Kane, S.K., Bigham, J.P. and Wobbrock, J.O. (2008). Slide Rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques. In *Proc. ASSETS '08*. New York: ACM Press, 73-80.

[8] Kane, S. K., Wobbrock, J. O. and Ladner, R. E. (2011). Usable gestures for blind people: understanding preference and performance. *Proc. CHI '11*. ACM, New York, NY, USA, 413-422.

[9] Laird, N.M. and Ware, J.H. (1982). Random-effects models for longitudinal data. *Biometrics 38* (4), 963-974.

[10] Littell, R.C., Henry, P.R. and Ammerman, C.B. (1998). Statistical analysis of repeated measures data using SAS procedures. *Journal of Animal Science 76* (4), 1216-1231.

[11] Lyons, K., Starner, T., Plaisted, D., Fusia, J., Lyons, A., Drew, A. and Looney, E.W. (2004). Twiddler typing: One-handed chording text entry for mobile phones. *Proc. CHI '04*. Vienna, Austria (April 2004). New York: ACM Press, 671-678.

[12] MacKenzie, I.S. and Zhang, S.X. (1999). The design and evaluation of a high-performance soft keyboard. *Proc. CHI '99*. Pittsburgh, Pennsylvania (May 15-20, 1999). New York: ACM Press, 25-31.

[13] Mascetti, S., Bernareggi, C., and Belotti, M. (2011). TypeInBraille: a braille-based typing application for touchscreen devices. *Proc. ASSETS '11*. ACM, New York, NY, USA

[14] National Federation of the Blind. "How many children in America are not taught Braille?" http://www.nfb.org/nfb/Braille_Initiative.asp. Accessed on Sept. 16, 2011.

[15] National Federation of the Blind, "Usable consumer electronics." http://www.nfb.org/nfb/accessible_home_showcase.asp#Cell. Accessed on Sept. 16, 2011.

[16] Oliveira, J., Guerreiro, T., Nicolau., H, Jorge, J., and Gonçalves, D. (2011). BrailleType: Unleashing Braille over touch screen mobile phones. *INTERACT* '11. Heidelberg: Springer, 100-107.

[17] Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J., and Gonçalves, D. (2011). Blind people and mobile touch-based text-entry: acknowledging the need for different flavors. *Proc. ASSETS '11*. ACM, New York, NY, USA, 179-186.

[18] Royal National Institute of Blind People, "A Guide to Talking Mobile Phones and Mobile Phone Software." http://www.rnib.org.uk/livingwithsightloss/Documents/Mobile_phone_software_factsheet_PDF.pdf. Accessed on Sept. 16, 2011.

[19] Russel, S., and Norvig, P. (2011). "Learning Probabalistic Models." Artificial Intelligence: A Modern Approach. Prentice Hall, 802-829.

[20] Salter, K.C. and Fawcett, R.F. (1993). The ART test of interaction: A robust and powerful rank test of interaction in factorial models. *Communications in Statistics: Simulation and Computation 22* (1), 137-153.

[21] Sánchez, J. and Aguayo, F. (2006), Mobile messenger for the blind. Proc. of Universal Access in Ambient Intelligence Environments. Berlin: Springer, 369-385.

[22] Shannon, C.E. (1948). A mathematical theory of communication. The Bell System Technical Journal 27, 379-423,

[23] Shapiro, S.S. and Wilk, M.B. (1965). An analysis of variance test for normality (complete samples). *Biometrika 52* (3 & 4), 591-611.

[24] Soukoreff, R.W. and MacKenzie, I.S. (2003). Metrics for text entry research: An evaluation of MSD and KSPC, and a new unified error metric. *Proc. CHI '03*. New York: ACM Press, 113-120.

[25] Tinwala, H, and MacKenzie, I. S. (2009). Eyes-free text entry on a touchscreen phone. *Proc. TIC-STH '09*. New York: IEEE, . 83-88.

[26] Wobbrock, J.O. (2007). Measures of text entry performance. In *Text Entry Systems: Mobility, Accessibility, Universality*, I. S. MacKenzie and K. Tanaka-Ishii (eds.). San Francisco: Morgan Kaufmann, 47-74.

[27] Wobbrock, J.O., Chau, D.H. and Myers, B.A. (2007). An alternative to push, press, and tap-tap-tap: Gesturing on an isometric joystick for mobile phone text entry. *Proc. CHI '07*. New York: ACM Press, 667-676.

[28] Wobbrock, J.O., Findlater, L., Gergle, D. and Higgins, J.J. (2011). The Aligned Rank Transform for nonparametric factorial analyses using only ANOVA procedures. *Proc. CHI '11*. New York: ACM Press, 143-146.

[29] Wobbrock, J.O. and Myers, B.A. (2006). Analyzing the input stream for character-level errors in unconstrained text entry evaluations. ACM Transactions on Computer-Human Interaction 13 (4), pp. 458-489

[30] Yfantidis, G. and Evreinov, G. Adaptive blind interaction technique for touchscreens, Universal Access in the Information Society, 4, 2006, 328-337.