Eyes-Free Input on Mobile Devices

Shiri Azenkot

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Richard Ladner, Chair

Jacob O. Wobbrock, Char

Alan Borning

Program Authorized to Offer Degree:

Computer Science and Engineering

University of Washington

Abstract

Eyes-Free Input on Mobile Devices

Shiri Azenkot

Chairs of the Supervisory Committee:
Professor Richard Ladner
Computer Science and Engineering
Associate Professor Jacob O. Wobbrock
The Information School

I present new methods and studies that aim to improve eyes-free data entry for blind mobile device users. Currently, mobile devices are generally accessible to blind people, but text entry is almost prohibitively slow. Studies show that blind people enter text on an iPhone at a rate of just 4 words per minute.

I describe Perkinput, a chording text entry method where users touch the screen with one to three fingers at a time in patterns based on Braille. Instead of soft keys, Perkinput uses concepts from signal detection theory to determine the user's input. Based on Perkinput, I developed PassChords, a touchscreen authentication method that has no audio feedback. Unlike current eyes-free input methods, PassChords doesn't echo a user's input, so it won't broadcast the user's

password for others to hear. Finally, I will discuss another modality for eyes-free input: speech. I conducted a survey and a study to determine the patterns and challenges of the use of speech input for composing paragraphs on mobile devices. I conclude by presenting SpeakNSwipe, an eyes-free speech input method that facilitates correction of speech recognition errors.

Acknowledgements

I'd like to thank everyone who supported me and contributed to this dissertation. First and foremost, I thank my advisors, Richard Ladner and Jacob Wobbrock, for their guidance and support. Also, thanks to Alan Borning who has been an excellent mentor throughout my years at UW and the rest of my committee, James Landay and Gina-Anne Levow. I couldn't have done this work without my smart and creative collaborators, including Kyle Rector, Cynthia Bennett, Sanjana Prasain, and all the students who made the HCI lab an inspiring and fun place to work.

Thanks also to my wonderful friends, including Lydia Chilton, Carin Fishel, Carrie Tanner, Sharon Greenblum, Emily Kusak, and Noa Tal, who have supported me both personally and professionally.

Last but not least, I thank my family, who has been a powerful and consistent source of unconditional love and support. I love you all!

Table of Contents

List of Figures

Chapter 1. Introduction

1.1. Motivation

Mobile computing devices are ubiquitous. Over the last decade, cell phones, smartphones, tablets, and other mobile devices have become an essential part of a connected and productive life. The Pew Research Center reports that in January 2014, 90 percent of American adults owned a cell phone and 58 percent owned a smartphone [76]. Interestingly, 42 percent of American adults owned a tablet computer, a newer mobile device than a smartphone [76]. Globally, more people have a smartphone than a personal computer [34]. People use their mobile devices for personal and professional communication, browsing the Internet, getting directions, and a variety of other purposes. The most popular mobile platforms today are Google's Android and Apple's iOS [34]. As new and more powerful mobile devices emerge in the marketplace, people will spend more time performing a wider array of tasks on their various mobile devices.

Mobile devices have immense impact on the lives of people with disabilities in particular. For example, mobile devices enable blind people to access a wealth of visual information in their environment. A low-vision person stands on a street corner and wants to read the nearby street sign to determine where she is. She sees the sign but cannot read the text on it, so she takes a photo of the sign with her smartphone camera and magnifies the image on the screen. She is then able to read the text of the sign. Mobile devices have already become essential accessibility tools. Blind people benefit from standard applications such to access their email, calendars, and public transit information. Moreover, there are many "accessibility" applications that give blind

people information about their environment, including the VizWiz question-answering application [16], LookTel's Money Reader application [59], and Sendero's LookAround GPS-based application [84]. As sensor technology and computer vision continue to improve, many more accessibility applications will emerge.

In order for blind people to benefit from mobile device applications, the mobile devices themselves must be usable. People must be able to input information into and perceive output from the device efficiently and accurately. Currently, both Apple and Google devices have built-in screen readers that make them generally accessible to blind people. Apple released VoiceOver , the first commercial touchscreen screen reader, in 2009 [2]. VoiceOver has since become the *de facto* standard in eyes-free touchscreen interaction. It allows users to explore the screen in a "risk-free" manner [47]. As the user touches interface elements on the screen, VoiceOver speaks the elements' labels. To select an element, the user performs a second gesture, like a double tap. Google's screen reader TalkBack [36] enables a similar interaction.

Blind people can use smartphones with VoiceOver and TalkBack to some degree, but their experience is far from equal to that of sighted people. In particular, entering text with a screen reader is almost prohibitively slow. Researchers found that the text entry rate with a VoiceOver-like keyboard was 0.66 words per minute in one study [17] and 2.11 WPM in another [71][1]. In contrast, studies showed that sighted people entered text at a rate of over 20 WPM in one study [23] and over 30 WPM in another [12]. Some blind people prefer to use external hardware

[1] It is unclear why there is such a large discrepancy between the results of these two studies. Perhaps it is due to participants' varying levels of experience with touchscreens and technology in general.

keyboards like a small QWERTY keyboard [26] or a keyboard on a Braille device [18]. There are advantages to entering text directly on the device, however. People would not need to purchase, carry, or reach for an extra device. Text entry is a fundamental task in human-computer interaction, so it is critical for blind people to be able to enter text efficiently and accurately to use a device effectively.

In addition to being extremely slow, entering text with VoiceOver poses privacy and security concerns. First, VoiceOver raises the threat of eavesdropping on users' passwords. As the user enters a password, VoiceOver speaks the key labels aloud, allowing bystanders to hear a user's password. This threat is particularly relevant for mobile devices that tend to be used on-the-go and in public. Second, because VoiceOver text entry is so slow, users are likely to create short, easy-to-enter passwords or forgo optional passwords altogether. For example, it is likely that blind users will not use the optional passcode lock that protects a phone from unauthorized user access. It is thus also important for blind users to have secure and usable authentication methods to enable equal access in terms of privacy and security.

In this dissertation, I describe new eyes-free input methods on mobile devices that aim to narrow the usability gap between blind and sighted people. I explore input using onscreen gestures and speech. Both modalities have distinct advantages: speech in itself is more natural and faster than any commonly used keyboard technique, while gestures are more private and, given the state of the art in speech recognition engines, more accurate, especially in noisy environments. Speaking can also be inappropriate in certain social situations, such as during a meeting or lecture. In general, providing multiple methods for text entry is important because users will have varying preferences and abilities.

The work in this dissertation aims to support the following thesis statement:

> *Gesture-based input methods that use simple multi-touch taps, and speech-based input methods that facilitate error detection and correction, can both enable blind people to enter text more effectively on touchscreens than the de facto standard methods.*

Gesture-based methods are discussed in Chapters 3–5, and speech-based methods are discussed in chapters 6–7.

1.2. Contributions

Below I summarize the contributions and key results from each chapter. The contributions include four inventions: Perkinput, PassChords, DigiTaps, and SpeakNSwipe; and six major studies: one study to evaluate each invention, a study that explores security and privacy issues related to mobile access technology, and a study that explores the use of speech input by blind people on mobile devices. When discussing the work, I use the first-person plural voice ("we") since all of these projects were conducted with the help and support of collaborators.

Chapter 3. We contribute a new multi-touch technique for eyes-free input called *Input Finger Detection* (IFD), and a text entry method based on IFD called *Perkinput*. In IFD, a user inputs a signal into a touchscreen by touching the screen with several fingers, where each finger represents one bit, either touching the screen or not. There are no soft keys in IFD. Instead, the user sets reference points anywhere on the screen to indicate approximate finger positions and continues to tap the screen in roughly the same locations. The system detects which fingers touch the screen in subsequent touches using the recent reference points and maximum likelihood and tracking algorithms.

The Perkinput text entry method allows users to enter letters using multi-touch taps in patterns based on the 6-bit Braille character encoding. We evaluated Perkinput with a longitudinal study with eight blind people, comparing Perkinput to VoiceOver. Study results show that 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.

This work was originally published in [11].

Chapter 4. We present two contributions: (1) a study of security risks for blind mobile device users, and (2) *PassChords*, a new authentication technique for touch screens that is accessible, fast, and robust to aural eavesdropping. We interviewed 13 blind smartphone users to discover their attitudes towards and specific behavior patterns affecting security risks. We found that most participants were not concerned with security issues, and none used optional authentication mechanisms to protect their information.

Following our interview study, we sought to improve mobile device security. We describe a new accessible and secure authentication method called PassChords. PassChords are based on IFD and consist of several multi-point touches, defined by the set of fingers touching the screen. PassChords have no audio feedback, so they are robust to aural eavesdropping. In a study with 16 blind people, we found that PassChord entry was nearly three times as fast as entry of accessible personal identification numbers (PINs) and had about the same authentication failure rate.

The work presented in Chapter 4 was originally published in [9].

Chapter 5. We contribute the design and evaluation of *DigiTaps*, a novel method for nonvisual number input that uses natural prefix-free codes.

To enter a number with DigiTaps, a user taps or swipes the screen with one or more fingers (*e.g.*, a two-finger tap, a one-finger swipe). The DigiTaps gestures "feel" different, so people can discern whether they touched the screen with one finger or two. A user performs one to three gestures to enter a digit, according to a code that is derived from the semantics of the digits.

We evaluated DigiTaps as an eyes-free, minimal-audio input method in a longitudinal study with 16 people (six blind, ten sighted). Participants entered sets of digits with and without audio feedback. We found that removing audio feedback produced significantly higher entry rates, but also higher error rates. Participants entered digits at a mean rate of 0.87 characters per second (CPS) (*SD* = 0.32) with no audio feedback, and a mean uncorrected error rate of 5.6% (*SD* = 14.8). This was much faster than VoiceOver, as we found in prior work (Chapter 4) that participants entered four-digit PINs in 7.5 seconds (about 0.53 CPS).

The DigiTaps code was first presented by Ruamviboonsuk *et al.* [79] and then it was expanded and evaluated by Azenkot *et al*. [7].

Chapter 6. Our main contribution in this chapter is findings from a survey and a study that explore the patterns and challenges of the user of speech input by blind people on mobile devices. We conducted a survey with 169 people (105 sighted and 65 blind and low-vision) to learn how often people use speech input, what they use it for, and how much they like it. We then conducted a laboratory study with 8 blind people to observe how blind people used speech

to compose paragraphs. We wanted to discover what techniques people used to review and edit the speech recognizer's output and how effective these techniques were. This work aims to establish a foundation for improving eyes-free speech input systems.

In our survey, we found that blind people used speech for input more frequently and for longer messages than sighted people. Blind people were also more satisfied with speech than sighted people, probably because the comparative advantage of speech to keyboard input was far greater for them than for sighted people. Our laboratory study showed that speech was nearly five times as fast as the on-screen keyboard, but editing recognition errors was frustrating. Participants spent an average of 80.3% of their time reviewing and editing their text. Most edits were performed using the BACKSPACE key and reentering characters with the keyboard. Six out of eight participants in the study preferred speech to keyboard entry.

This work was originally published in [8].

Chapter 7. We contribute *SpeakNSwipe*, an eyes-free dictation system that facilitates error correction and requires only five sensors and a microphone for input.

In this chapter we propose *eyes-free dictation* (EFD), methods of entering and editing text on a variety of computing devices without visual feedback. EFD would largely expand the range of tasks we can perform with a device, since it would allow for entering text of any length. Surprisingly, there are no eyes-free dictation techniques in the human-computer interaction literature; people instead have focused on voice commands or speech recognition systems that require visual feedback for editing.

We present SpeakNSwipe, an eyes-free dictation system that leverages a commercial speech recognizer's output of alternative recognition results. SpeakNSwipe allows the user to speak, review, and edit text input in an eyes-free interface with a small set of simple gestures that can be performed with just five sensors (*e.g.*, buttons, gestures on a touch screen) and a microphone. We evaluated SpeakNSwipe in a study with nine sighted people along with two control methods. We found that participants found SpeakNSwipe to be less frustrating and more satisfying that both of the other methods.

Work on SpeakNSwipe has not yet been published.

Chapter 2.
Related Work

At the time the work in this dissertation was conducted, from 2009 to 2014, mobile devices commonly had multi-touch screens with only a few physical keys. So our work extends prior work on touchscreen interaction and accessibility. In this section, I present work on eyes-free touchscreen interaction in general, eyes-free touchscreen text entry methods using onscreen gestures and speech, and also eyes-free authentication methods.

2.1. Eyes-free Interaction on Touchscreens

Touchscreens technology was available in some form since at least the mid 1960's [14] and by the 1990's, touchscreens became more common in public kiosks and even some early PDA's [21]. Touchscreen interaction has mainly involved soft keys [66], fixed regions on the screen that can be activated with a touch, simulating hardware buttons. Unlike hardware keys, however, soft keys don't provide tactile feedback, making standard touchscreens inaccessible to blind people. Researchers have been concerned with the inaccessibility of touchscreens for a couple of decades [22,35]. Early work on touchscreen accessibility proposed making touchscreens accessible by adding speech output and a small number of hardware buttons [55,95]. In Vanderheiden's Talking Fingertip Technique [95], users explored the screen of a public kiosk by touching it, as the system spoke descriptions of the items being touched. To activate an item, the user pressed a hardware button. The Talking Tactile Tablet [54], proposed in 2003, used touch input with a custom overlay and speech output.

In 2008, Kane *et al.* presented the seminal Slide Rule [47], a set of interaction techniques that made multi-touch screens accessible with no additional hardware keys or overlays. Several techniques from Slide Rule were then adapted by Apple in the VoiceOver screen reader. While Slide Rule provided a special interface that was optimized for nonvisual use, VoiceOver incorporate ideas from Slide Rule that would enable access to the iPhone's visual interface. The approach of our work is similar to that of Slide Rule: we focus on input methods that are optimized for nonvisual use and require no extra hardware. Kane *et al.* later presented a study exploring blind user's preference and performance in making onscreen gestures [49].

Researchers have proposed nonvisual interaction techniques that were not specifically targeted at blind users as well. Instead, these techniques aim to make mobile devices usable in situations where it may be inconvenient or not possible for people to look at the screen, like while walking. Such approaches include earPods by Zhao *et al.* [106] and BlindSight by Lee and Baudisch [56]. These systems require hardware modifications to mainstream devices, however, so they differ from our work. The interest in nonvisual access for sighted people highlights the potential use of our work by sighted users as well as blind users.

2.2. Eyes-free Text Entry Using Onscreen Gestures

The need for efficient eyes-free text entry is highlighted by the extensive number of methods proposed in the academic literature. Developers have also built commercial applications with alternate keyboards that are adopted by some blind people. When designing applications, researchers considered the trade-offs between learnability, speed, and accuracy. Most researchers evaluated speed and accuracy empirically in laboratory studies, but their study design methods

and the characteristics of their participants differed greatly, so it is difficult to compare performance results among methods.

We divide existing input methods into two categories: tap-based and stroke-based methods. Perkinput and DigiTaps are motivated by the prevalence of soft keys in tap-based techniques, that have no tactile delineations and are not optimal for nonvisual interaction. Meanwhile, techniques that use simple strokes often require multiple strokes to enter each character. As explained in Chapters 3 and 5, Perkinput and DigiTaps use novel entry paradigms where a user performs multi-touch taps anywhere on the screen with no anchored targets. Our work also includes rigorous longitudinal studies to evaluate our Perkinput and DigiTaps, while most of the related methods lack such studies.

2.2.1. Tap-based Input Methods

The *de-facto* standard nonvisual text entry method is the Standard Typing mode with VoiceOver, which involves taps on a QWERTY keyboard. Keys can be selected with a split-tap or a double-tap, or an alternate technique called Touch Typing [108]. To enter text in the Touch Typing mode, a user selects a key by simply lifting her finger from the screen. Studies show that double-tap and split-tap text entry with VoiceOver is almost prohibitively slow; Bonner *et al.* [17] found that the mean text entry rate on an iPhone with VoiceOver was only 0.66 words per minute (WPM) and Oliveira *et al.* [71,72] report a mean speed of 2.1 WPM. To our knowledge, there are no studies that evaluate the speed and accuracy of Touch Typing, but anecdotal evidence suggests that it is, at best, marginally faster than Standard Typing mode.

One approach that researchers have taken to improve text entry efficiency is to enlarge the size of the keys. Sanchez and Aguayo [81] presented a text entry method in 2006 that emulated multi-tap with nine soft keys. They explain that blind users can find the keys using the phone's edges and corners as guides. In 2010, Bonner *et al.* proposed No-Look Notes [17], where several triangular keys that meet at the center of the screen represent groups of characters. The user selects the correct group and is then presented with several keys representing each character from that group. Unlike Sanchez and Aguayo's method, No-Look Notes used risk-free exploration as in VoiceOver. Bonner *et al.* evaluated No-Look Notes against VoiceOver with blind people, and found that entry rates were 1.32 words per minute (WPM) and 0.66 WPM, respectively.

Researchers created several text entry methods based on Braille character encodings. BrailleType, by Oliveira *et al.* [71], divides the screen into six large keys representing the six dots in a Braille cell. Users touch the keys corresponding to the dots in the Braille character they want to enter. Oliveira *et al.* found that with an entry rate of 1.45 WPM, BrailleType was significantly slower than VoiceOver, albeit more accurate. The similarly-named TypeInBraille [63] is another Braille-based entry method that uses three simple gestures to input a character based on its Braille encoding. There is no evaluation of TypeInBraille.

Frey *et al.* [32] described BrailleTouch, a text entry method that most resembles Perkinput. The two methods were developed around the same time. Like Perkinput, BrailleTouch users tap the screen with fingers corresponding to the dots in Braille characters. However, the six Braille dots in BrailleTouch are represented by fixed regions on the screen that do not adapt to the user's input. An evaluation of BrailleTouch [88] found that expert Braille typists entered text at a rate of 23.1 WPM with a Total Error Rate of 14.8%. Non-expert Braille typists entered text at a far

lower rate with significantly higher error. It is important to note that, unlike in the Perkinput and DigiTaps evaluations, users of BrailleTouch were unable to correct errors. Correcting errors takes time, so BrailleTouch entry rates would likely be lower in real use, when users corrected their mistakes.

In addition to research prototypes, people have developed commercial applications that have alternative tap-based entry methods. Anecdotally, it seems that these applications are adopted to some degree by blind people. The BrailleTouch application [19] is based on the BrailleTouch research prototype. MBraille [65] is similar to BrailleTouch and BraillePad [39] is similar to BrailleType. Fleksy [30] has a QWERTY keyboard with an aggressive language model, so users hit in the rough vicinity of a key to enter a character and then choose from a set of possible words to resolve ambiguity. There are no rigorous studies evaluating these approaches.

2.2.2. Stroke-based Input Methods

Before multi-touch screens were standard on mobile devices, several eyes-free text entry methods were proposed that are based on simple strokes on a screen. In 2005, Yfantidis and Evreinov [105] presented a method that involved a 3-layered pie menu. Users moved from one layer to the next by dwelling and selected a character by lifting their finger. Similarly, Google' s early Android phones included a text entry method using pie menus in the Eyes-Free Shell [38], a custom nonvisual interface. The Eyes-Free Shell text entry method. Instead of dwelling to move to the next level, the pie menu included items that loaded the next layer of the menu. One advantage of these methods it that they can be used with just one hand, where the user holds the phone and touches the screen with her thumb. There are no studies with blind people for either method that evaluate performance.

Similar to pie menus, Oliveira *et al.*'s NavTouch [72] uses simple directional strokes. In NavTouch, users "navigate" through a 2-dimensional arrangement of the alphabet until the desired letter is selected. In later work, Oliveira *et al.* evaluate NavTouch against VoiceOver in a single-session study, finding that NavTouch was slower than VoiceOver, with an entry rate of just 1.72 WPM, and an error rate of over 10%.

Researchers have also proposed eyes-free text entry methods for sighted people that may be challenging for blind users. Tinwala and MacKenzie's eyes-free text entry method [94], based on the Graffiti alphabet [29]. This method was designed for sighted users who know how to handwrite characters and can easily learn the Graffiti alphabet. Blind users, especially those who were blind since birth, do not necessarily know how to hand write characters. There has been some work on teaching blind people how to perform gestures on touchscreens [70], but it is still an important open problem. Jain and Balakrishnan [43] designed a text entry method based on bezel swipes and evaluated its performance with sighted users who split their visual attention between entering text and reading numbers from another screen. The bezel swipe method has a high learning curve and it is unknown how well blind users would perform with this method.

2.3. Eyes-Free Text Entry Using Speech

While there has been a wealth of research on gesture-based text entry methods, there is a paucity of research on speech-based input methods. To our knowledge, we are the first to study eyes-free speech input in the human-computer interaction literature. In the space of eyes-free interaction, speech has mostly been studied as a form of nonvisual output (*e.g.*, [77,89]) rather than nonvisual input. There has been some work on hands-free dictation, both for people with motor impairments [83], and for the general population [40,50].

Prior work on speech input interaction focuses on error correction, the "Achilles Heel of speech technology" [74]. Desktop dictation systems such as Dragon Naturally Speaking [69], which gained popularity in the late 1990's, use speech commands for cursor navigation and error correction. Users speak commands such as "move left" and "undo" to edit text or reposition the cursor. Karat *et al.* [50] found that novice users entered text at a rate of only 13.6 WPM with a commercial desktop dictation system and 32.5 WPM with a keyboard and mouse. This striking discrepancy was due to (1) cascades of errors triggered by a user's correction and (2) spiral-depth, a user's repeated attempts to speak a word that is not correctly recognized.

Some work has aimed to alleviate the difficulty of error correction through touch or stylus input. Suhm *et al*. [90] present a system where users touch the word they want to correct, eliminating speech-based navigation. Their system also supports small gestures such as striking out a word as shortcuts. Martin and Welch [62] enable users to correct errors of preliminary results with a mouse click. These systems make error correction more efficient but have high visual demands and are not appropriate for blind users.

There is little recent research on speech-based input systems. Voice Typing, introduced by Kumar *et al.* in 2012 [52], displays recognized text as the user dictates short phrases. Users can correct recognition errors with a marking menu. Kumar *et al.* found that correcting errors with Voice Typing required less effort than correcting errors with the iPhone's dictation model.

The iOS and Android dictation systems resemble the literature described above. Dictation on the iPhone follows an open-loop interaction model, where the recognizer outputs text only after the user completes the dictation. Android, in contrast, currently has incremental speech recognition, displaying recognized text as the user speaks. With VoiceOver, iOS dictation seems to be

accessible to blind people, but it is unclear how incremental recognition affects accessibility on Android, especially since Android is not as generally accessible as iOS.

2.4. Eyes-Free Authentication on Mobile Devices

Our work on PassChords (Chapter 4) is motivated by the security and privacy concerns that arise from existing eyes-free interaction techniques. Related work falls into two categories: security issues for blind people and mobile authentication techniques for the general population. Our work is the first, to our knowledge, to focus on security issues for blind mobile device users and develop and evaluate an accessible touch screen authentication method.

Kane *et al.* [48] discussed patterns and challenges of mobile device use for people with visual impairments and briefly mentioned users' privacy concerns. The authors did not delve into potential security problems. The study was conducted in 2009, before the iPhone introduced VoiceOver. Since blind people now use touch screen devices, new security challenges have arisen.

Some work has been done in the area of accessible security, but, to our knowledge, none has focused on mobile devices. Kuber and Sharma proposed accessible authentication methods for desktop computers using a tactile mouse [51]. Several papers discussed the accessibility of CAPTCHA's [15,41,86] which are used to verify human users, but do not protect against unauthorized access. Our work concerns user authentication with accessible and secure password techniques.

The security community has widely acknowledged the inadequacy of alphanumeric passwords, and alternative authentication methods have been proposed. Graphical passwords [75] have been

studied extensively over the past decade, including techniques that require users to select a sequence of photos that are displayed on the screen [27], to select a sequence of points in displayed images [96], or to draw a "secret" shape or design on a grid [46,73]. These techniques are generally inaccessible to blind people.

One potentially accessible technique is TapSongs [102], a rhythm-based authentication method for devices with a single binary sensor (*e.g.*, button). (TapSongs were later utilized and extended by Nokia researchers in their RhythmLink system; they named such rhythm-based passwords "tapwords" [57].) A difference between TapSongs and PassChords is that the duration of a TapSong was about 6-8 seconds, while PassChords tended to be less than 4 seconds long. Also, since it is not clear what the entropy of TapSongs is, it is difficult to evaluate their security strength, although the Nokia researchers made some attempt to do so.

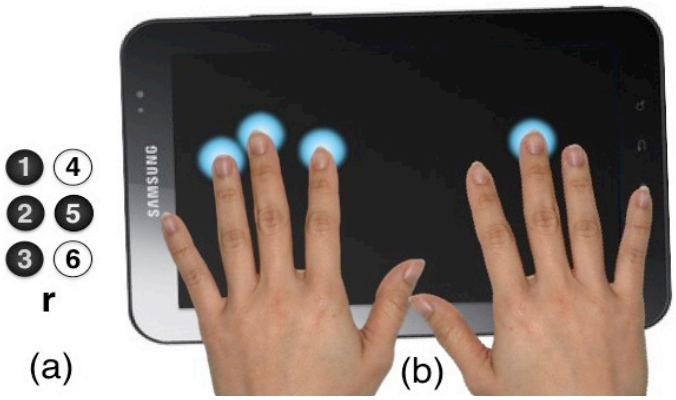
Biometric authentication offers another potentially accessible alternative to graphical or alphanumeric passwords [104]. Robust biometric techniques (*e.g.*, iris scans, hand and fingerprint recognition) often require special hardware, that has not been adopted on mainstream mobile devices. Our approach is lightweight and requires only a touch surface.

Chapter 3.
Perkinput: Eyes-Free Touchscreen Text Entry based on Braille

3.1. Motivation

We began work on a new eyes-free text entry method for touchscreens in 2010, less than a year after Apple introduced the VoiceOver screen reader. By then, many blind people were already using iPhones but it was clear that text entry was a barrier. While we were developing Perkinput, several other eyes-free text entry methods were published in the literature [17,32,71,72], but they showed little to no performance improvements over VoiceOver (see Chapter 2 for a thorough review of related work).

In this chapter, we describe Input Finger Detection (IFD), a new approach to touchscreen input. Then we describe the Perkinput text entry method (Figure 1) which uses IFD and a longitudinal study with 8 blind users where we evaluate Perkinput compared with text entry with VoiceOver.



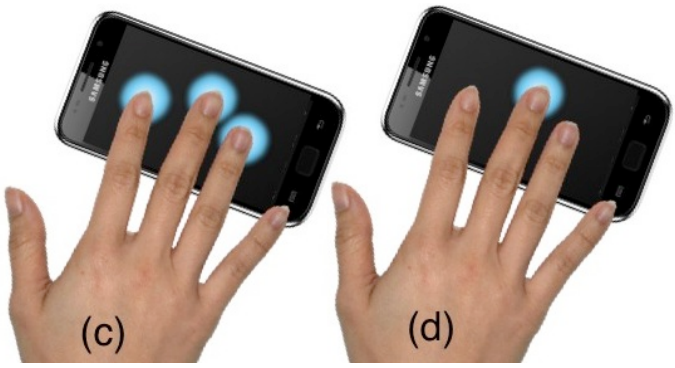


Figure 1. A user enters a character with Perkinput using one or two hands. The (a) Braille encoding for 'r' is the binary string "111010." It is entered with two hands simultaneously as shown in (b), or with one hand by touching the phone twice. The first touch (c) inputs bits 1-3 and the second touch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

3.2. Input Finger Detection (IFD)

We propose *Input Finger Detection* (IFD), a multi-touch input technique where users enter information into a device with multi-touch taps. We model touchscreen input as transmission of information over a noisy channel [22], from the human to the device. In 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 *ith* finger contacts the screen for a given touch, the *ith* bit in the binary sequence is 1. Otherwise, the *ith* 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.

3.2.1. 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.

$$P(d|h_\theta) = \frac{1}{(2\pi)^{1/2}\,\sigma}\exp\left[-\frac{(d-\theta)^2}{2\sigma^2}\right] \tag{3.1}$$

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 signal estimate for Figure is thus the binary sequence "100," since only finger 1 touches the screen.

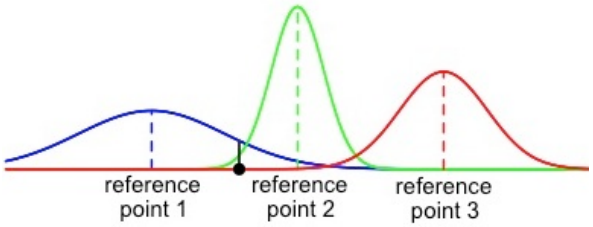


Figure 2. The curves show the likelihood distributions of a touch point input by each finger. The black dot marks the location of a touch point. The height of the curves above the touch point show that it was most likely input by finger 1, even though the point is closer to reference point 2 than to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

In the case where the observed data has more than one point, the MLE 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 3 reference points and 2 touch points. The MLE is {1,2}, which maximizes the sum of the likelihood of the reference points given their respective input points. The binary sequence is "110."

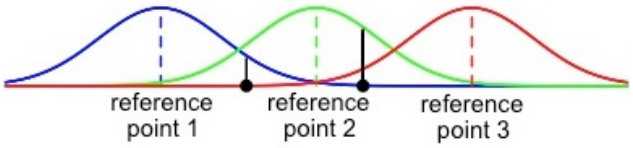


Figure 3. The black dots marks the location of 2 touch points and the black lines show the probabilities that the first point was input by finger 1 and the second point was input by finger 2, which is highest than any other combination.

As Figure 2 and Figure 3 show, the variance of touches around a reference point is an important factor in finger location inference. Variances can be determined per finger per user, by averaging

the distances between input points and their corresponding reference points. Adjustments can be made dynamically as well, by updating the variance with new input data as the user enters text.

3.2.2. Updating Reference Points

While our maximum likelihood estimation is based on the assumption that a user keeps her fingers close to their reference points, we realize a user is likely to move her hand as she enters text. To account for this movement, we *track* the reference points by adjusting them slightly with every input. This technique is based on a Phase-Locked Loop [33], commonly used in electronic applications.

After we determine which fingers touched the screen, we move each reference point toward 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. Since we aim to correct for drift of the entire hand, we assume that the drift of individual fingers is correlated. So 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:

$$R_{n+1} = k \cdot C \cdot E_n + R_n \tag{3.2}$$

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

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

The scalar constant k is the *adaptation coefficient* that indicates the extent to which errors are used to shift reference points. Small values of k minimize 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:

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

When using this value of $\boldsymbol{C}$, 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.

3.2.3. Advantages for Nonvisual Input

We discuss advantages of finger location inference 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 variance, 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 and user abilities.

Finger location inference 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 [17,32,71] to avoid or quicken the select-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 [63,72].

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 maximum likelihood 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 usually require vision. Also, complex gestures [49,94] are difficult for blind people to input. Furthermore, finger location inference, 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 finger location inference for Braille-based text entry for blind people, showing how the theory we presented works in practice.

3.3. The Perkinput Text Entry Method

We begin by giving an overview of Braille and describe how we used finger location inference 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.

3.3.1. Braille and the Perkins Brailler

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 down on the keys that correspond to the dots raised in the character.

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

3.3.2. Finger Location Inference in Perkinput

Perkinput uses finger location inference with the Braille character encoding. This encoding offers an advantage for blind people who read Braille and type on a Perkins Brailler. 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. Also, 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.

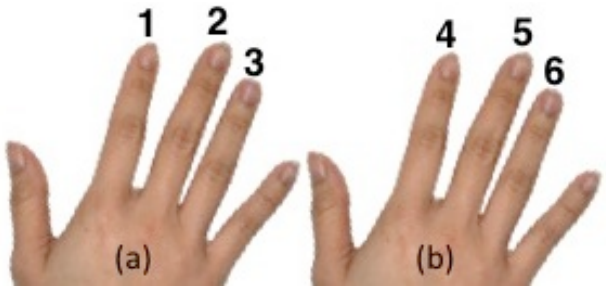


Figure 5. The numbers above the fingers show which fingers correspond to which Braille dots for (a) the first and (b) the second touch inputs with one-handed Perkinput.

A user registers reference points with a "long press." If 6 fingers are registered, Perkinput uses a 6-bit encoding. If only 3 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 finger location inference, 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 0. This simplifies the problem of MLE to finding the nearest reference point(s) in 2 dimensions. In the future, we plan to dynamically track a user's finger variance. We derived the following tracking constants through experimentation:

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

While 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 2-finger swipe in any direction represents a space and a 3-finger swipe in any direction represents a backspace. This involves placing 2 or 3 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.

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.

3.3.3. Perkinput for Two-Devices

As touch screen phones become more common and less expensive, it is interesting to explore interaction techniques using two phones at once. A Perkinput user may want to borrow a friend's phone for a few minutes or buy a second device for daily use. Even two touch screen devices are less expensive and easier to carry than many Braille devices.

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

The non-primary phone sends the primary phone information about each multi-point touch it receives, which the primary phone then decodes along with its own input to determine what character was entered. The user must register reference points on each phone. The primary phone 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 phone 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 phone—there is no time limit. If the user touches one phone twice before touching the second phone, the first input on the former phone is disregarded. Thus, to indicate a "blank" column, the user must input a swipe as she does on standard Perkinput.

3.4. Study 1: Perkinput vs. VoiceOver

To evaluate Perkinput, we sought to compare it with the current state of the art. We chose iPhone's VoiceOver text entry method as the second method, since it seems to be the most popular touch screen method among blind people. We conducted a longitudinal study with 8 blind participants to evaluate performance over time.

3.4.1. Method

Participants

We conducted the study with 8 blind participants (5 female, 3 male), with an average age of 55. None were able to visually identify keys on a phone. Almost all participants were proficient at reading Braille and typing with a Perkins Brailler. All were proficient at typing on a QWERTY keyboard. Only one participant had experience with VoiceOver—all other participants had little to no experience with touch screen devices in general.

Apparatus

We developed a prototype of 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).

Our prototype did not include any contractions (Grade 2 Braille) to maintain a fair comparison with VoiceOver. The same number of characters must 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.

Procedure

The study included 8 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 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 [60,100] 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. The prototypes spoke a phrase and we asked participants to repeat the phrase verbally before beginning to enter it. This helped ensure that participants understood and remembered the phrase.

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 (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 [58] 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 [53]. 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 [100] 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 [85]), ruling out the use of parametric analyses. Therefore, we used the nonparametric *Aligned Rank Transform* procedure [80,99] which enables the use of ANOVA after alignment and ranking, and which maintains the integrity of interaction effects. The same statistical model was used on 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.

3.4.2. Results

Words per Minute (WPM)

Average entry speed over all sessions for 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.

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.

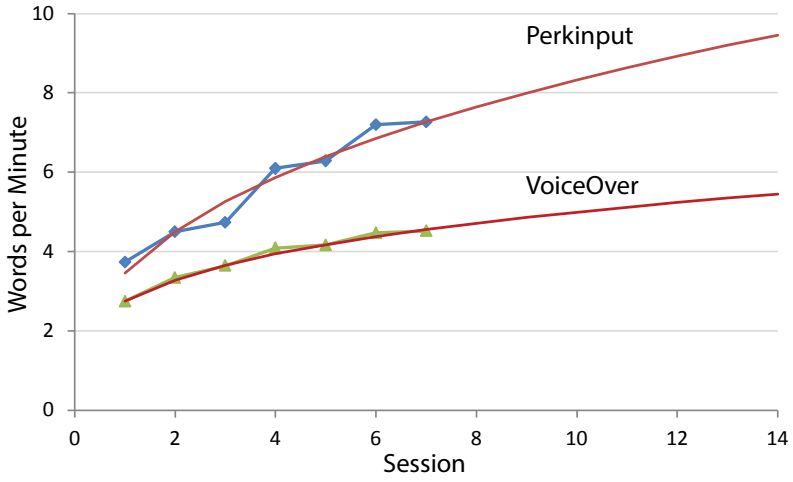


Figure 7. Entry rates as WPM over sessions 1-7, and extrapolated to session 14, for each input method. Higher is better.

It is customary (see, *e.g.*, [61,98]) 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:

Perkinput: $y = 3.4589x^{0.3812}$, $R^2 = .9546$;

VoiceOver: $y = 2.7450x^{0.2600}$, $R^2 = .9893$.

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.

Error Rates

We calculated both uncorrected and corrected error rates [100]. 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 [101]. Thus, uncorrected errors are of chief importance. Figure 8 shows uncorrected error rates by method over sessions.

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.*).

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.*).

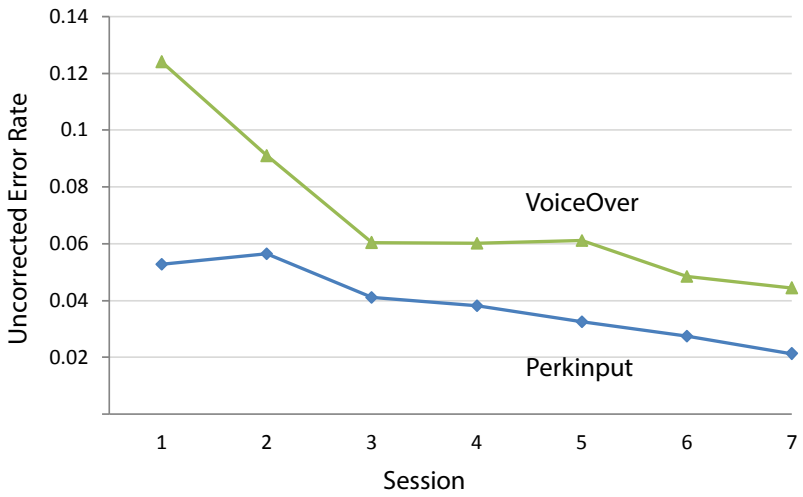


Figure 8. Uncorrected error rates over sessions 1-7 for each input method. Lower is better.

Subjective Preferences

At the end of Study 1, 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 ($\chi^2_{(1,N=8)} = 2.00$, $p = .16$).

3.5. Study 2: Expert Performance With Perkinput

In Study 2, we sought to assess expert performance on Perkinput after additional hours of practice.

3.5.1. Method

We conducted a case study with one participant (female, age 59) from Study 1. We will refer to this participant as P5. P5 is proficient in Braille but has little experience with a touch screen. In Study 1, P5 achieved the highest text entry rates on Perkinput. We conducted 6 additional sessions with P5, each consisting of 30 minutes of entering phrases with Perkinput. We used a similar procedure to Study 1, although we did not test VoiceOver in Study 2.

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 3-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.

3.5.2. 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 at 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 Perkinput usage.

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

Figure 10 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: $y = 3.1332x^{0.6348}$, $R^2 = .8435$

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 can enable rapid enough error correction so as to achieve noteworthy speeds with few uncorrected errors in the end.

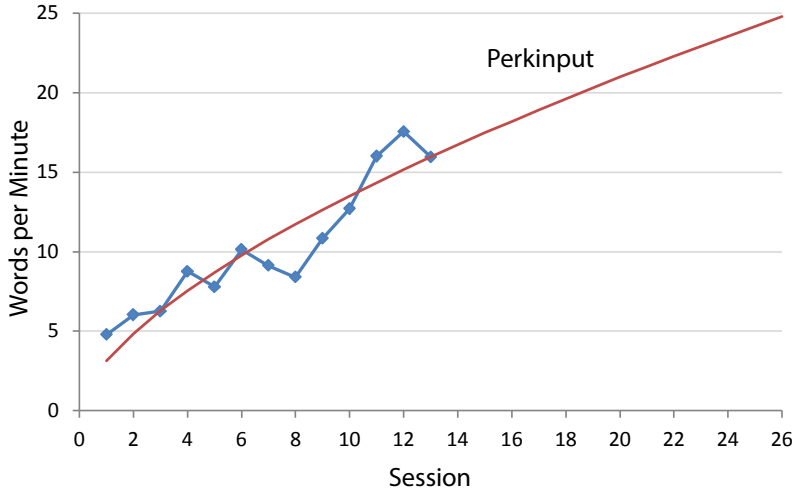


Figure 9. Entry rates for P5 with Perkinput over sessions 1-13, and extrapolated to session 26. Higher is better. Note that unlike Figure 7, the Y-axis here reaches 25 WPM, not just 10 WPM.

3.6. Study 3: Two-Handed Version of Perkinput

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

3.6.1. Method

We conducted a case study with P5 who already had 6.5 hours of practice with standard Perkinput. The procedure was similar to the procedure for Study 1, but Study 3 only included 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.

3.6.2. 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 ($F_{2,222} = 171.92$, $p < .0001$).

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 standard Perkinput were taken from P5's fastest overall session.

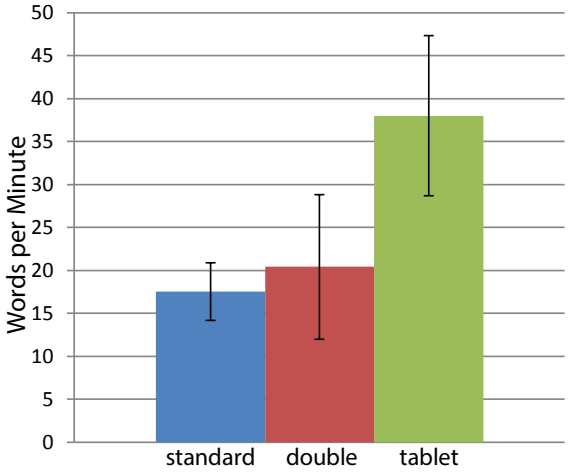


Figure 10. Entry rates in WPM for P5 with one-handed, two-device, and two-handed-one-device entry with Perkinput, Higher is better. Error bars represent ±1 *SD*.

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.

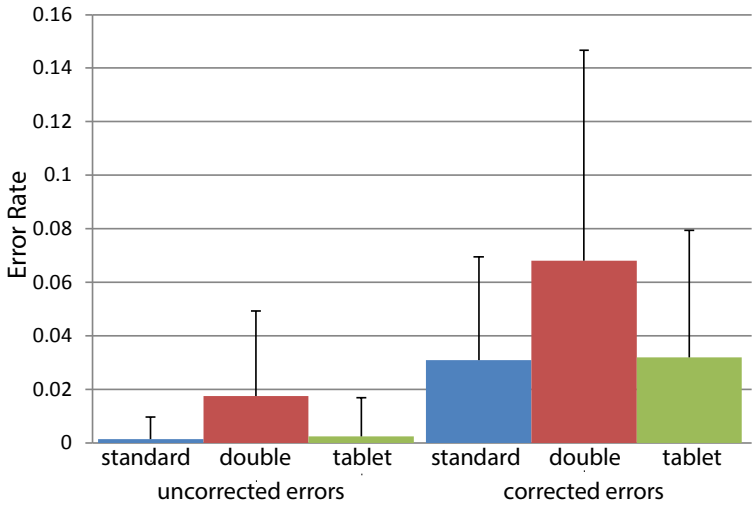


Figure 11. Error rates for P5 with one-handed, two-device, and two-handed-one-device entry with Perkinput. Lower is better. Error bars represent ±1 *SD*.

3.7. Discussion

Study 1, our longitudinal study, showed that Perkinput outperforms VoiceOver, the *de-facto* eyes-free text entry method for touch screen devices. Perkinput was faster and more accurate in the first as well as the final sessions of the study. Perkinput was preferred by the majority of participants (6 out of 8). Surprisingly, both users who preferred VoiceOver had higher average

entry rates with Perkinput, but higher corrected error rates. The errors frustrated them, although they managed to correct many of them quickly.

Performance on both VoiceOver and Perkinput for Study 1 was worse than we had expected. With both methods, participants had difficulty remembering a phrase, remembering how much of a phrase they had entered, or remembering how to spell words in a phrase. Also, participants often touched the screen without intending to do so. Some of these challenges were probably a result of the older age of most participants (average age was 55) and the lack of experience they had with touch screens (all but one had little to no experience). We were also 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 [17,32,72] are even lower than our findings.

Study 3 showed that Perkinput text entry on a tablet is dramatically faster than on a phone. This implies there is much potential for tablet use among the blind community and we encourage more work in this area. Similarly, there is a clear trend (p=.07) suggesting that Perkinput on two devices is faster than one-handed Perkinput. Nearly half of the phrases entered with two devices (45%) were entered at a faster rate than 22.47 WPM, the *fastest* single phrase entry rate achieved with one hand. Two-device input was more error-prone, however. This shows that there is potential to achieve far higher average rates with two devices after more practice, in cases where a larger device is not available.

Perkinput's Braille encoding was evaluated in our studies with participants who were proficient in Braille. Unfortunately, they are among a minority of blind people [67]. A person who does not know Braille would have more difficulty learning Perkinput than a person who is proficient with

Braille and it is likely that using VoiceOver would be faster initially. There is no reason, however, for VoiceOver to have an advantage to Perkinput after the user has memorized Braille character encodings. One participant suggested Perkinput would be a useful tool for learning Braille in the first place. Other encodings can be used as well, such as an encoding that assigns more ergonomic finger patterns to more frequent characters for faster and less strenuous entry.

Chapter 4.
PassChords: Secure Eyes-Free Authentication for Touchscreens

4.1. Motivation

This chapter focuses on eyes-free entry of private information on mobile devices and, more broadly, security and privacy concerns for eyes-free use of mobile devices. While working on Perkinput (see Chapter 3), we realized that speech-based access technology on mobile devices poses unique security and privacy risks for blind users. When a blind person uses a screen reader on a mobile device, people can hear the device speaking the contents of the screen and the user's input. The threat of eavesdropping is increased when people use their devices on public places. Suppose a blind person checks her email at a bus stop. A bystander may hear the blind person's device speaking the contents of an email, or information about the blind user's travel destination.

Another security issue that differs for blind and sighted users is accessibility of password entry. User authentication is an effective and common way to protect private data [45]. A recent study found that when smartphones were left unattended in public places, 89% of people who found the phones attempted to access the phone owner's private information. Use of a password to unlock the device screen protects against unauthorized user access. Yet password entry is likely to be an obstacle for blind users, since even sighted users find password entry on small touch screens to be a major frustration [44]. Moreover, screen readers introduce a severe vulnerability by speaking touched keys during password entry (see Figure 12). Over the past decade, the security community has explored the use of graphical authentication techniques as an alternative

to alphanumeric passwords [46,73,91,96]. These techniques do not require text entry but are inaccessible to blind people.

In this chapter, we discuss a study of privacy and security issues related to mobile access technologies. To alleviate some of these issues, we developed PassChords, an eyes-free authentication method based on Input Finger Detection (a technique presented in Chapter 3).

Figure 12. When entering passwords with an iPhone and VoiceOver, a user's input is spoken as she touches the screen, posing a severe security risk.

4.2. Threats and Defenses

We outline security threats for blind mobile device users and possible defenses against them. Like sighted people, we assume blind people access private data such as email communications, text messages, social networking, online banking, contacts information, and travel directions. We also assume blind people use their devices in public places like buses, street corners, and cafes, where others are present nearby. Unlike for sighted people, however, we believe the following

threats pose far greater risks for blind people because of screen reader technology or the lack of security features available in specialized access technologies.

We consider the following threats in this chapter:

Aural eavesdropping. Casual or malicious bystanders may overhear private information spoken by screen readers. Additionally, as a user enters input, the screen reader echoes the user's button selections. This occurs when a user enters a password as well, as shown in Figure 1. The threat of aural eavesdropping has been studied in the security literature for more subtle audio feedback such as keystroke sounds [6,13,31], highlighting the severity of the threat for screen reader output.

Visual eavesdropping. Casual or malicious bystanders may oversee private information displayed on a mobile device screen. If a person with low-vision is using large fonts or screen magnification, people may see the screen's contents from an extended distance.

Unauthorized user access. Both blind and sighted people face this threat, which occurs when a device is misplaced, lost, or stolen. We are interested in this threat because, as we discuss below, blind people may find it far more challenging to defend against it.

To assess the risk posed by the threats listed above, we enumerate possible defenses. In the following section we discuss how and when these defenses are used through an interview study, and asses threat risk.

Headphones. One can mitigate the risk of aural eavesdropping by using headphones when listening to screen reader output. However, when on-the-go, blind people use their hearing to

understand their environment and using headphones may be unsafe or inconvenient. A blind person may not want to use headphones every time she enters a password to unlock her screen.

Screen occlusion. It is possible to physically cover a screen with a hand or use software such as the iPhone's Screen Curtain [2]. Some access technologies such as Braille displays or audio recorders may be used instead of smartphones, as they do not have screens at all. Not displaying visual output would mitigate the risk of visual eavesdropping but may be impractical or difficult to use. Also, people with some functional vision may benefit from visual output.

Password protection. Protecting a device with a password that requires a user to authenticate herself before using a device is an effective defense against unauthorized user access. Many access technologies do not have password locks, however. People using smart devices that do have such features may find the standard password techniques to be too slow and error-prone (in addition to being insecure, because of screen readers speaking the input password).

4.3. Security-Related Use Patterns

Defenses against security threats have trade-offs and may negatively impact a user's experience with a device. We conducted interviews with blind people to understand how and why possible defenses were practiced. This enabled us to asses the risk of the security threats in our model.

2.2.1. Method

Participants

We recruited 13 participants (6 male, 7 female). The average participant age was 51 years (age range 26–64). We required that participants (1) were legally blind and (2) used smartphones daily. Two participants had some functional vision, one had light perception, and the remaining

10 were completely blind. Participants were recruited through email lists that catered to blind people.

Procedure

We conducted a semi-structured interview with each participant. All interviews were conducted over the phone and lasted about 20 minutes. We began by asking participants for demographic information such as gender and age. Then, we asked questions in the following categories: (1) context and frequency of mobile device use, (2) types of information accessed on mobile devices, (3) use of passwords on mobile devices, (4) use of headphones, and (5) use of screen occlusion techniques.

Analysis

The interviews were transcribed, coded, and then organized based on interview questions.

4.3.1. Results

All participants owned iPhones that they used with the VoiceOver screen reader. When on the go, 6 participants also carried a Braille notetaker, 2 carried accessible GPS systems, 1 carried a portable CCTV, and 1 carried a laptop. All devices were used on a daily basis in various contexts, including public places such as streets, cafes and restaurants, and also at home and at an office.

As expected, participants stored a wealth of private and personal information on their devices:

> *Gosh, you know [my iPhone] is just a part of me. I can't think of anything I don't do [on it].*

Participants regularly accessed private information, including email communications, social networking sites, and location-tracking applications such as Four Square. Nearly half of the participants used banking applications on their iPhones. One participant expressed a preference for accessing private data on her Braille notetaker, because others could not hear or see what she was reading.

None of the participants used optional authentication features to protect the information on their devices. In fact, the iPhone was the only device mentioned that had an authentication feature. All but one of the participants were aware of the iPhone's password protection feature, the Passcode Lock, and had decided not to use it. Some participants stated using the Passcode Lock was inconvenient: "No, [Passcode entry] is inconvenient—I don't want to do that"; others thought it was unnecessary: "...because I have my [iPhone] with me all the time."

Passwords were entered only when required by some applications that participants used, such as Facebook and Netflix. These passwords were usually stored by the applications, however, and did not require repeated entry. One participant expressed concern regarding aural eavesdropping, noting that VoiceOver spoke a key label as it was touched during password entry.

All participants used headphones when listening to screen readers in public spaces; 12 participants used headphones regularly (but not exclusively), and the remaining participant used them occasionally. Three participants (partly) attributed headphone use to concerns about aural eavesdropping, but most used headphones to avoid disturbing others around them or simply for better sound quality. There was a trade-off between the advantages of headphones and the need to hear sounds in one's environment.

> *I like to listen on the headphones but I don't like to have my hearing completely blocked out because it's hard to hear a bus stop and if there is something happening on the bus I need to be hearing. You know, like a fight or who knows?*

The iPhone's Screen Curtain feature, which disables visual output, was used by 10 participants; not being able to see the screen may serve as a security advantage for them. Four participants used the Screen Curtain to prevent visual eavesdropping, and most participants cited the desire to save battery power (the advertised purpose of the Screen Curtain).

> *I love that people can't look over my shoulder and see what I'm doing.*

No other screen occlusion techniques (*e.g.*, holding the device close to one's chest) were used.

While participants occasionally mentioned security threats, their primary concerns were related to iPhone accessibility. Participants had difficulty inputting text and accessing information from applications that were not compatible with VoiceOver. Several participants noted the physical challenge of interacting with their devices while using a cane. Only three mentioned the security risks associated with online banking, location tracking, and aural and visual eavesdropping. One participant acknowledged the need for better security mechanisms, although, like other participants, he did not use optional authentication methods.

> *I feel like I should use [security features on [my iPhone] and I'll probably be sorry one day that I didn't.*

4.3.2. Discussion

Our results indicate that a minority of users are aware of security threats, including aural and visual eavesdropping, and unauthorized user access. This is disturbing, but not surprising given that related work found that the general population lacks awareness and understanding of security threats [24,45,78]. We concur with this prior work that users should receive better

training—whether from Orientation & Mobility instructors or blindness organizations in general—about potential mobile device security risks.

The finding that our participants did not use optional authentication methods like the Passcode Lock to protect their devices from unauthorized user access was most alarming. Clarke and Furnell [24] report that one third of 297 (all sighted) participants locked their phones with PIN-based authentication, noting that this ratio was low. The fact that no participants used a Passcode Lock in our study was egregious, highlighting the severe risk of unauthorized user access. Text entry rates with VoiceOver were only about 4 words per minute (WPM) [11], so it may be infeasible for blind people to enter a PIN every time they unlocked the screen of their phone.

Security threats from aural and visual eavesdropping were mitigated by use of headphones and the Screen Curtain. Although all participants used headphones, they acknowledged their disadvantages. Security defenses should, therefore, not solely rely on headphone use, especially for highly private information such as passwords. Screen Curtains were not used by all participants, and participants were generally unaware of the need for protect against visual eavesdropping. It would be interesting to interview people who used magnification rather than screen readers, since magnification increases the risk of visual eavesdropping.

4.4. Secure Authentication with PassChords

The most severe security problem we identified from our study is the risk of unauthorized user access, which may be attributed to lack of user awareness, and the inaccessibility and insecurity of current password techniques. To address this problem, we developed a new touch screen

authentication method that is entirely non-visual, faster than PIN techniques, and robust to aural eavesdropping.

2.2.1. Design Principles

When developing an authentication method, we considered several design principles based on our interview study, our threat model, and standard authentication guidelines [5,82]. These principles emphasize both security and usability.

1. Speed. Users should be able to enter a password quickly.
2. Robust to aural eavesdropping. Users should be able to input a password without audio feedback that broadcasts their input.
3. Robust to visual eavesdropping. There should be little or no visual indication of the user's input.
4. High password strength. Password strength should not be sacrificed and the technique should be robust to guessing or brute-force attacks.
5. High recall. Passwords should be easy to remember.

4.4.1. The PassChords Technique

PassChords are a new authentication technique based on Input Finger Detection [11], where a user taps a touch surface several times with 1 to 4 fingers (see Figure 2). The PassChord is defined by the set of fingers used in each tap. At the beginning of a PassChord entry, the user calibrates the touch surface by entering reference points, which the PassChords algorithm uses to model the true locations of the fingers on the screen.

The PassChords algorithm determines which fingers touched the screen using Maximum Likelihood (ML) detection given the finger reference points. In Input Finger Detection [3], the variance of each finger is tracked and used in the ML detection. Since PassChords are short and we assume that, unlike in text entry, a user will not enter many PassChords in succession, we do not track variance. Instead, we assume equal variance for each fingers. ML thus reduces to finding the set of reference points that have the minimum combined distance from the set of input points.

Figure 13. A user calibrates (left) and enters a 3-tap PassChord. The blue circles show which fingers contact the screen in the figure but do not appear as output to the user. Note that the fingers are not striking bounded regions like buttons; rather, the finger locations are interpreted probabilistically, meaning some flexibility in their hit-location is allowed, while the number and identity of the fingers is appropriately strict.

As the user enters a PassChord, she receives only vibration feedback with no visual or audio output. A short vibration is produced when the user touches the screen. To calibrate, a user presses 4 fingers to the screen until a second vibration is produced less than a second later. No further feedback is needed because, as with any chording technique, people can discern their input through proprioception. Techniques that rely on a fingertip at a certain position require audio feedback because different inputs “feel” the same.

We believe that PassChords would be easy to remember because the chording nature of the technique is evocative of playing a piano or another chording instrument. Also, people may associate numbers with the fingers used, allowing similar recall techniques to numeric passwords.

4.4.2. Entropy

Information entropy is a common measure of password strength, indicating how robust a technique is to guessing or brute-force attacks [20]. In this metric, the information entropy of a password of n symbols from a symbol set of size m is $log_2 mn$, measured in bits. In other words, the information entropy of a password technique is the minimum number of bits needed to encode the set of all possible passwords, assuming all symbols are equally likely.

In one tap of a PassChord, there are 15 possible finger combinations. Each of 4 fingers is either touching the screen or not, and a tap where all fingers are not touching the screen is invalid. A PassChord with 4 taps therefore has information entropy $log_2 154 \approx 15.6\ bits$. By contrast, consider a standard PIN's information entropy. Each digit in the PIN has 10 possible inputs, so a 4-digit PIN has information entropy $log_2 104 \approx 13.3\ bits$. Both the 4-tap PassChord and the 4-digit PIN require the same number of symbols as input, but the information entropy of the PassChord technique is higher, indicating it may be more robust to attacks.

The information entropy assumes that all symbol entries are equally likely, which is probably not true for PIN entry and certainly not true for tap entry. As we will see in our study, some finger combinations are more likely than others because of the physiology of the hand. For example,

simultaneously tapping the middle and pinky fingers is more difficult than tapping the index finger. A better estimate of the entropy of password strength is the first-order entropy:

$$H = n \sum_{i=1}^{m} p_i \, log_2(1/p_i) \tag{4.1}$$

where p_i is the probability of symbol i occurring in any position in a password. We will empirically calculate *H* from user data to estimate the security strength of PassChords.

4.4.3. Evaluation

To evaluate the PassChords authentication technique, we sought to compare PassChords to a standard password technique. We chose the iPhone's Passcode Lock with VoiceOver as a basis for comparison, which consists of a 4-digit PIN that is entered with an on-screen number pad.

Method

Participants. We recruited 16 blind participants (8 male, 8 female), with an average age of 51 (age range 27–61). While all were legally blind, five participants had some vision and were able to identify numbers on an iPhone's number pad. The remaining 11 had no functional vision. Eight participants had experience with VoiceOver on iOS devices. We recruited participants through mailing lists that communicated with blind people.

Apparatus. We built prototype applications for PassChord and PIN entry. We did not use an iPhone's built-in Passcode so we could instrument the application. The PIN application was visually similar to the iPhone's Passcode Lock, enabling split-tap and double-tap selection of

keys. As with the iPhone, the PIN application spoke button labels as they were touched, but did not provide feedback when a number was entered. Both applications logged every user input.

A Samsung Galaxy phone was used for all user studies, with a 4-inch screen.

Procedure. Participants completed two sessions, one with each authentication method. The beginning of each session included a training period, where we taught participants how to use the method for the current session. Participants practiced the method until they were able to authenticate with three different passwords.

After training, participants entered three passwords: the first was prescribed by the experimenter and the other two were created by the user. We sought to simulate a realistic password creation and entry scenario, so we asked participants to create a password, confirm it, and then enter it 20 times. The confirmation of a password allowed participants to practice their new password and ensure they had created it as intended.

The first PIN was a randomly generated sequence of 4 digits. The first PassChord included three touches, each consisting of one randomly selected finger. We anticipated certain multi-finger combinations would be difficult for participants, so we gave them a PassChord where each touch included only one finger. For the next two PassChords, we instructed participants to create a PassChord where at least one of the touches had more than one finger. For both methods, participants were instructed to create passwords that were ”realistic.”

Participants were able to correct errors during password entry. The VoiceOverPIN number pad included a backspace key. A PassChord could be ”reset” if the user made an error by calibrating

and re-entering the PassChord. Such errors and corrections were included in the time measured for a given password entry.

After entering three PassChords repeatedly, we asked participants to create yet another PassChord which they were tasked to memorize. Two days after the study we called each participant and asked them to repeat the memorized PassChord. We instructed participants to behave as though this was a ”real” password, and use whatever memorization technique seemed appropriate.

Design and Analysis. The study was a within-subjects factorial design with two factors, Method and Order. The levels of Method were (PassChords, VoiceOverPIN) and the levels of Order were (1, 2). The Order factor indicated whether the current Method was performed first or second in the study, allowing us to evaluate possible crossover effects.

We analyzed two measures: authentication time and failure rate. The former was measured as the difference between the time of the first and last touch events of a password (including PassChord calibration), and the flatter was the proportion of times the user failed to authenticate. The failure rate included completed passwords that turned out to be incorrect, not counting errors that were corrected by the user with the backspace key or a re-calibration. Such errors were subsumed by the password entry time. Both measures were analyzed with mixed-effects model analysis of variance, with a fixed effect for Method and a random effect for Participant to account for correlated measurements for different methods within subjects. Authentication times were averaged for trials in each method. We used a significance level of $\alpha = 0.05$.

Neither authentication time nor failure rate was normally distributed ($W = 0.90$, $p < 0.001$ for time; Shapiro–Wilk $W = 0.89$, $p < 0.001$ for failure rate). Therefore, we used the nonparametric Aligned Rank Transform procedure [99], which enables the use of ANOVA after alignment and ranking, while maintaining the integrity of interaction effects.

Results

Authentication time. PassChords were nearly three times as fast as VoiceOverPINs. The mean authentication time for PassChords was 2.67 seconds (SD = 0.722), while that for VoiceOverPIN was 7.52 seconds (SD = 2.40). This difference resulted in a significant effect of Method on authentication time ($F1,13 = 113.6$, $p < 0.001$).

The number of taps per PassChord ranged between 3 and 6 taps, and the mean time per tap was 0.62 seconds (SD = 0.17). The mean time for a VoiceOverPIN input was 1.89 seconds (SD = 0.60). Thus, it is evident that PassChords would have outperformed VoiceOverPINs if we had required an equal number of inputs for each. The large difference was not surprising since participants often had to search for the correct VoiceOverPIN input by moving their finger across the screen while listening to screen reader output.

Strangely, there appeared to be an asymmetric skill transfer between methods. Participants who entered PassChords after they had entered VoiceOverPINs performed better with PassChords than participants who entered PassChords first. This resulted in a significant effect of Order ($F1,13 = 12.8$, $p < 0.01$) and a significant interaction of Method by Order ($F1,13 = 10.0$, $p < 0.01$). As Figure 14 shows, however, the difference between method entry times was incontrovertible, in spite of the effect of order.

Failure rate. There was no speed-accuracy trade-off, as the failure rate was slightly lower for PassChords than for VoiceOverPIN. Participants failed to authenticate 16.3% of the time with PassChords (SD = 14.5%) and 20.2% of the time with VoiceOverPIN (SD = 17.3%). This differences were not significant, however ($F_{1,13} = 1.49$, *n.s.*).

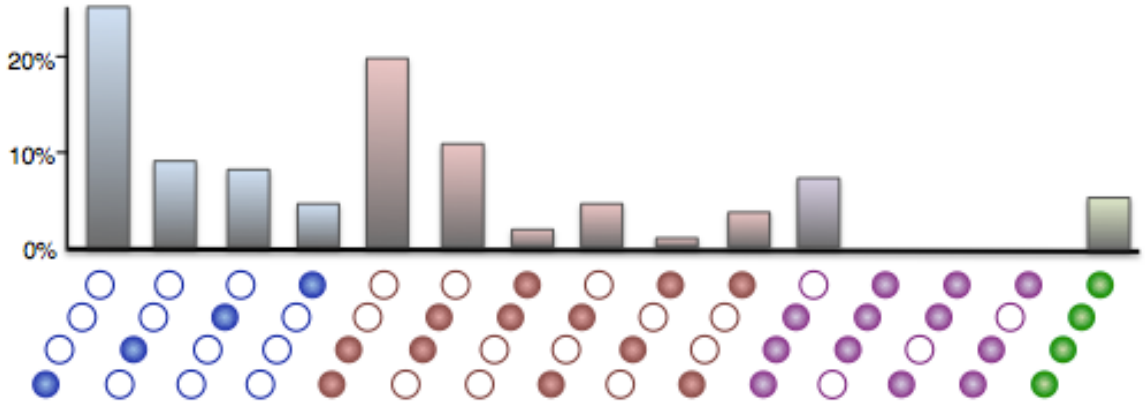


Figure 14. : PassChord pattern frequencies. A sequence of circles represents a tap pattern, with the index finger shown on the bottom left and the pinky finger on the top right. Certain patterns were chosen far more often than others.

Recall. Twelve of the 16 participants (75%) remembered their PassChord two days after they were asked to memorize it. Most participants tapped the password several times to memorize its "feel," and associated the fingers in each tap with numbers to memorized their pattern.

Password strength. We assess password strength by observing common patterns in the 32 user-generated PassChords. Our prior discussion of entropy assumed a uniform distribution of possible inputs. This is not the case, however, for user-generated passwords. Prior work shows that the most common digit in alphanumeric passwords is 1, and the most common letters are a,

e, o, and r [7]. Such patterns reduce the difficulty of guessing or brute-force attacks, so they are important to identify and avoid [8].

Figure 3 shows the frequency of each tap pattern in the user-generated PassChords. A striking trend was the frequent use of the index finger, which was present in 66.5% of taps. The pinky finger was used least, in only 14.6% of taps (see Table 1). Users tended to create passwords with repeating finger combinations and individual taps were often made with adjacent fingers. The most common PassChord length was three taps, although this may be attributed to the length of the initial, prescribed PassChord that served as an example.

Table 1. Frequencies of finger use in PassChord taps. There was a strong preference for using the index finger, which is often used for touch screen input.

Finger	***Frequency***
Index	66.5%
Middle	51.7%
Ring	36.6%
Pinky	14.6%

4.4.4. Discussion

We have shown through the design and evaluation of PassChords that our design principles were satisfied. In terms of usability, PassChords are nearly 75% faster to enter than accessible PINs, with comparable authentication failure rates. While merely preliminary, our study of PassChord recall demonstrated that there were no unexpected obstacles with PassChord memorization.

The security of PassChords was considered in their design and evaluation. Unlike accessible PINs, PassChords produce no audio feedback, so they are more resistant to aural eavesdropping.

PassChords also display no visual feedback, making visual eavesdropping more challenging. It would be interesting in future work to assess the threat of "shoulder-surfing" attacks that occur when an adversary eavesdrops by looking over a user's shoulder and observing her finger motions.

The data collected in our study, which included 112 PassChord taps and 128 PIN digits, yields preliminary estimates of the security strength in terms of first-order entropy using Equation (1). The first-order entropy of 4-tap PassChords was $H \approx 12.6$, comparable to the first-order entropy of 4-digit PINs of $H \approx 12.7$. Our sample size was too small to produce these estimates with high confidence, but they give a rough idea for the security strength of both methods.

The security strength of PassChords can be improved by ensuring that the distribution of taps is as close to a uniform distribution as possible and using as many taps as possible. This leads us to several guidelines to help users create more secure PassChords:

1. Use each finger at least once in your PassChord.
2. Use taps of one, two, and three fingers.
3. Use four or more taps in your PassChord.

Since knowledge-based passwords are common authentication mechanisms, we believe PassChords will impact the security of mobile devices for blind people. They are an important first step at addressing security challenges for blind mobile device users, as discussed in our threat model. Since entering passwords on small touch keyboards is challenging for sighted users as well, we believe PassChords would benefit people with all visual abilities.

Chapter 5.
DigiTaps: Eyes-Free Number Entry with Natural Prefix-Free Codes

5.1. Motivation

One of the limitations of Perkinput was its reliance on Braille. While studies show that it is extremely important for blind people to learn Braille, many blind and low-vision people are not Braille-literate [67,92]. Moreover, sighted people who can also benefit from eyes-free input do not usually know Braille and have less of an incentive to learn it than blind people. Thus, we sought to develop an eyes-free input method that was easily-learnable and did not require specialized knowledge.

In this chapter, we describe *DigiTaps*, an eyes-free number entry method with natural prefix-free codes (Figure 15). DigiTaps can potentially be used in a variety of contexts: when making phone calls, interacting with automated voicemail systems, entering addresses in GPS-based applications, and entering personal identification numbers (PIN). The code for each digit is based on the semantics of the digit, so DigiTaps is easy to learn and does not require knowledge of Braille, Morse code, or another specialized encoding.

Figure 15. A blind person enters numbers with DigiTaps. The white circles at the bottom of the screen mark her touch points.

Like Perkinput and PassChords, the DigiTaps gestures "feel" different, so people can discern whether they touched the screen with one finger or two. We thus hypothesized that Digitaps can be used with *minimal audio feedback*. With VoiceOver and many other access technologies, users have to listen carefully as they interact with their device. This can be difficult to do in noisy environments. Using headphones can make it easier to hear the audio feedback from the device, but headphones can block out environment noise and can simply be clumsy and time-consuming to attach. We sought to show that DigiTaps not only learnable, but also usable with only minimal-audio feedback.

5.2. The Design of Digitaps

The discussion in the previous section leads us to four design goals for numeric entry. We strive to develop a method that is (1) eyes-free, (2) minimal-audio while being (3) efficient and (4) learnable. Efficiency and learnability are important for widespread adoption.

5.2.1. DigiTaps Gestures

To enter a digit with DigiTaps, a user performs one, two, or three gestures anywhere on the

screen. The gestures used for digit entry are a one-finger swipe, a one-finger tap, a two-finger tap, and a three-finger tap. A BACKSPACE is entered with a 2-finger swipe. Swipes can be oriented in any direction.

During our formative studies, we considered using a four-finger tap. It was difficult for people to touch a surface with four fingers simultaneously, however, because the smallest finger, the "pinky," was typically much shorter than the other three fingers. Also, some people could not easily fit four fingers on a phone's screen, so we decided to use gestures that involved up to three fingers at a time.

5.2.2. DigiTaps Codes

After designing the DigiTaps gestures, our goal was to create a code that uses just four symbols to represent 10 digits. Usually, we use 10 symbols, the standard digits 0–9, to encode numbers using place values. One possible way to encode 10 digits with four gestures is to use base-4 notation. The 10 digits would be represented by 0, 1, 2, 3, 10, 11, 12, 13, 20, 21, respectively. This encoding could be difficult to learn, however. Can people quickly learn that the digit 5, for example, is represented by 11 in base-4? Furthermore, there is ambiguity in this encoding: if the digit 1 is represented by a one-finger tap, how would the system distinguish the entry of digit 5, input with two one-finger taps, from the entry of two 1's, which are also input with two one-finger taps. To resolve this ambiguity we could use a fixed-length, base-4 code to represent the 10 digits. This code would be 00, 01, 02, 03, 10, 11, 12, 13, 20, and 21 for digits 0 to 9, respectively. There is no ambiguity, but the code is less efficient and could still be difficult to learn.

To reduce the number of gestures per digit, we use prefix-free codes that allow for a varying number of symbols per digit with no ambiguity. The prefix-free property means that no symbol sequence for one digit is a prefix of another [42]. In the original base-4 code, 1 is a prefix of 11, which was ambiguous and led us to define the fixed-length base-4 code. In a prefix-free code, such ambiguity cannot happen and the number of symbols per digit is lower than in a fixed-length code. If all the digits are equally likely, the average number of symbols per digit is the sum of the lengths of the codes for all digits divided by 10.

As such, we developed a learnable, prefix-free code called DigiTaps$^{2.1}$ (first introduced in [79]. The digits can be derived by adding the symbols 0, 1, 2, and 3, entered by gestures, as shown in Table 2. For example, 0 is represented by a one-finger swipe, 1 by a one-finger tap, and 2 by a two-finger tap. The digit 3 is represented by a three-finger tap followed by a one-finger swipe, or 3 + 0. Similarly, 4 is represented by a three-finger tap followed by a one-finger tap (3 + 1). We represent 6 with two three-finger taps and a one-finger swipe (3 + 3 + 0). The digits 3 and 6 need a final one-finger swipe to ensure the code is prefix-free. The digit 9, however, is represented by three three-finger taps (3 + 3 + 3), since three three-finger taps are not a prefix to another input.

DigiTaps$^{2.1}$ uses 2.1 symbols on average per digit, which is close to the two symbols per digit of the fixed-length base-4 code, without the complexity of base-4 notation. An optimal prefix-free code, however, uses 1.8 symbols per digit. There are $6 \cdot 10!$ [2]$= 21{,}772{,}800$ distinct optimal codes, but are any as learnable as DigiTaps$^{2.1}$?

---

[2] There are 6 ways to pick the two one-symbol codes from 4 symbols, and then there are 10! possible assignments of the 10 digits to the 10 resulting codes.

We developed a second code, DigiTaps$^{1.8}$, which has an optimal number of symbols per digit (see Table 2). In DigiTaps$^{1.8}$, the one-finger swipe represents either 0 or 10. It represents 10 when it is the first gesture in a two-gesture digit, and the digit is derived with subtraction. For example, the digit 7 is represented by a one-finger swipe followed by a three-finger tap (10 – 3). Similarly, the digits 8 and 9 are represented by a one-finger swipe followed by a two-finger tap (10 -2) and a one-finger tap (10 – 1), respectively. In all other cases, the one-finger swipe represents 0. The digit 3, for example, is represented by a three-finger tap and a swipe (3 + 0). DigiTaps$^{1.8}$ and DigiTaps$^{2.1}$ have the same codes for five of the 10 digits.

Table 2. DigiTaps Gestures.

Digit	***DigiTaps$^{2.1}$***	***DigiTaps$^{1.8}$***
0	1-finger swipe	1-finger swipe, 1-finger swipe
1	1-finger tap	1-finger tap
2	2-finger tap	2-finger tap
3	3-finger tap, 1-finger swipe	3-finger tap, 1-finger swipe
4	3-finger tap, 1-finger tap	3-finger tap, 1-finger tap
5	3-finger tap, 2-finger tap	3-finger tap, 2-finger tap
6	3-finger tap, 3-finger tap, 1-finger swipe	3-finger tap, 3-finger tap
7	3-finger tap, 3-finger tap, 1-finger tap	1-finger swipe, 3-finger tap
8	3-finger tap, 3-finger tap, 2-finger tap	1-finger swipe, 2-finger tap
9	3-finger tap, 3-finger tap, 3-finger tap	1-finger swipe, 1-finger tap

5.2.3. DigiTaps Feedback

DigiTaps provides haptic and optional audio feedback that "clicks" when a gesture is entered and speaks each digit. DigiTaps' haptic feedback enables users to determine whether the system detected their gesture correctly so they can correct errors inline. In formative studies, we found

that (1) participants were sometimes unsure whether their touches were registered by the system, and (2) the system sometimes registered a swipe when a tap was entered and vice versa. Thus, DigiTaps gives two kinds of haptic feedback. When the user touches the screen, the device vibrates. If the touch produced a swipe, the system pulses an extra vibration. There is no extra pulse for a tap, so the user knows which gesture was registered.

5.3. Evaluation

We evaluated DigiTaps to find out whether it was a viable method for use with minimal audio feedback. We compared DigiTaps entry (1) with and without audio feedback, and (2) with the DigiTaps$^{2.1}$ code and the DigiTaps$^{1.8}$ code.

5.3.1. Methods

Participants

We recruited 16 participants to complete the DigiTaps longitudinal study (six blind, 10 sighted). Blind participants had a mean age of 45 (range: 31-57), and sighted participants had a mean age of 28 (range: 22-34). All had experience using touchscreen devices. We recruited blind and sighted participants through mailing lists that were affiliated with blindness organizations and university communities, respectively.

Procedure

Blind and sighted participants completed the same study procedures with one exception: sighted participants held the phone under the table (as in Clawson *et al.*'s study [25]), while blind people held the phone in their preferred position. Participants completed six sessions that lasted about 35 minutes each. Sessions were at least two hours, but no more than four days, apart.

In each session, participants entered sets of random four-digit sequences under four conditions: with each DigiTaps code and each feedback method (haptic only or haptic with audio). In sessions 2 - 6, participants entered 15 sequences in each condition. In session 1, they only entered sequences with audio feedback for each code, since they were still learning the codes.

We counterbalanced the order of the DigiTaps codes but did not counterbalance the order of feedback methods. We were concerned that participants would confuse the two codes if they were alternating between them and no audio feedback was given.

Apparatus

We developed an instrumented prototype of DigiTaps that ran on two Galaxy Nexus phones. The phone screens were 4.65 inches long.

Design and Analysis

The experiment had three within-subject factors: *Code* (DigiTaps$^{2.1}$ and DigiTaps$^{1.8}$), *Session* (1-6), and *FeedbackMethod* (Audio or Haptic). We computed the characters per second (CPS) and uncorrected error rate to assess speed and accuracy, respectively, following the formulas and algorithms described by Wobbrock and Myers [100]. The uncorrected error rate measures the frequency of errors that were not corrected by the user and appear in the final transcribed strings. Another accuracy measure, the corrected error rate, measures the frequency of errors that the user corrected. It is subsumed by the CPS, however, so we exclude it from our report.

We assessed the normality of our data through graphical means (plotting histograms and boxplots). CPS was normally distributed so we modeled it with a linear, mixed-effects model using *Participant* as a random effect. The uncorrected error rate was not normally distributed,

which is usually the case with error rates. We therefore compared sample means of uncorrected error rates with Wilcoxon Signed Rank tests. We use significance levels of $\alpha = 0.05$.

5.3.2. Results

Entry Rate

We found that DigiTaps entry with no audio feedback was faster than with audio feedback, and that the DigiTaps$^{1.8}$ code was faster than the DigiTaps$^{2.1}$ code. This resulted in a significant effect of *Code* ($F_{(1,15)} = 6.90$, $p = 0.02$) and *FeedbackMethod* ($F_{(1,154)} = 27.28$, $p < 0.01$) on CPS. Participants significantly improved their entry rate over time, yielding a significant effect of *Session* ($F_{(1,156)} = 27.28$, $p < 0.01$) on CPS. Their improvement was about the same with both codes and feedback methods, resulting in no significant interaction effects. Table 3 shows the mean entry rates for each condition and Figure 16 shows the entry rates over all sessions of the study.

Table 3. Mean CPS rates for each DigiTaps feedback method and code.

	DigiTaps$^{1.8}$	***DigiTaps$^{2.1}$***
Haptic feedback	0.89 (SD = 0.31)	0.85 (SD = 0.32)
Audio feedback	0.82 (SD = 0.36)	0.78 (SD = 0.34)

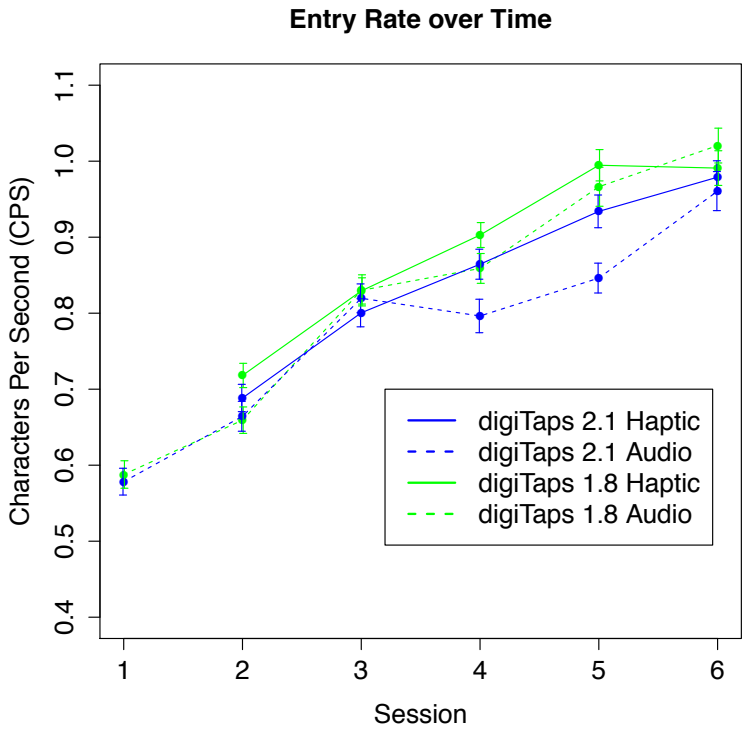


Figure 16. Entry rates for the six sessions of the study. The error bars show the standard error of the means. Higher numbers are better.

Accuracy

The uncorrected error rate was higher for haptic feedback than for audio feedback. This resulted in a significant effect of *FeedbackMethod* ($W = 5$, $p < 0.01$) on Uncorrected Error Rate. There was no significant difference between the uncorrected error rates for DigiTaps$^{2.1}$ and DigiTaps$^{1.8}$. Participants made fewer errors as the study progressed, and we found a significant effect of *Session* ($F_{(1,156)} = 15.69$, $p < 0.01$) on the uncorrected error rate. Table 4 shows the mean entry rates for DigiTaps feedback methods and codes and Figure 17 shows how they changed over time.

Table 4. Mean uncorrected error rates for each DigiTaps feedback method and code.

	***DigiTaps*$^{1.8}$**	***DigiTaps*$^{2.1}$**
Haptic feedback	5.58% (SD = 14.6)	5.68% (SD = 15.0)
Audio feedback	2.17% (SD = 8.96)	2.22% (SD = 8.7)

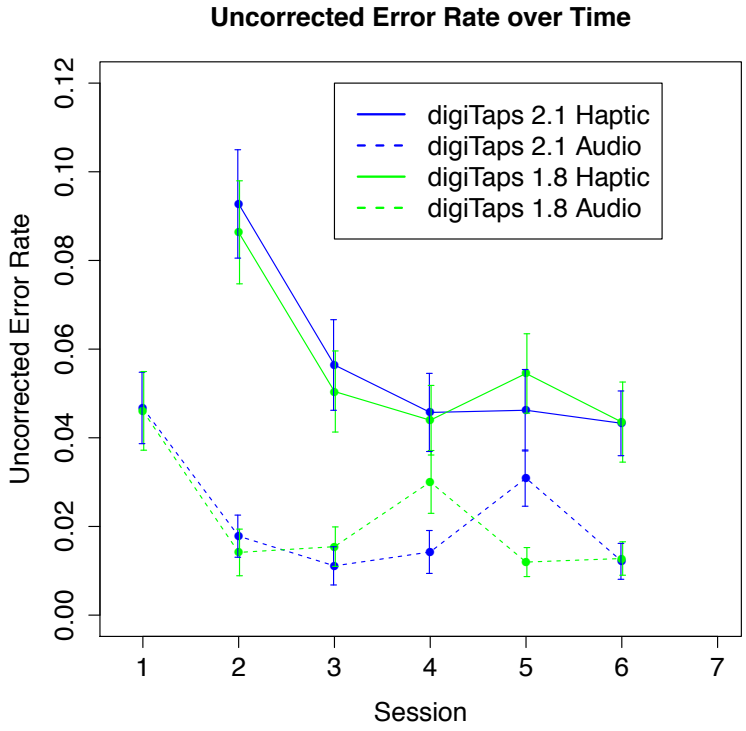


Figure 17. Uncorrected error rates for the six sessions of the study. The error bars show the standard error of the means. Lower numbers are better.

Subjective Preference

We asked participants which method they preferred at the end of the study. Twelve (71%) out of the 16 participants preferred DigiTaps$^{1.8}$, citing its speed and the lower number of taps per digit. A one-sample Pearson Chi-Square test of proportions showed that this preference was significantly different than chance, ($\chi^2_{(1,N=16)} = 4.00$, $p = 0.046$).

5.4. Discussion

Overall, we believe the speed and accuracy results show DigiTaps is a promising eyes-free technique. Our work on PassChords (Chapter 4) showed that blind iPhone users entered a familiar 4-digit PIN in 7.52 seconds on average with VoiceOver. With DigiTaps, blind and sighted people entered randomly generated four-digit sequences in 7.01 seconds after only the first study session. During the final session, they entered four-digit sequences in 4.1 seconds on average. Moreover, participants' rate of improvement (shown in Figure 2) suggests their entry speed will continue to increase with more practice.

Number entry with only haptic feedback was faster but also less accurate than entry with haptic and audio feedback. The former was faster probably because users did not pause after each entry to hear the spoken digit, and did not correct as many of their errors. We were pleased that by Session 6, the uncorrected error rate was still rather low, at only 4.6%. However, it is difficult to draw conclusions from the comparison of methods of feedback, because we did not counterbalance the order of the audio and haptic conditions in the study. There may have been carry-over fatigue or learning effects from the audio condition. Nonetheless, it seems likely that performance with only haptic feedback was not substantially *worse* than with audio feedback.

There was a small, albeit statistically significant, difference between entry rates between DigiTaps codes. DigiTaps$^{1.8}$ was faster than DigiTaps$^{2.1}$ with no speed-accuracy trade-off. Surprisingly, even though most participants felt DigiTaps$^{2.1}$ was easier to learn, performance with both codes was nearly equal in the first session. In later sessions, the difference in speeds between the codes was small, but most preferred DigiTaps$^{1.8}$.

We believe our results show that DigiTaps offers a new approach to eyes-free input that requires *minimal* audio feedback. Blind people would still need audio feedback to confirm the numbers

they entered were correct; they would also need audio feedback for any other verbal output from their device. However, they wouldn't be as reliant on the audio feedback when entering numbers with DigiTaps. A blind person could sit in a loud café and comfortably enter a number without holding the device up to his or her ear. DigiTaps could be useful for sighted people as well. A sighted person could enter a street address while walking and glance down to ensure it was correct. Eyes-free minimal audio interaction would enable people to use their touchscreen devices more comfortably in a variety of situations.

Chapter 6.
Exploring the Use of Eyes-Free Input by Blind People on Mobile Devices

6.1. Motivation

In the previous chapters, we have discussed our efforts and those of many other researchers to improve eyes-free text entry on touchscreen devices. All of the methods involved onscreen gestures. Surprisingly, to our knowledge, no one had explored speech as an eyes-free input modality. Yet speaking is natural, fast, and nonvisual, and automatic speech recognition (ASR) is already well integrated on mobile platforms. Mobile device users can enter text with speech in any text entry scenario on iOS and Android devices (see Figure 18) and perform actions such as searching the internet and creating calendar events with Google Voice Search [37] and Siri [3].

Figure 18. The Android (left) and iPhone (right) keyboards have a dictate button to the left of the space key that enables users to dictate text instead of using the on-screen keyboard.

While the act of speaking is eyes-free, the process of dictating, reviewing, and editing text is complex, and requires additional input. Both reviewing and editing may be challenging for blind people with today's mobile technology. A sighted person can review a speech recognizer's

output by simply reading the text on the screen. Meanwhile, a blind person can use VoiceOver to review the text with speech output. Detecting recognition errors with speech output may be difficult, however, since the recognizer is likely to output words that *sound* like those the user said, but are incorrect. For example, ASR is known to have difficulty with segmentation, and may recognize the words "recognize speech," as the similarly-sounding "wreck a nice beach."

After identifying recognition errors, blind people may face challenges correcting these errors. Prior work has shown that sighted users spend the majority of their time correcting errors when dictating text. Karat *et al.* [50] found that sighted people spent 66% of their time editing ASR output on a desktop dictation system. The editing process for blind users on mobile devices is likely to be much slower, since users may resort to using the on-screen keyboard to correct and edit ASR output. Perhaps the combined difficulty of reviewing and editing ASR output will outweigh the ease and speed of speech.

In this chapter, we explore the patterns and challenges of the use of speech input by blind people on mobile devices. Our ultimate goal is to improve the experience of nonvisual speech input, but first we study current use to identify specific challenges that the accessibility community can address.

6.2. Survey: Patterns of Speech Input among Blind and Sighted People

We conducted a survey to determine how often blind people use speech input on their mobile devices, what they use it for, and how they feel about it. We surveyed both blind and sighted people to evaluate the nonvisual experience against a baseline of the common (visual) use case.

6.2.1. Methods

We surveyed 54 blind participants, 10 low-vision participants, and 105 sighted participants. There were 31 female and 33 male blind/low-vision (BLV) participants, with an average age of 40 (age range: 18 to 66). Sighted participants were younger, with 51 males and 54 females and an average age of 32 (age range: 20 to 66). We sent emails on mailing lists related to our university and blindness organizations to recruit participants. Participants did not receive compensation for completing the survey.

The survey included a maximum number of 9 questions. The first three questions asked for demographic information: age, gender, and disability. The next question asked:

> Have you **recently** used dictation instead of a keyboard to enter text on a smartphone?
> Examples of dictation include:
> - Asking Siri a question, *e.g.*, "what's the weather like today?"
> - Giving Siri a command, *e.g.*, "call John Johnson"
> - Dictating an email or text message
> Required.
>
> [ ] Yes
> [ ] No

If the participant answered "No" to the question above, the survey concluded with a final question that asked why not. If the participant answered "yes," she was asked to recall a specific instance in which she used dictation. She was then asked several questions about this instance, such as when the instance occurred, and what she dictated (a question to Siri, a text message, *etc*.). The penultimate question presented the user with three statements and asked her to describe how she feels about each statement on a Likert scale. The statements were:

- Dictation on a smartphone is *accurate*
- Using dictation on a smartphone (including the time it takes to correct errors) is *fast* relative to an on-screen keyboard.

- I am *satisfied* with dictation on my smartphone.

The survey concluded with a prompt for "other comments" and a free-form text box for their response.

Surveys were completed on the Internet and responses were anonymized.

To analyze the results, we graphed the data and computed descriptive statistics for all questions. We used Wilcoxon Rank Sums tests to compare means between Likert scale responses. We modeled the data with one factor, *SightAbility*, with two levels: *BLV*, and *Sighted*. The measures corresponded to the three Likert response statements: *Accurate*, *Fast*, and *Satisfied*.

6.2.2. Results

The survey responses showed that BLV people used dictation far more frequently than sighted people. Interestingly, 58 BLV participants (90.6%) and 58 sighted participants (55.2%) used dictation recently. Among BLV participants, only one used speech input on an Android device and the rest used it on iOS devices. Among sighted people, 21 participants used speech input on an Android device and 34 on an iOS device. Most BLV people used speech input within the last day while most sighted people used it within the last week.

Both BLV and sighted participants used speech most for composing text message. Table 1 shows the kinds of messages participants composed. Many more BLV than sighted people used speech to compose emails. Figure 2 supports this finding, showing that BLV people composed longer messages.

Table 5. Number of responses for a survey question.

What did you use speech input for?	***BLV***	***Sighted***

A command (*e.g.*, "Call bob smith")	8	14
A question (*e.g.*, "Siri, what's the weather like today?")	13	14
An email	12	4
A text message	20	19
Other	5	7

Figure 19 shows the means and standard deviations (SD's) of participant Likert scale responses to the penultimate question of the survey. Histograms of the data showed the distributions of responses were roughly normal, so the means and SD's represent the responses appropriately. As Figure 3 shows, BLV people were more satisfied with speech and thought it was faster than the on-screen keyboard compared with sighted people. This resulted in a significant effect of *SightAbility* on *Satisfaction* ($W = 2296$, $p < 0.001$) and *Speed* ($W = 2240$, $p = 0.001$). There was no significant effect of *SightAbility* on *Accuracy*, but there was a strong trend ($W = 1977$, $p = 0.067$). Perhaps BLV people were able to get fewer recognition errors because they had more practice using speech for input.

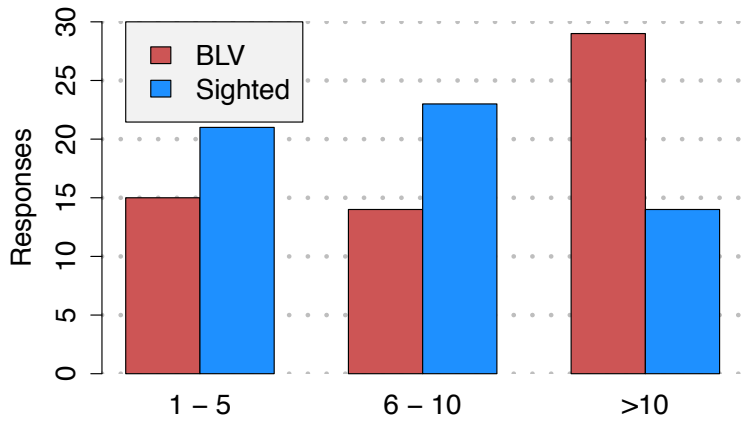


Figure 19. Survey responses to the question, “About how long was your dictated text?”

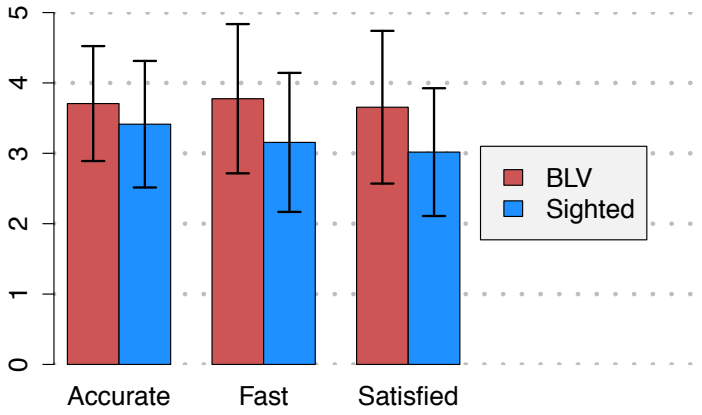


Figure 20. Survey responses on a 5-point Likert-scale: 1 is strongly disagree, and 5 is strong agree. Responses were roughly normally distributed.

When prompted for other comments, many participants noted the challenge of editing the recognizer’s output, and speaking in noisy environments. One blind participant explained,

> *Accuracy in noisy environments is the biggest challenge I feel. I prefer to dictate short commands and text, saving long e-mail responses for a standard computer. Editing can be a challenge.*

Three sighted participants felt awkward using speaking to their device. Another sighted participant echoed this concern, feeling frustrated with the lack of feedback, “I find it hard to talk to the device. Do you yell at it and hope it understands better?”

Participants who did not use speech for input recently were mostly concerned with accuracy and errors. Some were also concerned about privacy or social appropriateness, since other people can hear what they say when they speak to their devices. Some sighted participants said that they simply “don’t need” to use speech for input or haven’t figured out how to use it yet.

6.2.3. Discussion

The survey results suggest that speech is already a widely used eyes-free alternative to keyboard input. Blind people seem more satisfied with speech than sighted people. This is probably because keyboard input with VoiceOver is so much slower than standard keyboard input, that sighted people do not feel speech input offers a significant advantage.

We were surprised that nearly half of the blind participants reported that their recent speech input message was over 10 words long. From anecdotal experiences and formative work, it seems that it would be difficult to review and edit a message that was more than 10 words long. On the other hand, more blind people entered text messages with speech than emails, suggesting that perhaps speech input was preferred for shorter and more casual messages.

The survey findings raised new questions. Exactly how long were the messages blind people entered with speech? The results suggested speech input was much faster than keyboard input, but by how much? How did participants review and edit their text, and how much time did they spend doing so? In the next section, we describe the study we conducted to answer these questions.

6.3. Study: Observing the Use of Speech Input by Blind People

After finding that nearly all of our blind survey participants used speech for input, we wanted to learn more about their experience using it. We conducted a laboratory study to observe blind people composing paragraphs with dictation and the accessible on-screen keyboard.

6.3.1. Methods

Participants

We recruited eight blind participants (five males, three females) with an average age of 44 (age range: 22 to 61). We required that participants be blind and use a smart mobile device such as a smartphone. All participants owned iPhones, which they used many times a day. Two participants had their phones for about a year, and the rest had their phones for two or more years. All had no functional vision and used VoiceOver to interact with their phones.

Since we wanted to observe how blind people use speech input in their daily lives, we also required that participants have experience using speech on their mobile devices. Six participants used speech input every day, while the remaining two participants used speech input weekly.

Procedure & Apparatus

Participants completed one session in the study that was about one hour and fifteen minutes long. At the beginning of a session, we asked participants for demographic information, and asked them how often and what they used speech as input for on their mobile devices. We then showed participants an iPod Touch 5 that we used for the study and adjusted the keyboard and VoiceOver settings to match the participant's preferences (*e.g.*, disable auto-correct).

We then asked participants to compose paragraphs using either speech or the on-screen keyboard for input. When composing with speech, participants used the DICTATE button on the keyboard, located to the left of the SPACE key (shown in Figure 18). They could use the keyboard when editing the ASR output, but we required them to use speech for the initial composition process. Participants composed text on a simple website we developed that included only a prompt, a *textarea* widget, and a *submit* button. When they completed a paragraph, they clicked *submit*.

We presented participants with short prompts such as, “Tell us about a book you read recently. What did you like about it?” In addition to telling them which input method to use (speech or keyboard), we gave them two guidelines:

Enter about 4 to 8 sentences in response to the prompt.

Use professional language, as though you were emailing a potential employer.

We hoped the first guideline would encourage participants to type reasonably long paragraphs, that would reveal more interesting behavior, and the second guideline would encourage them to write complete sentences with proper grammar. We recommended that participants review and edit their text as they normally would (with both input modalities). We wanted to study speech input for long and formal paragraphs because we aim to make it usable for a variety of contexts, not just short, casual text messages.

Our procedure differed from standard text entry studies in at least two ways. First, in standard text entry studies (*e.g.*, [11,61,72,88,94,100]) participants transcribe sets of phrases. This approach is not appropriate for speech input, however, because people speak differently when reading text. Speech recognizers are trained on conversational speech, not read-aloud phrases. Recognizers would be likely to produce more errors if participants read phrases out loud. Second, most text entry studies do not permit participants to edit text post-hoc (*i.e.*, insert, delete, or replace text after repositioning the cursor). Since post-hoc editing is a common part of the composition process, especially with dictation, we included it in our study design.

After a participant entered one paragraph with each modality, we decided whether to ask him or her to enter a second set of paragraphs, depending on the amount of time remaining in the study.

We counter-balanced the order of input methods and alternated methods between successive paragraphs for each participant (if he or she entered more than one paragraph). We concluded each study with a 15-minute semi-structured interview. The interview included open-ended questions about what participants liked and disliked about using speech for input. We also asked them to respond to three statements on a 7-point Likert scale (1 is **strongly agree**, 7 is **strongly disagree)**. The statements were:

1. Entering text with speech was *fast* compared to entering text with the on-screen keyboard.
2. Entering text with speech was *frustrating* compared with entering text with the on-screen keyboard.
3. I am *satisfied* with using speech to enter text compared to using the on-screen keyboard.

We recorded audio for each study, along with a video capture of the iPod Touch's screen. We mirrored the iPod Touch screen onto the researcher's computer using the built-in AirPlay [4] client on the iPod and a Mac application called AirServer [1].

Design & Analysis

Participants composed a total of 22 paragraphs, 11 with speech input and 11 with keyboard input. Only three participants entered two paragraphs with each method and the rest entered one paragraph with each method. The average length of a paragraph was 66.4 words (*SD* = 41.6).

The study was an 8x1 design, with a single factor *Method* with two levels, *Speech* and *Keyboard*. We calculated entry speed in terms of Words per Minute (WPM), using the formula [101]:

$$WPM = \frac{|T|-1}{S} \bullet 60 \bullet \frac{1}{5} \quad (6.1)$$

where T is the transcribed string entered, $|T|$ is the length of T, and S is the time in seconds from the entry of the first character to the final keystroke. We included post-hoc editing in the entry rate calculation, so S included the time taken to review and edit a paragraph.

We evaluated accuracy in two ways. First, we computed the error rate of the speech recognizer for all text entered with speech. Second, we computed the error rate of the final transcriptions for both speech and keyboard inputs. We measured the error rate in terms of the Word Error Rate (WER), a standard metric used to evaluate speech recognition systems [62]. The WER is the word edit distance (*i.e.*, the number of word insertions, deletions, and replacements) between a reference and transcription text, normalized by the length of the reference text. For evaluating recognition errors, the reference text is the offline, human-perceived transcript of the participant's speech. For evaluating the final, edited keyboard and speech text, determining the reference text is less straight-forward. Since we are uncertain of the user's intended entry (this was a composition and not a transcription task), it is difficult to identify errors. As such, we considered a word to be an error if it was (1) a misspelled word, (2) a non sequitur that made no sense in the context of the sentence, or (3) a clear grammatical error such as a repeated word, or a short sentence fragment. Since participants' input was conversational, classifying words as errors was relatively straight-forward, given the aforementioned categories.

We used two-sided t-tests to compare text entry rates, since the WPM was roughly normally distributed. For error rates, which were not normally distributed, we used Wilcoxon Rank Sums tests to compare the means between input methods.

6.3.2. Results

There was high variability in the speech input behavior among participants, so we explain some of the results in terms of individual performance. P5 and P6 did not know how to reposition the cursor, so their ability to edit was limited, impacting both speed and accuracy. P5's accuracy was also affected by her thick foreign accent, although she still used speech for input weekly. On the other hand, P3 had many years of experience with dictation systems on different platforms and spoke with no hesitation and clear diction.

Entry Time and Accuracy

The entry rate for speech input was much higher than for keyboard input. With speech input, participants entered text at a rate of 19.5 WPM (*SD* = 10.1), while they entered only 4.3 WPM (*SD* = 1.5) with the keyboard. As expected, this resulted in a significant effect of *Method* on WPM ($t_{(10)}$=-5.07, $p < 0.001$). The maximum entry rate was achieved by P3, at 34.6 WPM with speech, while the maximum speed for keyboard input was achieved by P7, at 6.7 WPM.

When inputting speech, participants spent most of their time reviewing and editing recognition errors. On average, this amounted to 80.3% (*SD* = 10.2) of the composition time. P1 spent 94.2% (about 10 minutes) of his time reviewing and editing the recognizer's output—more than any other participant. Figure 4 shows the amount of time participants spent dictating vs. reviewing and editing their speech input compositions. Each bar in the plot represents one composed paragraph (*i.e.*, one task).

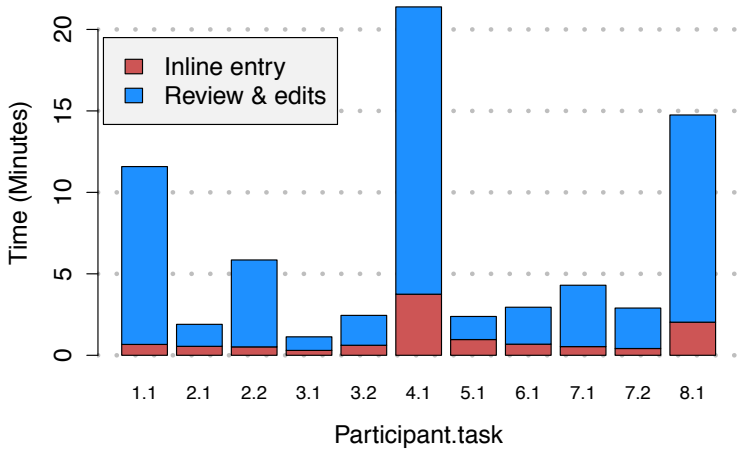


Figure 21. Time spent entering (red) and reviewing and editing (blue) text. Each column represents one composed paragraph (*i.e.*, a task).

While participants spent most of their time reviewing and editing text when using speech input, they spent little time on these activities when using the keyboard. On average, participants spent only 9.0% (*SD* = 11.7) of their time reviewing and editing keyboard output. This amounted to an average of 1.2 minutes of review and edit time with the keyboard compared to 5.4 minutes of review and edit time with speech input. Participants made most of their edits in the keyboard condition inline, by deleting text with the BACKSPACE key and reentering it.

The number of errors produced by the ASR largely determined the amount of reviewing and editing participants performed, and, in turn, their overall rate of entry. The average WER for the iOS ASR was 10.2% (*SD* = 10.4), ranging from 0% for P3 to 35.6% for P5. Figure 5 shows the entry rates of each paragraph input with speech as a function of the ASR's WER. As expected, there was a negative correlation, with an outlier point in the far right for P5.

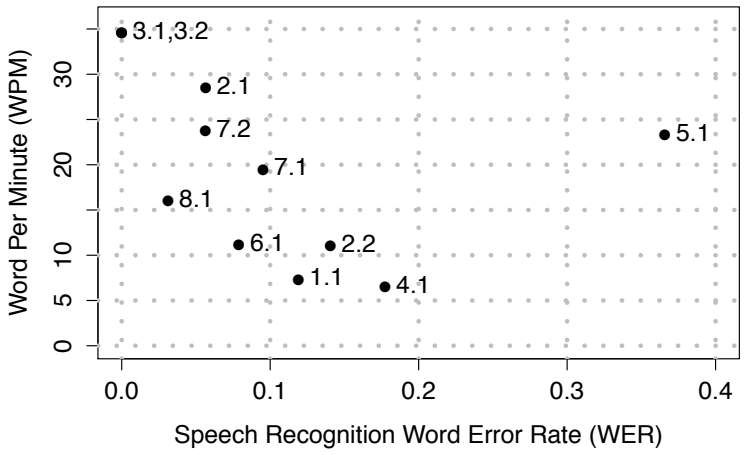


Figure 22. Entry rate vs. speech recognition error rate for speech input. The labels of the plotted points indicate the participant (1 – 8) and paragraph number (1 – 2).

Participants corrected most of the speech recognizer's errors, yielding a mean WER of 3.2% (*SD* = 4.6) for the final text input with speech. The mean WER for text input with the keyboard was slightly higher, at 4.0% (*SD* = 3.3). There was no significant effect of *Method* on the WER of the final text (W = 77, *n.s.*).

Reviewing and Editing Text

In this section, we describe the reviewing and editing techniques that participants used when composing paragraphs. After entering several sentences, most participants reviewed their text by making VoiceOver read it word by word. If a word did not sound correct, they read it character by character. To do this, participants used the VoiceOver rotor which enables users to read text one unit at a time. A unit can be a line, word, or character. To move the cursor from one unit to the next, a user swipes down on the screen. The user can also swipe up to move to the previous unit. In this way, participants moved the VoiceOver cursor around the text in the text area to

verify their input. When reviewing, participants often read words several times, then iterated through their characters.

Some ASR errors were difficult to detect using the VoiceOver text-to-speech because they sounded similar to the user's original speech. For example, P1 dictated the phrase "lost my sight," but the speech recognizer output the phrase, "lost my site." P1 did not detect this error. Another participant, P6, said, "you can hike," but the speech recognizer output, "you can't hike." P6 did not detect the error either. Some grammatical errors were also not detected in the review process. VoiceOver's text-to-speech does not differentiate upper- and lower-case letters, so there were some uncorrected case errors. The composed paragraphs also included extra spaces between words and sentences that participants did not seem to be aware of.

We observed that VoiceOver did not alert participants to passages that were recognized with low-confidence. Occasionally, the iOS speech recognizer underlined a word or phrase that was recognized with low confidence, and would present the user with alternative recognition options if the phrase was touched. This information was not accessible. Also, we observed that VoiceOver communicated punctuation marks in the text in a subtle manner, by varying prosody and pausing. This made it difficult for participants to detect punctuation marks in the recognized text when reviewing it.

The review process took longer than we expected, but participants spent much more time editing the ASR output. Figure 23 shows the number of edits participants performed for each paragraph input with speech. The total number of edits performed during speech input tasks was 96. Edits included inserting, deleting, and replacing characters, words, or strings of words. Only 15 (15.6%) of these edits were done using speech input. Although inputting speech was much faster

than using the keyboard, participants preferred using the keyboard to edit the ASR output. Several participants inserted words or phrases with speech but deleted erroneous text with the keyboard's BACKSPACE key. Also, there was no way to move the cursor using speech commands, so participants preferred to continue using touch to make character- or word-level edits.

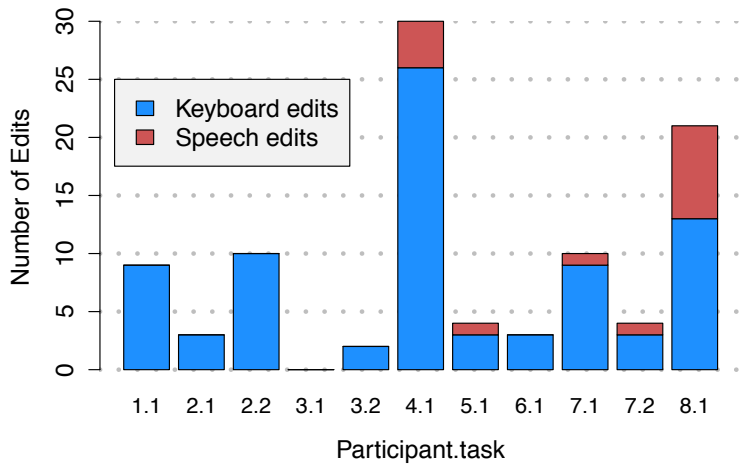


Figure 23. Number of edits performed with each modality for each composed paragraph (*i.e.*, task).

We observed three editing techniques for speech input in our study. We describe them as follows.

Hone in, delete, and reenter. The first technique was the most common and was used by all participants except P5 and P6 who did not know how to use VoiceOver gestures. This technique involved (1) moving the cursor to a desired position using the VoiceOver gestures describe previously, (2) using the BACKSPACE key to delete unwanted characters, words, or even phrases, and finally (3) to enter characters using the keyboard.

Hone in, select, and reenter. The second editing technique seems more efficient than the first, but was only used by P8. It involved (1) using VoiceOver gestures to move the cursor to a desired position, (2) choosing the EDIT option from the VoiceOver rotor, (3) selecting the word (one of the EDIT options, along with COPY and PASTE), and (4) re-enter the word with the keyboard. P8 sometimes re-entered the word with speech input. This technique required fewer key presses than "hone in, delete, and reenter."

Delete and start over. P5 and P6 did not know how to reposition the cursor, so they used this technique. They deleted entered text with the DELETE key, starting from the end (where the cursor was located by default) and re-entered with speech. They both said that they often used speech for short messages, and deleted everything and started over if there was a recognition error. This was the least flexible and, most likely, the least efficient editing technique for longer messages.

Qualitative User Feedback

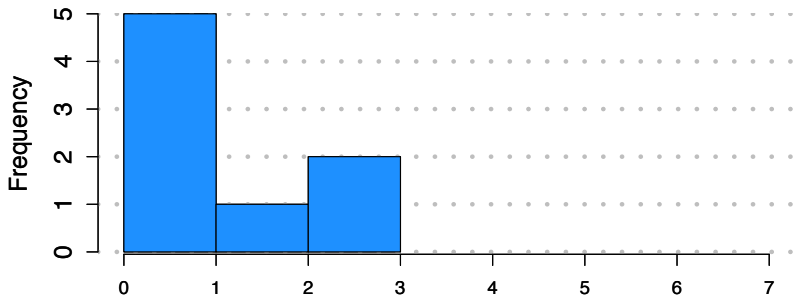
Two of the 8 (25%) participants in our study preferred keyboard input over speech input. These were P1 and P5, who were the only two participants who used speech weekly, rather than daily. All participants mentioned speed as the primary benefit of using speech for input. P1 and P5 preferred the keyboard because of the challenge of editing: P5 said she knew there were mistakes and didn't know how to fix them. P1 said that "although [with] speech you can get more volume of text in there…but my weakness is efficiently editing. that's the downside of speech."

P3 and P8 said it was easy for them to express their thoughts verbally, having had prior experience with dictation systems, unlike P1 and P2 who found it more difficult. P8 also said that

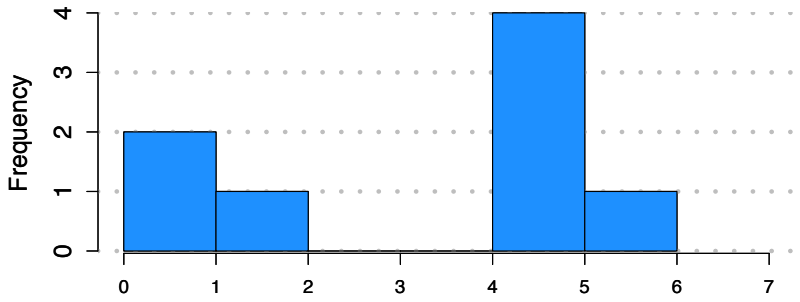
speech input helped him avoid spelling mistakes—auto-correct was not sufficient for correcting spelling mistakes.

> *I rely on dictation more because my spelling is not the greatest and feel like can compose more of coherent sentence verbally than by writing, especially on an iOS device where it's difficult to correct mistakes. On the computer, you can see with spell check and correct. but on iOS device it's easier not to have to worry about it.*

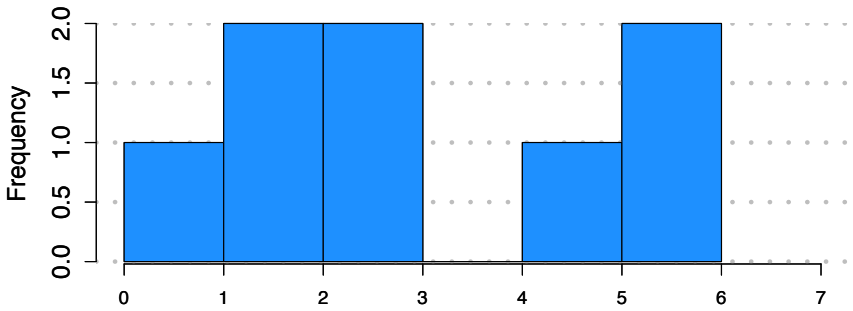
While most preferred speech, all participants found certain aspects of speech input frustrating. All participants except P3 cited editing as a source of frustration. P8 would like to be able to do "inline" editing as he spoke. P7 wanted an easier way to edit with speech rather than using the keyboard, which she did for most edits. P7 and P3 mentioned the problem of dictating words that were out-of-vocabulary, such as names. Figure 7 shows Likert scale responses to three statements from the interviews at the end of the study sessions, showing varying levels of satisfaction and frustration. Figure 7 also shows that all participants felt inputting text with speech was much faster than with the keyboard.



Speech is fast compared to the keyboard. Mean = 1.6 (SD = 0.9).



Speech is frustrating compared to the keyboard. Mean = 4.8 (SD = 2.1).



I'm satisfied with speech compared to the keyboard. Mean = 3.5 (SD = 1.9).

Figure 24. Responses to three statements on a 7-point Likert scale (1 is strongly agree, 7 is strongly disagree).

6.4. Discussion

Our study showed that speech input is an efficient entry method for blind people compared to the on-screen keyboard, yet it is impeded by the time required to review and edit ASR output. People can speak intelligibly at a rate of about 150 WPM [97], but the average entry rate of blind people using speech in our study was just 19.5 WPM. Nonetheless, this was comparable to the entry rate of sighted people using the on-screen keyboard of a smartphone, as found in prior work [23]. Furthermore, we found that the error rate of speech input was no higher than that of keyboard input for participants in our study. It is important to note, however, that we measured accuracy

only in terms of the WER, which does not necessarily correlate with the intelligibility of text [64]. The WER penalizes equally for small and major errors in a word, but it is the standard measure for evaluating ASR accuracy.

Six of the eight participants (75%) preferred speech over the keyboard because of speed, but all participants faced challenges when using speech input. Editing was the primary challenge; participants spent the majority (80%) of their time editing the text output by the recognizer. Surprisingly, their most common editing technique was highly inefficient in terms of keystrokes. Participants deleted characters with backspace and then re-entered them with the keyboard. It was unclear why they did not select whole words to replace them, or use speech for editing more than the keyboard. Perhaps some participants did not know how to select whole words with VoiceOver. They may have preferred to edit text with the keyboard because it was more predictable, preventing additional errors.

Study responses were less positive than our survey responses were. This was probably because, in our study, we asked participants to enter paragraphs that were longer and more formal than many smartphone communications. For example, a text message input by a survey participant was probably less than four sentences long and not as formal as an email that one would write to a potential employer (referring to the guidelines we gave participants in the study). Speech is currently better suited for short, casual messages, probably because of the difficulty of correcting and identifying errors. We believe the research community should facilitate the process of correcting content and grammar to make speech input more versatile.

Our study also uncovered interesting keyboard input behavior with VoiceOver. Since this was not our focus, we did not document challenges with keyboard input rigorously, but observed

several interesting trends. Surprisingly, some participants did not use the auto-correct feature, which could have improved their speed and accuracy. They found it difficult to monitor and dismiss auto-correct suggestions. We also observed that VoiceOver did not communicate punctuation clearly, and some minor grammatical issues, such as extra spaces between words, were only noticeable when reviewing text character by character. VoiceOver had a setting in which it speaks punctuation marks, but not one participant used this setting. VoiceOver also did not communicate misspelled words that were visually identified with an underline. Enabling participants to more easily identify punctuation and grammar and spelling issues would likely improve efficiency and composition quality for both keyboard and speech input.

Throughout the chapter, we have compared speech input with the de facto standard accessible input method for touchscreens: on-screen keyboard input with VoiceOver. However, there are input alternatives that are commonly used by both blind and sighted people that should be considered when evaluating speech. Several study participants used a small external keyboard with hard keys; one participant used the keyboard on his Braille display, which connected to his iPhone; one participant used the on-screen input method Fleksy [30], one of many gesture-based text entry methods (see Related Work for others). These alternatives are more private than speech, and probably more reliable in noisy environments. It would be interesting to compare these methods to speech in the future.

6.5. Challenges for Future Research

We distill our findings into a set of challenges for researchers interested in nonvisual text entry. These challenges can be incorporated into both speech and gesture-based input methods.

1. **Text selection –** a better method for nonvisual selection of text. This can also include other edit operations, such as cut, copy, and paste.
2. **Cursor positioning** – an easier way to move a cursor around a text area; enable a user to easily hone in on errors.
3. **Error detection** – an easier way to detect errors such as spelling mistakes, letter case errors, and low-confidence ASR output.
4. **Auto-correct** – a study of how well auto-correct works for nonvisual use, and a way to make it more effective.

Chapter 7.
SpeakNSwipe: Leveraging Alternative Results in Eyes-Free Dictation

7.1. Motivation

Speech recognition has improved significantly in recent years. Wearable devices, which are increasing in popularity, commonly use speech for input because it is fast and natural. A relatively small and inexpensive device can have a microphone, speaker for output, and a wireless radio for online speech recognition. However, the input is usually restricted to speech commands or questions: “what’s the weather today?” or “Call bob.” While this is suitable for performing a specified set of tasks in an interface, it is not a general input method. People still rely on keyboards to enter, edit, and format text for various applications (*e.g.*, composing an email).

To handle a wide variety of text entry tasks on mobile and wearable devices, we propose and coin the term *eyes-free dictation* (EFD). We define EFD as the interacting process of entering, reviewing, and editing text on a computing device using primarily speech for input while not requiring a screen for output. Unlike speech commands, EFD enables people to input text of arbitrary content and length. Even as speech recognizers continue to improve, users will need to review and edit text for certain compositions as they do when typing or handwriting.

Surprisingly, there are no EFD methods in research and industry today, to our knowledge. One related example is the iOS dialog system Siri’s eyes-free mode, which is similar to Google Now.

The eyes-free mode allows the user to dictate and review short messages through an eyes- and hands-free dialog. The interaction is as follows:

> *User: Send a text message to Bob, "I'm running late."*
> *System: Ready to send it?*
> *User: Review*
> *System: Your message to Bob says, I'm running late. Would you like to send, cancel, review, or change it?*
> *User: Send it.*

If the message was not recognized correctly, the user can try to speak the full message over again. This may be convenient to enter short, casual text messages with such a system, but probably not longer, more formal compositions. We don't consider Siri's eyes-free mode EFD because it doesn't allow the user to edit the composed text, only reenter it.

7.2. SpeakNSwipe

SpeakNSwipe is an EFD method that allows users to enter, review, and edit text on devices with no screen and five input sensors. In this section, we describe the design of the system and the algorithms it uses to process output from a commercial speech recognizer with a minimal API.

7.2.1. Design

The design of SpeakNSwipe was motivated by the work described in Chapter 6, where we observed blind people using the speech recognizer on iOS to compose paragraphs with the VoiceOver screen reader. Dictation with VoiceOver served as an unintended example of EFD. Based on the challenges blind people faced, we developed the following design guidelines for SpeakNSwipe:

1. No visual output and a small number of nonvisual inputs. This follows the vision of EFD.

2. Facilitate error detection. It can be difficult to hear errors when they are spoken from a speech synthesizer because the recognizer is designed to output phrases that *sound like* what the user said.
3. Fast error correction. Efficient error correction has always been a challenge in dictation systems.
4. Incorporate a commercial speech recognizer. We wanted the system to benefit from a state-of-the-art speech recognizer since the recognition is a core part of the overall experience.

Following Guideline 1, we chose five non-visual non-speech controls: LEFT, RIGHT, UP, DOWN, and CENTER. These inputs can be implemented as gestures on a touchpad or touchscreen, or physical buttons on a device. SpeakNSwipe has no visual feedback; instead, it speaks text and plays sounds to the user.

To enter text with SpeakNSwipe, the user begins speaking to the system as she would with existing dictation software: she performs a long-press on CENTER and starts speaking. She can speak for as long as she wants, explicitly punctuation marks like "period" and "comma." When the user finishes speaking, she taps CENTER or waits for the SpeakNSwipe to detect that she has stopped. For example, one user in our formative studies said, "Hey comma do you need it today question mark." The captured audio is sent to a third-party speech recognizer (Guideline 4).

After receiving recognition results from the speech recognizer, SpeakNSwipe (1) detects uncertain words or phrases that may contain an error, and (2) divides the text into *certain* and *uncertain* segments. Segmenting the text aims to facilitate comprehension, error detection and editing as the user listens and processes only short bursts of synthesized speech (Guideline 2). A

speech recognizer typically returns several transcriptions of the entire audio file: a top result (*i.e.*, the speech recognizers "best guess"), and several alternative hypotheses (*i.e.*, an N-best list). In our example from the previous paragraph, the speech recognizer returned, "Hey, do you need it a day?" as the top result. As the alternates, the speech recognizer returned, "Hey, do you need today?," "Hey, do you need it today?," and "Hey, do you need to day?" Intuitively, we see that the words "a day" in the top result are *uncertain*—they are likely to be incorrect, since the recognizer returned several different hypotheses for that utterance. SpeakNSwipe finds one or more continuous uncertain words and groups them into a *segment*. It then links corresponding segments in each alternative hypotheses.

After creating uncertain segments, SpeakNSwipe splits the transcription further into *certain* segments. Certain segments do not have alternative recognition results and consist of phrases that are delimited by commas or end-of-sentence punctuation marks *(e.g.*, a period or a question mark), or the start of an uncertain segment. This segmentation aims to facilitate comprehension of the synthesized speech since the user is able to process the transcription more easily one phrase at a time. Thus, in our example, we have two certain segments followed by one uncertain segment: "Hey," "do you need", and "a day | today | it today | to day?"

The user reviews the transcribed text by navigating from one segment to another using the LEFT and RIGHT controls. SpeakNSwipe speaks each selected segment. If the segment is uncertain, SpeakNSwipe also plays a *ding* sound effect to alert the user to a potential error. The user can replace the uncertain segment with alternatives using the UP and DOWN controls. In our example, the user can quickly replace "a day" with "today" by tapping DOWN once. Honing in

on uncertain words and presenting alternatives aims to help the user detect errors and minimize edit operations (Guideline 3).

If a user wants to make an edit and does not find an appropriate alternative result, she can re-speak a segment. SpeakNSwipe sends the audio file for a segment to the speech recognizers and receives, once again, a top result and a list of alternative hypotheses. Instead of replacing the previous segment with the new results, SpeakNSwipe combines the previous results for the segment with the new ones, and performs its error detection segmentation algorithms for the segment. The user is then able to choose among all recognition results from the previous and latest recognition to avoid spirals of errors.

To illustrate how the system handles re-speaking, suppose the user in our example changed her mind and wanted to edit her original sentence to, “Hey, will you need it by today?.” She would navigate to the “do you need” segment, long-press CENTER to initiate audio recording, and speak the words, “will you need.” The speech recognizer returns the following results: “willy need,” and “will I need.” Neither is correct. SpeakNSwipe would then take the previous recognition results for the segment—in this case, there’s just one: “did you need”—and combine them with the new results. It then detects likely errors and segments the text yielding three new segments: “willy | will | did,” “ you | I,” and “need.” Overall, SpeakNSwipe now presents the user with five segments:

1. [Certain] Hey
2. [Uncertain] willy | will | did
3. [Uncertain] you | I
4. [Certain] need

5. [Uncertain] today | it today | to day

Using the controls, the user can quickly review the segments and their alternative hypotheses to construct her revised phrase. Note that the segmentation process allows the user to construct phrases that were not output by the speech recognizer directly.

7.2.2. SpeakNSwipe Algorithm

As described in the previous section, the SpeakNSwipe algorithm performs two main tasks: it detects uncertain words and then divides the output into certain and uncertain segments. The algorithm takes as input a set of transcriptions for the recorded audio file that include a top result and zero or more alternative results. The algorithm outputs a list of segments, where a segment includes a sequence of one or more words, a flag indicating whether it is uncertain or not, and a list of zero or more alternate sequences. Only uncertain segments have alternatives.

Detecting Uncertain Words

We observed that detecting uncertain words can be done with a multiple sequence alignment (MSA) [28] of the recognition results, commonly used by biologists to compare genetic sequences. In MSA, three or more sequences are aligned by inserting some number of gaps between their bases. The goal is to optimize the similarity of bases across the alignment columns, according to some similarity metric. In our application, we think of each transcription as a sequence and each word as a base in that sequence. We use the edit distance to measure similarity among words, since words that sound alike usually have similar spellings. Below is an example of an alignment of the transcriptions from our example. The highlighted columns have a

mismatch: they include at least one word that is not the same as the others in the column. The words in these columns are the uncertain we want to detect.

Top result:	Hey,	do	you	need	it	a	day?
Alternative 1:	Hey,	do	you	need	[GAP]	[GAP]	today?
Alternative 2:	Hey,	do	you	need	it	[GAP]	today?
Alternative 3:	Hey,	do	you	need	to	[GAP]	day?

Unfortunately, MSA is NP-hard and the optimal algorithm has an asymptotic runtime of $O(n^k)$, where n is the length of the longest sequence in words and k is the number of transcriptions (*i.e.*, rows in the alignment). So we developed a new algorithm to produce an alignment that allows us to detect uncertain words. This algorithm is more efficient than the optimal MSA algorithm. It performs $2k$ pairwise alignments. Since each pairwise alignment runs in $O(n^2)$ time, our algorithm runs in $O(kn^2)$ time, where n is the length of the longest sequence in words and k is the number of sequences. We use the Needleman-Wunsch algorithm [68] for all pairwise alignments.

To produce an alignment of all transcription, our algorithm does two things. First, it merges the transcriptions into a single sequence that includes all certain words but has placeholders for uncertain words. We denote the placeholders with the lablel *[uncertain]*. The merged sequence is computed iteratively. The algorithm performs a word-level pairwise alignment of the first two transcriptions. It creates a merged sequence from the alignment by copying the words that match in each column and inserting a placeholder for each column that has a mismatch or a gap. Using our example, the algorithm begins by performing an alignment of the top result and the first alternative result. As before, columns with mismatches or gaps are highlighted.

Top result:	Hey,	do	you	need	it	a	day?
Alternative 1:	Hey,	do	you	need	[GAP]	[GAP]	today?

Then the algorithm constructs a merged sequence from the pairwise alignment:

Merged 1:	Hey,	do	you	need	[uncertain]	[uncertain]	[uncertain]

Next, the algorithm performs a pairwise alignment between the current merged sequence and the next transcription:

Merged 1:	Hey,	do	you	need	[uncertain]	[uncertain]	[uncertain]
Alternative 3:	Hey,	do	you	need	[GAP]	[GAP]	today?

It then merges the aligned transcriptions into one sequence. In this case, the new merged sequence is identical to the previous one:

Merged 2:	Hey,	do	you	need	[uncertain]	[uncertain]	[uncertain]

The current merged sequence is then aligned and merged with the next alternative result and this process continues until all *n* transcriptions have been used to form a single merged sequence. In our example, the final merged sequence happens to be the same as the first one:

Merged:	Hey,	do	you	need	[uncertain]	[uncertain]	[uncertain]

The second stage of the SpeakNSwipe algorithm takes the merged sequence as input and outputs an alignment of the merged sequence and the *n* transcriptions. Because of the way the merged sequence was constructed, we don't have to perform a MSA. Instead, we can compute an

alignment matrix of all the transcriptions by performing a pairwise alignment between each transcription and the final merged sequence. This involves *n* pairwise alignments: the merged sequence and the top result, the merged sequence and the first alternative, the merged sequence and the second alternative, etc. When we present the results of each of these pairwise alignment, an alignment matrix of all the sequences emerges.

Merged:	Hey,	do	you	need	[uncertain]	[uncertain]	[uncertain]
Top result:	Hey,	do	you	need	it	a	day?
Alternative 1:	Hey,	do	you	need	[GAP]	[GAP]	today?
Alternative 2:	Hey,	do	you	need	it	[GAP]	today?
Alternative 3:	Hey,	do	you	need	to	[GAP]	day?

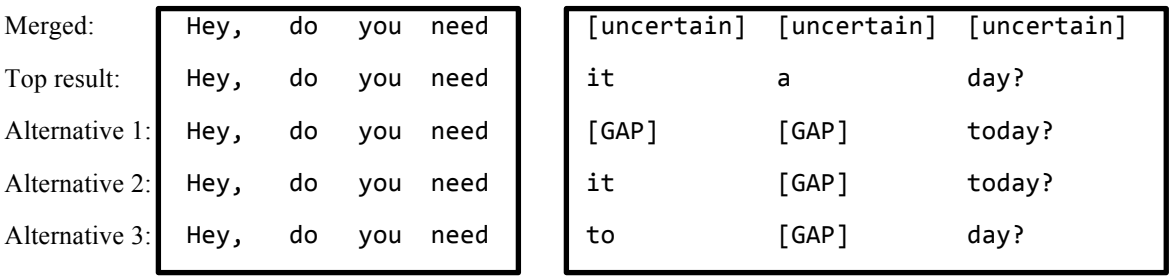
Thus, we've constructed a multiple sequence alignment by performing *2k* pairwise alignments and *k* sequence merges. Now, the algorithm flags the words in each column that has "*[uncertain]*" in it. These words are our uncertain words. We have thus detected all uncertain words in the transcriptions.

Segmenting the text

After detecting uncertain words, SpeakNSwipe segments the text into certain and uncertain segments. The segmentation algorithm takes as input the alignment matrix produced in the previous section. The first segment begins with the first column of the alignment matrix. If the first column is flagged as uncertain, the algorithm includes all the following columns in the segment until it reaches a column that is *not* marked as uncertain. In other words, all columns in an uncertain segment are uncertain. If the first column is not flagged as uncertain, subsequent columns are included in the segment until the algorithm reaches (1) an uncertain column, or (2) a

comma, period or other punctuation mark that delimits a clause. The algorithm continues to segment the remaining columns iteratively.

In our example, the first segment is certain and includes three columns. The second and final segment is uncertain, and includes the last three columns.



Segments that contain certain columns are tagged as certain and, by construction, have the same words in each row. Segments that contain uncertain columns are tagged as uncertain and have at least two different words in each column.

The runtime of the segmentation algorithm is $O(kn)$, where n is the length of the longest sequence in words and k is the number of results. Thus, the total runtime of the SpeakNSwipe algorithms is $O(kn^2 + kn)$.

After the segments have been created, the user can move from one segment to the other and iterate between the alternative results for each segment.

7.3. Evaluation

We evaluated two variants of SpeakNSwipe against a control technique in a laboratory study. Since there are no existing methods for editing speech input without looking at the screen, we

wanted to observe users' behavior and reactions to SpeakNSwipe and its features. In particular, we were interested in the following questions:

- How quickly and accurately can users enter text with SpeakNSwipe?
- How effectively can users detect and correct errors with SpeakNSwipe?
- What are users' subjective reactions to SpeakNSwipe?

Since there is not baseline for eyes-free editing to compare SpeakNSwipe to, we designed a control method that, like eyes-free Siri, only allows users to Respeak and entire paragraph.

7.3.1. Method

Participants.

We recruited 9 participants (4 female, 5 males) with an average age of 25 years (age range: 22 – 36). All participants had smart mobile devices: five had Android phones and four had iPhones. Two participants spoke English with foreign accents. None of the participants used dictation on their phones on a regular basis. Most said that it was not accurate and correcting errors took too long. Several participants used Siri or Google Now weekly. Participants were recruited through a university mailing list.

Procedure.

Each participant entered paragraphs with each method during one hour-long session. We counterbalanced the order of methods for each participant using a Latin Square. Before entering paragraphs with a method, we explain how the method works to the participant and allowed them to complete one practice task. Then we asked participant to transcribe 5 paragraphs and compose 2 additional paragraphs. Most participants only had enough time to compose one paragraph with

each method. Compositions allowed us to measure realistic text entry behavior, while transcriptions allowed us to reliably measure errors since we knew what the target words were.

For transcription tasks, we chose paragraphs from the Enron Mobile email dataset [107], that included real emails that were entered on mobile devices. The paragraphs were 4-8 sentences long. For composition tasks, we presented participants with benign prompts, such as "Tell us about a movie you saw recently." We instructed participants to compose a paragraph that was 4 – 8 sentences long. For all paragraphs, we told participants to enter text "as quickly and accurately as possible," as is standard in text entry studies.

After participants completed their tasks, we conducted a brief interview. We asked participants what they liked and disliked about each method. We then asked them how frustrated and satisfied they were with each method using responses on a 7-point Likert scale. We emphasized that we were evaluating the speech input systems as a whole rather than just the speech recognition component, which was identical for all three methods.

Apparatus

We developed prototypes for three techniques:

- **SpeakNSwipe (SNS),** as described in the Design section.
- **SpeakNSwipe-Word (SNS-Word).** We included a variant of the SpeakNSwipe technique in the study. SNS-Word used the same algorithm to identify and align uncertain words but it segmented the text into words rather than phrases. We were interested to see whether segmenting the text into phrases was in fact the most effective design.

- **Control**. The control technique allowed users to speak, review, and re-speak their entire input. This technique was based on the experience of using eyes-free Siri to compose message.

All techniques used Google's chromium ASR service. Participants used a Nexus 5 device.

Design and Analysis

Participants entered a total of 135 transcribed paragraphs and 36 composed paragraphs. The mean number of words was 32.5 in the transcribed paragraphs and 60.1 in the composed paragraphs.

We compared speed and accuracy across the three methods for transcribed and composed paragraphs separately. We measured entry speed in words per minute (WPM) [101]. We constructed linear mixed effects models to predict WPM as a function of the fixed effect of method and random effects of participant and speech recognition error rate. For transcribed paragraphs, we measured accuracy in terms of word error rate (WER), as is standard in speech recognition literature. There was no reliable way to measure accuracy for composed paragraphs, since participants often changed what their intended input while editing. Since the WER was not normally distributed, we compared the WER among methods using Kruskal-Wallis tests. As is common, Likert-scale responses were not normally distributed, so we used Kruskal-Wallis tests to analyze those as well.

7.3.2. Results

Entry Speed

Participants entered text faster with the Control method than with SpeakNSwipe and SpeakNSwipe-Word while transcribing paragraphs. The difference among the three methods was significant for transcribed phrases ($F(2,115)=27.5$, $p < 0.001$). Post-hoc tests revealed that the differences between the Control and SpeakNSwipe ($z=-5.4$, $p<0.001$) and the Control and SpeakNSwipe-Word were significant ($z=-7.3$, $P<0.001$), but there was no significant difference between SpeakNSwipe and SpeakNSwipe-Word ($z=-2.1$, *n.s.*). Figure 25 shows the means and standard deviations of entry rates for transcribed phrases using the three methods.

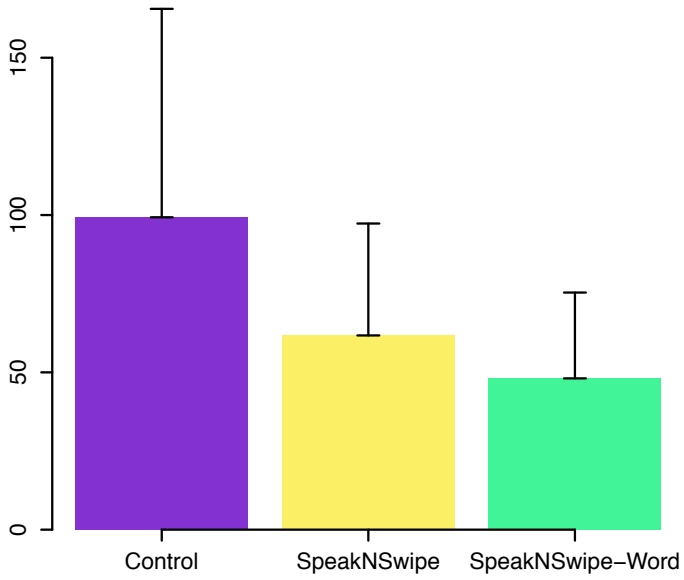


Figure 25. Mean entry rates (in words per minute) for transcribed paragraphs with the Control, SpeakNSwipe, and SpeakNSwipe-Word.

Accuracy

We measured the accuracy of the speech recognizer and the accuracy of the participants' final edited phrases. Participants' experience using all methods depended heavily on the accuracy of the speech recognizer, which varied greatly. Overall, the mean speech recognizer WER for all transcribed phrases was 7.4% (*SD*=11.6). Participants corrected many of the speech recognizers' errors, so the mean error rate of all submitted transcribed paragraphs was 3.1% (*SD*=3.7). Surprisingly, there was no significant difference in submitted paragraphs' error rates across methods (χ^2=1.87, *n.s.*). Figure 26 shows the mean WER for the three methods.

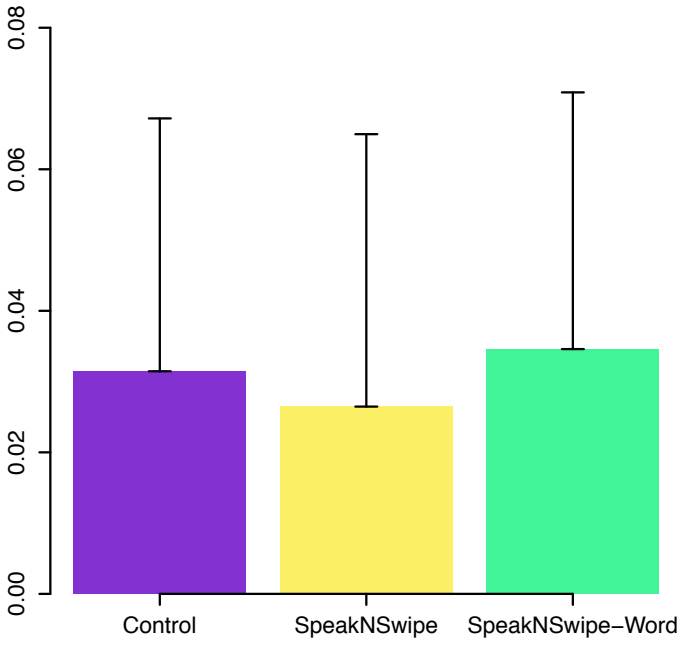


Figure 26. Word Error rate in transcribed paragraphs for the three methods used in our study.

Subjective Experience

Most participants preferred SpeakNSwipe overall, followed by SpeakNSwipe-Word. Even though the Control method was faster and about as accurate as the other two methods, participants said the Control was more frustrating and less satisfying. Interestingly, people *perceived* SpeakNSwipe and SpeakNSwipe-Word to be faster than the Control. They felt SpeakNSwipe was fastest but SpeakNSwipe-Word gave them more control than SpeakNSwipe. However, most participants did not like to gesture so many times when using SpeakNSwipe-Word. Ironically, several participants noted that one advantage of using the Control was that it motivated them to speak slowly and carefully to avoid speech recognition errors all together.

We asked participants to rate their levels of frustration and satisfaction with each method on 7-point Likert scales (Figure 27 and Figure 28, respectively). On a scale of 1 to 7, where 1 was "not frustrated" at all and 7 was "very frustrated," participants rated the Control method 5.8 on average (SD=1.1). Meanwhile, they rated SpeakNSwipe 2.9 (SD=0.8) and SpeakNSwipe-Word a 3.2 (SD=1.1) on average. These differences were significant (χ^2=16.1, $p<0.001$). Participants said they didn't like to repeat an entire paragraph to correct several errors. In several cases, we observed that re-speaking in the Control method introduced new speech recognition errors that participants found very frustrating.

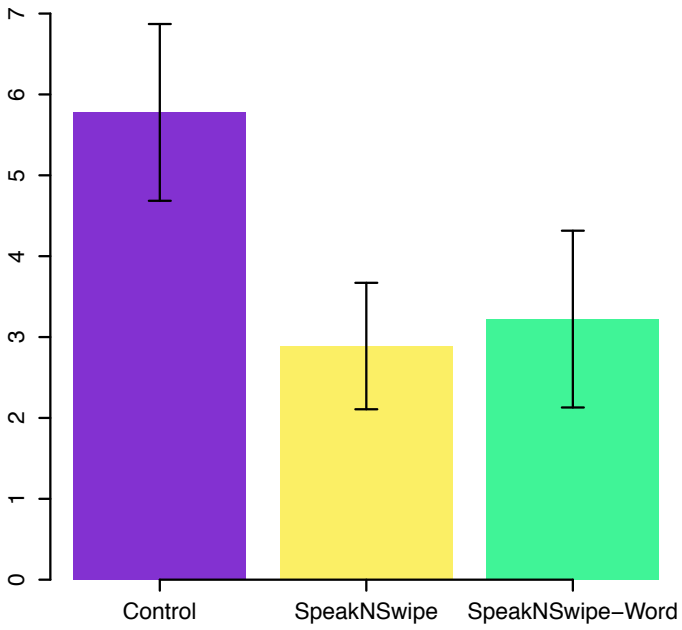


Figure 27. Participants responses to the question, "How frustrated were you by each method?" Responses are given on a 7-point Likert scale, where 1 is "not frustrated at all" and 7 is "very frustrated." Lower scores are better.

In terms of satisfaction, participants rated the Control a 2.8 on average (*SD*=1.8), where 1 was "not satisfied" and 7 was "very satisfied." In contrast, they rated SpeakNSwipe and SpeakNSwipe-Word 5.4 (*SD*=0.9) and 4.7(*SD*=1.2), respectively. This difference was statistically significant (χ^2=10.6, p=0.005). Participants all recognized that the ability to edit messages that were longer than one sentences was "useful." Several said that reviewing their input phrase by phrase was convenient, but they would like to only repeat an erroneous word instead of an entire phrase.

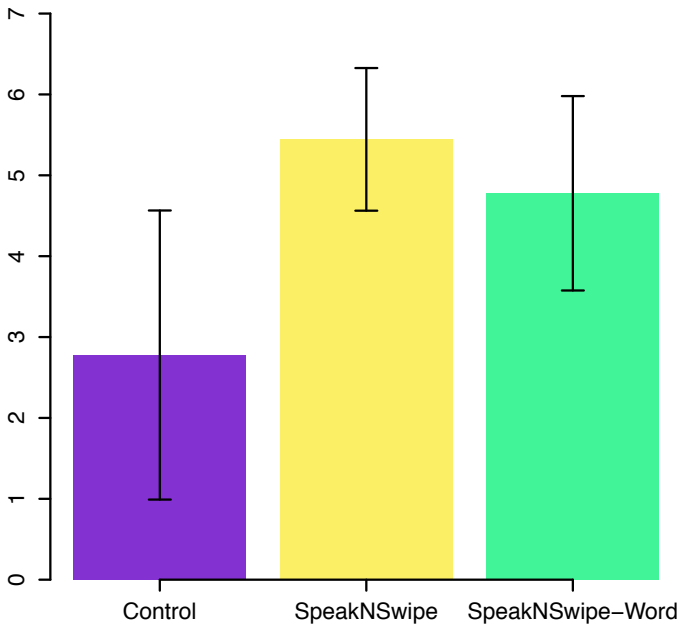


Figure 28. Participants responses to the question, "How satisfied were you with each method?" Responses are given on a 7-point Likert scale, where 1 is "not satisfied at all" and 7 is "very satisfied." Higher scores are better.

7.4. Discussion

Overall, the study results indicated that participants had a strong preference for SpeakNSwipe over SpeakNSwipe-word and the Control despite the fact that the Control was faster. What caused this contradiction between the performance results and users' comments? We describe our observations of users' behavior and responses throughout the studies that help to explain our findings.

One reason participants had high entry speeds with the Control was that, in some cases, participants submitted paragraphs without attempting to edit or correct errors to spare themselves frustration. They reviewed the paragraphs, detected errors, but chose to submit them anyway

because they didn't want to Respeak the entire paragraph. Meanwhile, with SpeakNSwipe and SpeakNSwipe-Word, participants attempted to correct errors, using more time for the composition overall.

If participants were spending more time correcting errors with SpeakNSwipe than the Control, why wasn't SpeakNSwipe more accurate than the Control? Recall that there was no significant difference in the WER among the three methods. There is no compelling explanation for this, but we do see that there are several causes for errors that do not relate to the methods used that add variance and cloud possible trends. In particular, we observed three causes for errors in submitted paragraphs that occurred with all three conditions.

First, the error rate of the speech recognizer varied greatly. Nearly 30 percent of submitted paragraphs didn't have any recognition errors, making our sample small and highly variable.

Second, sometimes participants did not detect errors in a sentence or paragraph, especially when SpeakNSwipe did not verbally highlight it. For example, when P1 reviewed a paragraph, the speech recognizer said "hand it" instead of "handed." P1 did not seem to detect this error and submitted the paragraph without attempting to correct it.

Lastly, in some cases participants were not able to correct an error. After several attempts to respeak, a participants could not get the speech recognizer to output the correct word. Since we did not allow participants to use another modality, they submitted the incorrect word and moved on. For example, this happened to several participants with the name "Irena," which appeared in one of the paragraphs from the Enron email dataset. We could have avoided this problem by choosing paragraphs from the datasets that did not have uncommon proper nouns, but we wanted

to make the experiment as realistic as possible and observe situations with out-of-vocabulary words.

7.5. Evaluating Speech Input Methods

We believe that we need better study methodology to avoid situations in which the performance results don't reflect users' experience and subjective responses. In particular, we make the following recommendations for conducting studies with speech input.

To ensure that participants don't "give up" and produce artificially fast entry rates, researchers should instruct participants to "enter text as accurately as possible." This differs from standard text entry studies where participants are instructed to enter text as "quickly and accurately as possible." Researchers can provide a monetary incentive for participants to follow these instructions with a compensation scheme where participants receive less money with every error they submit (this is common in psychology studies)

Researchers should control the error rate of the speech recognizer. This will reduce unwanted variance and help researchers focus on editing behavior. Controlling the speech recognizer error rate is an open problem. One approach is to build a Wizard of Oz system that allows a fast typist to input text with a system that introduces errors to that text at a certain probability.

We believe that studies of text entry performance will benefit from an error measure that better reflect human perception. WER is coarse: if a word has one incorrect character or the word is not present at all, the WER is the same. Also, errors in different words in the sentence carry the same weight. For example, consider the sentences: (1) "put that in the drawer," (2) "put that on the drawer," and (3) "put that in the dryer." Sentences (1) and (2) have the same WER as sentences

(1) and (3). However, sentences (1) and (2) are much more similar semantically than sentences (1) and (3). A user may choose to not correct the error in sentence (2) but would probably need to correct the error in sentence (3) to avoid confusion. A good error measure should reflect this difference in perception and resulting behavior.

7.6. Design Improvements

We have developed SpeakNSwipe, an eyes-free dictation method that interfaces with a commercial speech recognizer. SpeakNSwipe is the first method that enables people to edit speech recognition output in an eyes-free manner. Its designed to facilitate detection and correction of speech recognition errors and editing of phrases. We evaluated SpeakNSwipe against two methods: a SpeakNSwipe variant that we call SpeakNSwipe-Word and a Control method. Although participants entered text faster with the Control, they unanimously preferred SpeakNSwipe, finding it more satisfying and less frustrating.

Observing and interviewing our participants lead us to several direction for improving SpeakNSwipe. Participants felt that it would be useful to include features from the Control in SpeakNSwipe. In some cases, they wanted to hear the entire paragraph and Respeak the whole thing. Participants also wanted an efficient method to delete single words. Often the speech recognizer inserted unwanted words from unintentional utterances (*e.g.*, "uh"). In the current version of the system, participants had to respeak the phrase containing the unwanted utterance to remove it. However, participants wanted a "delete" operation that was similar to using a keyboard. Finally, we observed that our simple phrase segmentation heuristic caused some confusion when it failed to segment phrases properly. Participants like the idea of splitting a

paragraph into phrases, but wanted them to be more natural. In the future, we will investigate the possibility of using a parse [93] to segment text.

Chapter 8.
Conclusions and Future Work

In this chapter, I discuss the over-arching design principles and the lessons learned from conducting the work that was presented in this dissertation. I then conclude with directions for future research on the individual projects presented here and general work on eyes-free input on mobile devices.

8.1. Design Principles

Speed, accuracy, and learnability. In all of our inventions, we prioritized speed of entry, followed by accuracy, and lastly, the learnability of the technique. All of these common attributes are important, but we prioritized speed because current access technologies generally take so much longer to use than mainstream approaches. Accuracy was also important, although we observed that people had a relatively high tolerance for errors in many of their mobile communications (*e.g.*, text messages). Learnability, or the ease in which one can learn a new technique, was also important because the easier it is to learn a technique, the more likely users will be to adopt it. There are trade-offs among the three attributes: faster input is usually less accurate, and users may be willing to put more effort into learning a technique that will be much faster than what they currently use.

Enabling universal access. We aimed to design techniques that were usable by both blind and sighted people. We compared the performance of blind and sighted users when evaluating DigiTaps and found that their performance was comparable (see Chapter 5). In Perkinput, we used the Braille encoding which gives blind people an advantage because they are more likely to

know Braille. However, once a sighted person learns the encoding, there is no reason their performance would differ from a blind person's performance.

Using mainstream devices. A fundamental aspect of universal access is enabling people with disabilities to use mainstream devices. We designed all of our techniques so that they require only a multi-touch screen for input and speech and vibration capabilities for output. SpeakNSwipe can also be used on devices with no screen and only five buttons. During the time our work was conducted, these components were available on popular mainstream mobile devices.

No soft keys. The prevailing metaphor for touchscreen interaction is soft keys [66]. However, since soft keys have no tactile delineation, users must heavily rely on their vision to activate them. VoiceOver and TalkBack add an accessible "layer" over soft keys that make them somewhat accessible but this form of accessible interaction is much slower than using soft keys visually, as was originally intended. We designed our techniques in ways that did not use soft keys and did not rely on vision. Input Finger Detection in Perkinput and PassChords, simple multi-touch taps in DigiTaps, and simple directional swipes in SpeakNSwipe do not rely on a user's vision.

8.2. Lessons Learned

Effective feedback is critical. As we prototyped and iterated on the design of our techniques, we learned that it was critical to design effective feedback methods. Without prompt audio or vibration feedback from the device, people had difficulty interacting with a touchscreen with no visual feedback. Conversely, we were surprised that vibration feedback on its own was effective

for multi-touch input, without any audio or visual feedback. Participants were able to enter information with DigiTaps and PassChords relatively quickly and they were even able to discern and correct errors. To produce effective audio or vibration feedback, we found that it was important to produce a short sound or "buzz" promptly after input was registered. Lengthening or delaying a "click" sound or vibration slowed people down because they waited for the complete feedback before entering the next input.

Participant demographics matter. We showed the techniques we invented to many different people throughout the course of their development and observed a wide range of learning and performance abilities. A 22-year-old research assistant mastered Perkinput in 15 minutes while a retired 62-year-old with little touchscreen experience struggled with the technique after several half-hour sessions. Most of the users who participated in the Perkinput study were similar to this latter user: their average age was 55 and all had little to no experience with touchscreens. Young and tech-savvy people simply didn't have the time or desire to participate in a longitudinal study at the University of Washington campus. This lead to entry rates that were surprisingly low compared with those we saw in our formative studies with younger and more tech-savvy people.

When evaluating PassChords, we conducted a one-session study at the Department of Services for the Blind, where we found participants who had a wider age range and more experience with technology. In fact, we required that people use a smartphone on a daily basis to be included in the study. This lead to faster entry speeds than those in our Perkinput study. If we had conducted the Perkinput study with participants from the PassChords study, we would have gotten much higher entry rates.

Because we've seen how much performance can vary depending on participants' demographics and experience, we believe that the absolute numbers attained from studies are not representative of the entire population. Rather, we think the most important information to consider in a study are (1) the comparative performance of a new technique and an appropriate baseline, (2) the subjective experiences of participants, and (3) the background and demographics of the participants.

It is important to incentivize participants to do well in a study. We slightly varied the methodology of the studies described in this dissertation and found that small changes can affect participants' level of engagement and performance. In the Perkinput study, we asked participants to enter phrases for a set amount of time with each method. We didn't give them any feedback about how well they were doing. Most participants seemed incredibly bored, entering text slowly, trying to keep their focus. By contrast, in the DigiTaps study, we asked participants to enter set number of inputs for each method. At the end of each set, we told them what their speed and accuracy was. Participants found this study very engaging. They were eager to hear how well they performed and wanted to improve their measures. We believe that the tweaks in the study methodology resulted in a much better (and less painful) study for everyone involved. It also resulted in more realistic results, since people are generally engaged in their text entry task and eager to complete it quickly.

8.3. Future Work

8.3.1. Gestures-Based Methods

We have presented three gesture-based input methods: Perkinput, DigiTaps, and PassChords. Among these three, Perkinput promises to have the most impact on the daily experiences of blind people using mobile devices. As such, we have many ideas for future research to improve and study Perkinput.

First, we would like to deploy Perkinput and study users' performance in the wild. We expect that performance in real-life settings, where users compose rather than transcribe text, will be faster than that in our lab study. People will have a stronger incentive to enter text quickly and will accumulate more hours of practice over time. After we collect input data from dozens of participants, we plan to run simulations on this data to try and improve the Input Finger Detection (IFD) algorithms. Perhaps a more complex model of the hand is needed to reduce errors, or perhaps a simpler algorithm will produce similar results to the current version of IFD. Even if we improve the IFD algorithms, there will still be errors in users' input streams. So we plan to incorporate an "auto-correct" feature that uses a language model that will correct errors after the user has entered one or more words in a message.

We also plan to experiment with the feedback the Perkinput system gives the user. Currently, Perkinput plays a "click" sound for the first input and speaks the character entered after the second input for each character. Many blind expert typists prefer to receive verbal feedback after entering each word, not each character. When the user enters a space, Perkinput can optionally read the previous word. Furthermore, Perkinput can provide more informative non-verbal feedback after each input. Instead of playing the same "click" sound for each tap, Perkinput can play different musical chords for various inputs. We believe this will help the user detect errors quickly and early.

Finally, all participants requested support for Grade 2 Braille in Perkinput. In Grade 2 Braille, several characters that frequently occur together are represented by a single special character. It would be interesting to see how using contractions affects users' entry speeds and accuracy.

In addition to Perkinput, we would also like to deploy PassChords and DigiTaps. We believe that PassChords in particular has broad appeal to both blind and sighted people. It would be interesting to compute the guessing entropy for PassChords from a larger set of users who create real passwords. In DigiTaps, we could experiment with different feedback mechanisms as we would with Perkinput. We could try to incorporate DigiTaps into existing QWERTY keyboards for both blind or sighted users. Currently, users have to switch to a different keyboard mode to access digits and other symbols. Instead, users can enter numbers with DigiTaps gestures elsewhere on the screen while staying in same keyboard mode.

8.3.2. Speech-Based Methods

SpeakNSwipe was the first Eyes-Free Dictation method and it only scratches the surface of the many possibilities for innovations in this space. In addition to the design improvements we outlined in the end of Chapter 7, our work opened many directions for future research.

One promising direction for improving SpeakNSwipe involves integrating it with a speech recognizer. While there are some advantages to treating the speech recognizer as a black box, we may be able to produce better recognition results by giving the recognizer information about the user's interaction with system. As the user moves through phrases in SpeakNSwipe, she makes corrections. We will build a system that will inform the speech recognizer which phrases were corrected (*i.e.*, which phrases were not recognized correctly initially), so that the speech

recognizer can update its probabilistic model of what the user had said. The text in the remainder of the user's text can then be updated according to the revised model before the user proceeds to review it.

Other general directions for eyes-free dictation include improving speech recognition in itself by incorporating user interaction. For example, we found that people find it difficult to speak punctuation marks while dictating text. There has been some work on segmenting speech to infer punctuation marks [87], but not in an interactive, realtime system. We would like to build a system that used techniques developed by speech researchers, but incorporate user interaction to resolve uncertainty.

8.3.3. Input on Mobile Devices for People with Low-Vision

Our work has focused on enabling access to people who have little to no functional vision, so the techniques we studied and invented had no visual output. However, most people with a vision impairments have some functional vision; only fewer than 10 percent of visually impaired people have no functional vision [103]. Moreover, the number of people experiencing some vision loss is projected to grow over the next few years, as the population of older adults increases. Vision loss is common among people who are 65 and older. Thus, one important direction for future work on accessible input on mobile devices involves technology for people with *low-vision*. People who have some functional vision can potentially use their vision to access technology more effectively with techniques that are designed for people with various vision abilities. Despite the compelling need, so far there has been no known work on mobile device access for people with low-vision [10]. We plan to explore this area in the future.

8.4. Conclusion

In this dissertation, we sought to support the following thesis statement:

> *Gesture-based input methods that use simple multi-touch taps, and speech-based input methods that facilitate error detection and correction, can both enable blind people to enter text more effectively on touchscreens than the de facto standard methods.*

We have described three gesture-based methods with simple, multi-touch taps: Perkinput, PassChords, and DigiTaps. We compared Perkinput and PassChords to VoiceOver input in laboratory studies with blind participants. Perkinput and PassChords were both faster than VoiceOver, and Perkinput was more accurate than VoiceOver, the *de facto* standard. On the other hand, there was no appropriate *de facto* standard to compare to DigiTaps, since DigiTaps was the only method that enabled eyes-free input with minimal-audio feedback. However, PIN entry rates in the PassChords study suggest that DigiTaps is faster than the VoiceOver number pad.

We presented the speech-based input method SpeakNSwipe and evaluated it with sighted participants. We believe eyes-free dictation is important for enabling a variety of tasks on many new wearable and mobile devices with little or no screens, so we did not want to evaluate SpeakNSwipe with only blind people. Participants in our SpeakNSwipe study had no visual feedback, however, so there is no reason to believe that performance among blind people would differ. Since SpeakNSwipe is the first eyes-free dictation method, there was no appropriate standard to compare it to in our study.

While our thesis focuses on blind people, we believe that the work presented here introduces ideas that will improve access to technology for a wide range of people, with and without

disabilities. As the landscape of computing devices continues to involve, people use different devices in various situations, affecting their abilities to interact with the device and the world around them. Enabling eyes-free input is an important step in expanding the opportunities of technology to enhance everyone's lives.

Bibliography

1. AirServer. Mirror your display and stream content from your iOS devices. http://www.airserver.com/.

2. Apple. Accessibility: VoiceOver for iOS. https://www.apple.com/accessibility/ios/voiceover/.

3. Apple. Siri. , Siri. http://www.apple.com/ios/siri/.

4. Apple. AirPlay. http://www.apple.com/airplay/.

5. Asokan, N. and Kuo, C. Usable Mobile Security. *Distributed Computing and Internet Technology*, Springer Berlin Heidelberg (2012), 1–6.

6. Asonov, D. and Agrawal, R. Keyboard acoustic emanations. *IEEE Symposium on Security and Privacy*, (2004), 3–11.

7. Azenkot, S., Bennett, C.L., and Ladner, R.E. DigiTaps: eyes-free number entry on touchscreens with minimal audio feedback. *Proceedings of the 26th annual ACM symposium on User interface software and technology (UIST '13)*, ACM Press (2013), 85–90.

8. Azenkot, S. and Lee, N. Exploring the use of speech input by blind people on mobile devices. *Proceedings of the 15th International ACM ...*, (2013).

9. Azenkot, S., Rector, K., Ladner, R., and Wobbrock, J. PassChords: secure multi-touch authentication for blind people. *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '12)*, ACM Press (2012), 159.

10. Azenkot, S., Rector, K., Ladner, R., and Wobbrock, J. The need for research on mobile technologies for people with low-vision. *Third Mobile Accessibility Workshop (MOBACC '13). ACM Conference on Human Factors in Computing Systems (CHI '13)*, (2013).

11. Azenkot, S., Wobbrock, J.O., Prasain, S., and Ladner, R.E. Input finger detection for nonvisual touch screen text entry in Perkinput. *Proceedings of Graphics Interface 2012*, Canadian Information Processing Society (2012), 121–129.

12. Azenkot, S. and Zhai, S. Touch behavior with different postures on soft smartphone keyboards. *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services (MobileHCI '12)*, ACM Press (2012), 251–260.

13. Berger, Y., Wool, A., and Yeredor, A. Dictionary attacks using keyboard acoustic emanations. *Proceedings of the 13th ACM conference on Computer and communications security*, ACM Press (2006), 245–254.

14. Betts, P., Brown, C.J., Lynott, J.J., and Martin, H.F. Light Beam Matrix Input Terminal. *IBM Technical Disclosure Journal 9*, 5 (1965), 493–494.

15. Bigham, J.P. and Cavender, A.C. Evaluating existing audio CAPTCHAs and an interface optimized for non-visual use. *Proceedings of the 27th international conference on Human factors in computing systems (CHI 09)*, ACM Press (2009), 1829–1838.

16. Bigham, J.P., White, S., Yeh, T., et al. VizWiz: nearly real-time answers to visual questions. *Proceedings of the 23nd annual ACM symposium on User interface software and technology (UIST '10)*, ACM Press (2010), 333–342.

17. Bonner, M.N., Brudvik, J.T., Abowd, G.D., and Edwards, W.K. No-look notes: accessible eyes-free multi-touch text entry. *Pervasive Computing*, Springer Berlin Heidelberg (2010), 409–426.

18. Boundless Assistive Technology. Braille Notetakers. http://www.boundlessat.com/Blindness/Notetakers.

19. BrailleTech. BrailleTouch. http://www.cc.gatech.edu/~mromero/brailletouch/.

20. Burr, W.E., Dodson, D.F., Polk, W.T., and Evans, D.L. Electronic authentication guideline. *NIST Special Publication 800-63-2*, (2004).

21. Buxton, B. 31 . 1 : Invited Paper : A Touching Story : A Personal Perspective on the History of Touch Interfaces Past and Future Lost Along the Way. *41*, May (2010), 444–448.

22. Buxton, W., Foulds, R., Rosen, M., Scadden, L., and Shein, F. Human interface design and the handicapped user. *ACM SIGCHI Bulletin 17*, 4 (1986), 291–297.

23. Castellucci, S.J. and MacKenzie, I.S. Gathering text entry metrics on Android devices. *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems (CHI EA '11)*, ACM Press (2011), 1507–1512.

24. Clarke, N.L. and Furnell, S.M. Authentication of users on mobile telephones: A survey of attitudes and practices. *Computers Security 24*, 7 (2005), 519–527.

25. Clawson, J., Lyons, K., Starner, T., and Clarkson, E. The impacts of limited visual feedback on mobile text entry for the twiddler and mini-QWERTY keyboards. *Ninth IEEE International Symposium on Wearable Computers (ISWC'05)*, IEEE (2005), 170–177.

26. Costello, S. 9 Portable Keyboards for the iPhone. *About.com*. http://ipod.about.com/od/introductiontotheiphone/tp/keyboards-for-iphone.htm.

27. Dhamija, R. and Perrig, A. Déjà Vu: a user study using images for authentication. *USENIX Security Symposium 9*, (2000), 4–4.

28. Edgar, R. and Batzoglou, S. Multiple sequence alignment. *Current opinion in structural biology*, (2006).

29. Fleetwood, M.D., Byrne, M.D., Centgraf, P., Dudziak, K., Lin, B., and Mogilev, D. An evaluation of text-entry in Palm OS - Graffiti and the virtual keyboard. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting 46*, 5 (2002), 617–621.

30. Fleksy. Fleksy. http://fleksy.com/.

31. Foo Kune, D. and Kim, Y. Timing attacks on PIN input devices. *Proceedings of the 17th ACM conference on Computer and communications security*, ACM (2010), 678–680.

32. Frey, B., Southern, C., and Romero, M. Brailletouch: mobile texting for the visually impaired. (2011), 19–25.

33. Gardner, F.M. *Phaselock Techniques*. John Wiley & Sons, New York, New York, USA, 2005.

34. Gartner. Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013. http://www.gartner.com/newsroom/id/2623415.

35. Gill, J. Information for Designers of Public Access Terminals. *Royal National Institute for the Blind (on behalf of INCLUDE)*, (1997).

36. Google. Enable TalkBack. https://support.google.com/accessibility/android/answer/6007100?hl=en.

37. Google. Voice Search Anywhere. http://www.google.com/ .

38. Google. Announcing Eyes-Free Shell for Android. 2009. http://google-opensource.blogspot.com/2009/04/announcing-eyes-free-shell-for-android.html.

39. Guida, C. BraillePad. http://cguida.altervista.org/braillepad-en/.

40. Halverson, C., Horn, D., Karat, C., and Karat, J. The beauty of errors: patterns of error correction in desktop speech systems. (1999), 133 – 140.

41. Holman, J., Lazar, J., Feng, J.H., and D'Arcy, J. Developing usable CAPTCHAs for blind users. *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility (Assets '07)*, ACM Press (2007), 245–246.

42. Huffman, D. A method for the construction of minimum redundancy codes. *proc. IRE*, (1952).

43. Jain, M. and Balakrishnan, R. User learning and performance with bezel menus. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems (CHI '12)*, ACM Press (2012), 2221–2230.

44. Jakobsson, M., Shi, E., Golle, P., and Chow, R. Implicit authentication for mobile devices. *Proc. HotSec'09*, USENIX Association (2009), 9.

45. Jansen, W., Scarfone, K., Gutierrez, C.M., Patrick, D., Gallagher, D., and Director, D. Guidelines on cell phone and PDA security. *NIST Special Publication 800*, (2008), 124.

46. Jermyn, I., Mayer, A., Monrose, F., Reiter, M.K., and Rubin, A.D. The design and analysis of graphical passwords. (1999), 1.

47. Kane, S.K., Bigham, J.P., and Wobbrock, J.O. Slide rule: making mobile touch screens accessible to blind people using multi-touch interaction techniques. *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility (Assets '08)*, ACM Press (2008), 73–80.

48. Kane, S.K., Jayant, C., Wobbrock, J.O., and Ladner, R.E. Freedom to roam: a study of mobile device adoption and accessibility for people with visual and motor disabilities. *Proceeding of the eleventh international ACM SIGACCESS conference on Computers and accessibility (ASSETS '09)*, ACM Press (2009), 115–122.

49. Kane, S.K., Wobbrock, J.O., and Ladner, R.E. Usable gestures for blind people. *Proceedings of the 2011 annual conference on Human factors in computing systems (CHI '11)*, ACM Press (2011), 413–422.

50. Karat, C.-M., Halverson, C., Horn, D., and Karat, J. Patterns of entry and correction in large vocabulary continuous speech recognition systems. *Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit (CHI '99)*, ACM Press (1999), 568–575.

51. Kuber, R. and Sharma, S. Toward tactile authentication for blind users. *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility - ASSETS '10*, ACM Press (2010), 289.

52. Kumar, A. and Paek, T. Voice Typing: A New Speech Interaction Model for Dictation on Touchscreen Devices. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*, (2012), 2277–2285.

53. Laird, N.M. and Ware, J.H. Random-effects models for longitudinal data. *Biometrics 38*, 4 (1982), 963–74.

54. Landau, S. and Wells, L. Merging tactile sensory input and audio data by means of the Talking Tactile Tablet. *EuroHaptics '03*, (2003), 414–418.

55. Law, C. and Vanderheiden, G. The development of a simple, low cost set of universal access features for electronic devices. *Proceedings on the 2000 conference on Universal Usability (CUU '00)*, ACM Press (2000), 118–123.

56. Li, K.A., Baudisch, P., and Hinckley, K. Blindsight. *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, ACM Press (2008), 1389.

57. Lin, F.X., Ashbrook, D., and White, S. RhythmLink. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, ACM Press (2011), 263.

58. Littell, R.C., Henry, P.R., and Ammerman, C.B. Statistical analysis of repeated measures data using SAS procedures. *Journal of animal science 76*, 4 (1998), 1216–31.

59. LookTel. LookTel Products: LookTel Money Reader. http://www.looktel.com/moneyreader.

60. MacKenzie, I.S. and Soukoreff, R.W. Phrase sets for evaluating text entry techniques. *CHI '03 extended abstracts on Human factors in computing systems - CHI '03*, ACM Press (2003), 754.

61. MacKenzie, I.S. and Zhang, S.X. The design and evaluation of a high-performance soft keyboard. *Proceedings of the SIGCHI conference on Human factors in computing systems the CHI is the limit - CHI '99*, ACM Press (1999), 25–31.

62. Martin, T.B. and Welch, J.R. Practical speech recognizers and some performance effectiveness parameters. In *Trends in Speech Recognition*. Trends in Speech Recognition, Englewood Cliffs, NJ, 1980.

63. Mascetti, S., Bernareggi, C., and Belotti, M. TypeInBraille. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility - ASSETS '11*, ACM Press (2011), 295.

64. Mishra, T., Ljolje, A., and Gilbert, M. Predicting Human Perceived Accuracy of ASR Systems. *INTERSPEECH*, (2011).

65. Mpaja. mBraille. http://mpaja.com/.

66. Nakatani, L.H. and Rohrlich, J.A. Soft machines. *Proceedings of the SIGCHI conference on Human Factors in Computing Systems - CHI '83*, ACM Press (1983), 19–23.

67. National Federation of the Blind. How many children in America are not taught to read? https://nfb.org/braille-initiative.

68. Needleman, S. and Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, (1970).

69. Nuance. Dragon Speech Recognition Software. http://www.nuance.com/dragon/index.htm.

70. Oh, U., Kane, S.K., and Findlater, L. Follow that sound. *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*, ACM Press (2013), 1–8.

71. Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J., and Gonçalves, D. BrailleType: unleashing braille over touch screen mobile phones. (2011), 100–107.

72. Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J., and Gonçalves, D. Blind people and mobile touch-based text-entry. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '11)*, ACM Press (2011), 179–187.

73. Oorschot, P.C. van and Thorpe, J. On predictive models and user-drawn graphical passwords. *ACM Transactions on Information and System Security 10*, 4 (2008), 1–33.

74. Oviatt, S. Taming recognition errors with a multimodal interface. *Communications of the ACM 43*, 9 (2000), 45–51.

75. Owen, G.S. Graphical Passwords: A Survey. *21st Annual Computer Security Applications Conference (ACSAC'05)*, IEEE, 463–472.

76. Pew Research. Mobile Technology Fact Sheet. 2014. http://www.pewinternet.org/2012/11/30/the-best-and-worst-of-mobile-connectivity/.

77. Pitt, I.J. and Edwards, A.D.N. Improving the usability of speech-based interfaces for blind users. *Proceedings of the second annual ACM conference on Assistive technologies - Assets '96*, ACM Press (1996), 124–130.

78. Poulsen, K. Mitnick to lawmakers: People, phones and weakest links. 2009. http://www.politechbot.com/p-00969.html.

79. Ruamviboonsuk, V., Azenkot, S., and Ladner, R.E. Tapulator: a non-visual calculator using natural prefix-free codes. *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '12)*, ACM Press (2012), 221–222.

80. Salter, K. and Fawcett, R. The ART test of interaction: a robust and powerful rank test of interaction in factorial models. *Communications in Statistics-Simulation ...*, (1993).

81. Sánchez, J. and Tadres, A. Audio and haptic based virtual environments for orientation and mobility in people who are blind. *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility - ASSETS '10*, ACM Press (2010), 237.

82. Schneier, B. The secret question is: why do IT systems use insecure passwords? *The Guardian*, (2009).

83. Sears, A., Karat, C.-M., Oseitutu, K., Karimullah, A., and Feng, J. Productivity, satisfaction, and interaction strategies of individuals with spinal cord injuries and traditional users interacting with speech recognition software. *Universal Access in the Information Society 1*, 1 (2001), 4–15.

84. Sendero Group. Sendero GPS LookAround. http://www.senderogroup.com/products/shopiphone.htm.

85. Shapiro, S. and Wilk, M. An analysis of variance test for normality (complete samples). *Biometrika*, (1965).

86. Shirali-Shahreza, S. and Shirali-Shahreza, M.H. Accessibility of CAPTCHA methods. *Proceedings of the 4th ACM workshop on Security and artificial intelligence - AISec '11*, ACM Press (2011), 109.

87. Shriberg, E., Stolcke, A., Hakkani-Tür, D., and Tür, G. Prosody-based automatic segmentation of speech into sentences and topics. *Speech communication*, (2000).

88. Southern, C., Clawson, J., Frey, B., Abowd, G., and Romero, M. An evaluation of BrailleTouch. *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services - MobileHCI '12*, ACM Press (2012), 317.

89. Stent, A., Syrdal, A., and Mishra, T. On the intelligibility of fast synthesized speech for individuals with early-onset blindness. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility - ASSETS '11*, ACM Press (2011), 211.

90. Suhm, B., Myers, B., and Waibel, A. Multimodal error correction for speech user interfaces. *ACM Transactions on Computer-Human Interaction 8*, 1 (2001), 60–98.

91. Suo, X., Zhu, Y., and Owen, G.G.S. Graphical Passwords: A Survey. *Computer security applications ...*, IEEE (2005), 463–472.

92. The National Federation of the Blind Jernigan Institute. The Braille Literacy Crisis in America: Facing the Truth, Reversing the Trend, Empowering the Blind. https://nfb.org/images/nfb/documents/word/the_braille_literacy_crisis_in_america.doc.

93. The Stanford Natural Language Processing Group. The Stanford Parser: A statistical parser. http://nlp.stanford.edu/software/lex-parser.shtml.

94. Tinwala, H. and MacKenzie, I.S. Eyes-free text entry with error correction on touchscreen mobile devices. *Proceedings of the 6th Nordic Conference on Human-Computer Interaction Extending Boundaries - NordiCHI '10*, ACM Press (2010), 511.

95. Vanderheiden, G.C. Use of audio-haptic interface techniques to allow nonvisual access to touchscreen appliances. *Human Factors and Ergonomics Society Annual Meeting Proceedings 40*, 24 (1996), 1266–1266.

96. Wiedenbeck, S., Waters, J., Birget, J.-C., Brodskiy, A., and Memon, N. PassPoints: Design and longitudinal evaluation of a graphical password system. *International Journal of Human-Computer Studies 63*, 1-2 (2005), 102–127.

97. Williams, J. Guidelines for the use of multimedia in instruction. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting 42*, 20 (1998), 1447–1451.

98. Wobbrock, J.O., Chau, D.H., and Myers, B.A. An alternative to push, press, and tap-tap-tap: gesturing on an isometric joystick for mobile phone text entry. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (SIGCHI '07)*, ACM (2007), 667–676.

99. Wobbrock, J.O., Findlater, L., Gergle, D., and Higgins, J.J. The aligned rank transform for nonparametric factorial analyses using only anova procedures. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (SIGCHI '11)*, ACM (2011), 143–146.

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

101. Wobbrock, J.O. Measures of Text Entry Performance. In I. s. MacKenzie and K. Tanaka-Ishii, eds., *Text entry systems: mobility, accessibility, universality*. Morgan Kaufmann, San Francisco, CA, 2007, 47–74.

102. Wobbrock, J.O. TapSongs. *Proceedings of the 22nd annual ACM symposium on User interface software and technology (UIST '09)*, ACM Press (2009), 93–96.

103. World Health Organization. Visual Impairments and Blindness. 2014. http://www.who.int/mediacentre/factsheets/fs282/en/.

104. Xiao, Q. and Qinghan Xiao. Security issues in biometric authentication. *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, IEEE (2005), 8–13.

105. Yfantidis, G. and Evreinov, G. Adaptive blind interaction technique for touchscreens. *Universal Access in the Information Society 4*, 4 (2005), 328–337.

106. Zhao, S., Dragicevic, P., Chignell, M., Balakrishnan, R., and Baudisch, P. Earpod. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, ACM Press (2007), 1395.

107. Enron Email Dataset. https://www.cs.cmu.edu/~./enron/.

108. The VoiceOver Touch Typing Mode in iOS 6. *AppleVis*, 2012. http://www.applevis.com/podcast/episodes/voiceover-touch-typing-mode-ios-6.