BrailleSketch: A Gesture-based Text Input Method for People with Visual Impairments

Mingzhe Li, Mingming Fan, Khai N. Truong
Department of Computer Science
University of Toronto
franklin.li@mail.utoronto.ca, mfan@cs.toronto.edu, khai@cs.toronto.edu

ABSTRACT
In this paper, we present BrailleSketch, a gesture-based text input method on touchscreen smartphones for people with visual impairments. To input a letter with BrailleSketch, a user simply sketches a gesture that passes through all dots in the corresponding Braille code for that letter. BrailleSketch allows users to place their fingers anywhere on the screen to begin a gesture and draw the Braille code in many ways. To encourage users to type faster, BrailleSketch does not provide immediate letter-level audio feedback but instead provides word-level audio feedback. It uses an auto-correction algorithm to correct typing errors. Our evaluation of the method with ten participants with visual impairments who each completed five typing sessions shows that BrailleSketch supports a text entry speed of 14.53 word per min (wpm) with 10.6% error. Moreover, our data suggests that the speed had not begun to plateau yet by the last typing session and can continue to improve. Our evaluation also demonstrates the positive effect of the reduced audio feedback and the auto-correction algorithm.

CCS Concepts
• Social and professional topics→ Professional topics→ Computer profession → Assistive technologies

Keywords
Blind; Braille; text input; mobile devices; sketch; gesture.

1. INTRODUCTION
Text entry is an important communication mechanism. For people with visual impairments this can be accomplished by writing Braille. Traditionally, this is done manually by using a slate, a stylus, and Braille paper. However, it can be difficult for people with visual impairments to learn to hold the stylus up-right and write Braille backward. Six-key Braille typewriters, such as the Perkins Brailler [21], have since enabled people to type by simultaneously pressing keys that correspond to the different Braille dots accordingly.

Over the past decade, mobile devices have become increasingly more powerful and accessible. With commodity mobile devices, people with visual impairments are able to enter text using the onscreen keyboard with screen reader software, such as Apple's VoiceOver. However, this approach results in a very low text entry speed [4]. Alternatively, people can use speech recognition software to input text at a much faster rate [3]. However, speaking in public places may not always be appropriate and can introduce privacy concerns. A growing body of research has been exploring ways to leverage the user's ability to perform touch and multi-touch gesture inputs on mobile devices. For example, BrailleTap [9], TypeInBraille [15], Perkinput [4], and BrailleEasy [23] enable the user to type Braille by performing multiple taps sequentially to specify the dot codes for the desire letter—this can be time consuming. Methods such as Perkinput [4] and BrailleTouch [22] have also explored how touchscreen mobile devices can be used to support 6-finger chorded typing, as done on the Perkins Brailler.

Researchers have also explored gesture-based approaches [10][17] that allow users to draw gestures that are interpreted into letters. These approaches include methods which require the user to learn a new unistroke alphabet (*e.g.*, MoonTouch [10]) as well as those which allow the user to draw a gesture to represent the intended Braille code (*e.g.*, Edge Braille [17]). Approaches, such as EdgeBraille, allow people to leverage their knowledge of the Braille alphabet. There is only a low learning curve in this situation because people do not need to learn a new alphabet.

Inspired by this line of research, we present BrailleSketch (see Figure 1), a gesture-based text entry method for people with visual impairments to type Braille by drawing a path. To use

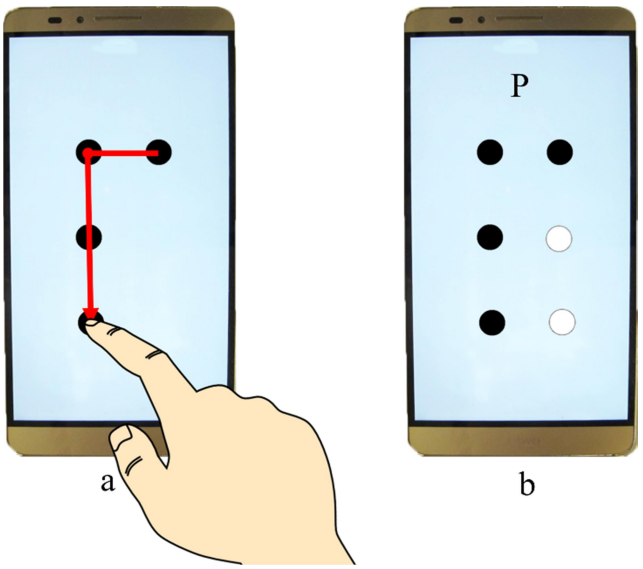

Figure 1. With BrailleSketch, a user simply sketches a path that connects all dots in a Braille code to type the corresponding letter.

ASSETS '17, October 29-November 1, 2017, Baltimore, MD, USA

ACM ISBN 978-1-4503-4926-0/17/10.
https://doi.org/10.1145/3132525.3132528

BrailleSketch, the user simply places her finger anywhere on the touch screen and sketches a path that passes through all the dots in a Braille code in a way that is intuitive to her. Different from existing approaches of typing Braille on the touchscreen, such as BrailleTouch [25], our design leverages the user's knowledge of the Braille alphabet but does not require users to know how to type Braille by performing 6-finger chorded input, as done with a Perkins Brailler. Our method also differs from other gesture-based text entry methods, such as MoonTouch [10], because it does not require people with visual impairments to learn an additional alphabet. We conducted a user study with people with visual impairments to evaluate the performance of BrailleSketch. Our results show that after ~100 minutes of typing, participants on average were able to type 14.53 word per minute (wpm) with 10.6% error rate. The fastest speed achieved overall was 19.32 wpm.

2. RELATED WORK

In this section, we review three general text entry approaches for people with visual impairments. The first approach explores the use of audio feedback to enable the user to input text. The second approach investigates typing-based and tapping-based methods for a variety of mobile devices, such as touchscreens and wearables. The final approach enables the user to perform gestures that are recognized into text.

2.1 Audio-based Methods

People with visual impairments often do not need to use a special keyboard when they use a computer. However, they require special screen reader software that voices the keys that are pressed. On mobile devices, the touchscreen accessibility function, such as Apple's VoiceOver, can help the user identify the keys that she touches in order to enter text. However, previous studies have shown that this method supports very low text entry rate (*e.g.,* 4.5 wpm [4], 4.3 wpm [3], 0.66 wpm [5]). To improve upon this, No-Look-Notes [5] explores dividing the screen into 8 pie segments with each corresponding to a set of characters (*e.g.,* 'ABC') that are easier to select. When the user touches a segment, the system reads the set of characters located in that segment. She can then tap the region with a second finger to select it and all of the characters get arranged vertically on the screen. When the user's finger touches the screen again, the system reads the character located at that position. She then can tap the screen with a second finger to select that letter or slide the finger to find the desired letter. In their evaluation of the system, Bonner *et al.* showed that participants were able to input text using No-Look Notes at a rate of 1.32 wpm with 11% error and VoiceOver at 0.66 wpm with 60% error.

Another audio-based approach to entering text is via speech. Azenkot and Lee's study [3] shows that the average text entry speed via speech is 19.5 wpm. However, speech interaction can potentially be inappropriate in public spaces and may introduce privacy concerns.

2.2 Typing- and Tapping-based Methods

The Perkins Brailler [21] is a typewriting tool that includes six keys which map to the six dots in a Braille code. To type, the user must simultaneously push the keys corresponding to the dots for the intended Braille code.

With the advent of mobile devices, researchers have explored additional ways to allow users to input Braille codes. For example, BrailleTap [9] maps inputs from the phone's physical keypad into dots in a Braille code. Guerreiro *et al.* showed that BrailleTap supports a rate of 3.6 wpm with 6.55% error. BrailleType [19] divides the touchscreen of a phone into a 3×2 grid. The user can sequentially perform a long-press in different grid cells, one cell at a time, to input dots in a Braille code. The user completes the typing of a character by performing a double tap. An evaluation of the system showed that participants were able to input text at a rate of 1.49 wpm with 9.7% error rate. With SingleTapBraille [1][2], users perform a series of sequential taps on the touchscreen to indicate the positions of the dots in a Braille code. Alnfiai and Sampalli evaluated SingleTapBraille and showed that participants achieved 4.71 wpm with 11% total error rate. TypeInBraille [15][16] allows users to input a Braille code by performing three multi-finger tap gestures sequentially. Each tap gesture types a row of dots in the Braille code. A 1-finger tap on the left side of the screen inputs the left dot in a row, while a 1-finger tap on the right side of the screen inputs the right dot in a row, a 2-finger tap input both dots, and a 3-finger tap inputs no dots in a row. This method results in a speed of 6.3 wpm with 3% error.

The Perkins Brailler design and 6-key chording method have also been extended to work on touchscreens and wearables. For example, BrailleEasy [23] and Perkinput [4] allow users to type a column of a Braille code one at a time by simultaneously tapping different fingers of one hand on the touchscreen. With Perkinput, on small touchscreen devices, the two columns of a Braille code can be tapped one after another. The simultaneous chording of 6-keys can also be supported on larger touchscreen devices with both hands touching the screen at the same time. Alternatively, two small devices can be paired to support input from a different hand. In their evaluation of the system, participants were able to achieve 6.05 wpm with 3.52% uncorrected error rate. BrailleTouch [8] allows users to cradle a mobile phone with two hands and type a Braille code with both hands at the same time. Much like how users would type with a Perkins Brailler, three fingers from each hand would be used to type each column of dots in a Braille code. With BrailleTouch, expert users were able to achieve a rate of 23.2 wpm with 14.5% error rate [25] while slower users reached 9.40 wpm with 39.3% error rate. Apple iOS natively includes Braille keyboard that uses a similar method to BrailleTouch. However, one drawback of these methods is that the number of people who know how to type Braille using Perkins-like methods is less than those who can read Braille [6].

Additionally, researchers have also examined glove based wearable text input methods. Lee *et al.* [18] designed a pair of chording gloves that allowed people with visual impairments to type Braille on any surface as if they were typing on a Braille typewriter. An evaluation of the work showed that participants were able to reach a rate of 24.3 wpm and 5.2% error rate [11]. Despite the high text entry speed, glove based methods require users to wear additional hardware on their hands.

2.3 Gesture-based Methods

Researchers have also explored the use of gestures as an intuitive method for people with visual impairments to input text. For example, EdgeBraille [17] users can enter a Braille code by swiping their finger along a mobile device's edges, which contain areas that could be touched to input dots. Participants achieved a speed of 7.17 wpm with 8.43% error with EdgeBraille. Because the dots are input by touching areas along the edge of the screen, users have to move their finger across the entire screen to draw Braille codes, which results in long travel distances. Inspired by EdgeBraille's design, our design removes these two constraints by allowing users to place their finger anywhere on the screen and to

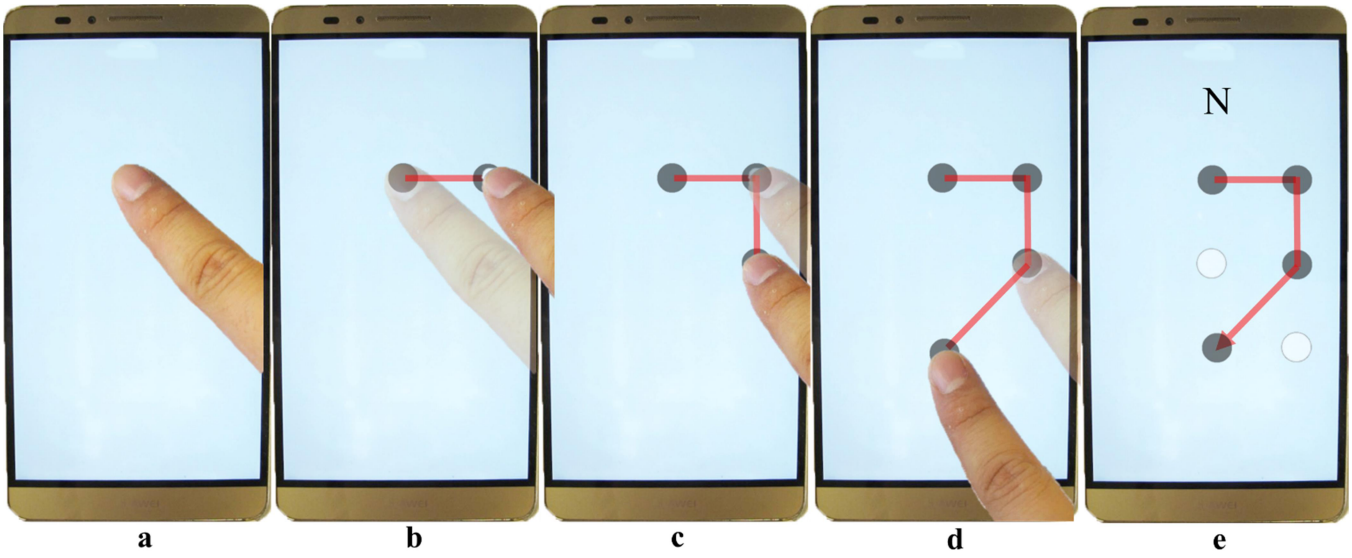

Figure 2. BrailleSketch on a smartphone. A user types the Braille code for 'N' by touching a finger anywhere on the screen first (a), then moving it horizontally to the right (b), then vertically to the bottom (c), then diagonally to the bottom left (d), and finally lifting the finger off the screen (e). The sketched path passes through the 4 dots in the Braille code of 'N' in a single gesture.

sketch the gesture without needing to reach specific positions. Our design also allows users to draw the same Braille code differently.

Another type of gesture-based text entry methods for people with visual impairments is to draw pre-defined gestures for each English letter. For example, Moontouch [10] allows users to draw gestures in Moon Alphabet to input English letters. Heni *et al.*'s evaluation of their approach shows that participants were able to type at 10.14 wpm. In contrast, our work examines the speed at which participants would be able to input text using gestures based on their knowledge of the Braille alphabets instead of learning a different alphabet, such as Moon.

3. SYSTEM DESCRIPTION

In this section, we describe how we implemented BrailleSketch as a gesture based text input method that allows people with visual impairments to sketch Braille codes on a touchscreen.

3.1 High-Level Design

We will first describe how users input a Braille code by performing a gesture. We will then discuss specific audio feedback provided to help the user to sketch the code without sight.

3.1.1 Sketching a Braille Code

To input a letter, the user simply sketches a path that connects all the dots in the desired Braille code. We illustrate how the user would type 'N' using BrailleSketch in Figure 2. The Braille for the letter 'N' contains 4 dots: 2 dots in the top row, one in the right column of the middle row, and one in the left column of the last row. In this example, the user simply places her finger anywhere on the screen to input the top left dot. Then, the user drags her finger to the right to input the top right dot. Next, the user drags her finger downwards to input the dot on the right side of the middle row. Finally, the user drags her finger downwards diagonally to input the fourth dot on the left side of the bottom row. Once the user lifts her finger from the screen, the letter 'N' is recognized.

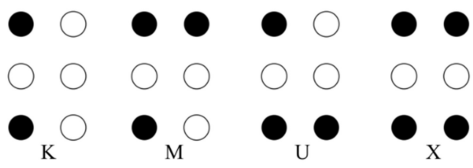

Figure 3. Braille codes for "K", "M", "U", and "X".

Most Braille codes can be drawn in more than one way. To make the system easy to use, BrailleSketch allows users to sketch the Braille code however it seems most natural to them. In Section 3.2.1, we discuss how the system recognizes the dots intended in the path drawn by the user and why this enables the gesture to be drawn flexibly.

Braille codes for the letters 'L', 'M', 'U', and 'X' (see Figure 3), do not have dots in the middle row. However, we designed our method to always input the next adjacent dot after the gesture is continued along a direction for a particular distance. Thus, to enable the user to input Braille codes that do not include dots in the middle row, the user must draw two gestures that are recognized together. A gesture path drawn very shortly (<500 ms) after another gesture path is treated as part of the previous gesture. For example, a 'K' is drawn using a double tap; 'X' is drawn using two lines; 'M' is drawn with a horizontal line first and then a dot; 'U' instead is drawn with a dot first and then a line.

3.1.2 Audio Feedback

To help people with visual impairments better perceive whether they have sketched is correct, the system provides different audio cues. Specifically, a 1000 HZ sine-wave audio sound will be played when a new dot is in a cardinal (*i.e.,* horizontal and vertical) direction adjacent from the previously added dot, and a 200 HZ sine-wave audio sound will be played if it is in an ordinal (*i.e.,* diagonal) direction from the previous dot. For example, when typing the Braille code for the letter 'N', the user will hear the same audio cue in Figure 2*b* and Figure 2*c,* because the new dot is cardinally adjacent to the previous dot. However, she will hear a different audio cue in Figure 2*e* because the new dot is ordinally adjacent to the previous dot. In this way, the user knows when she

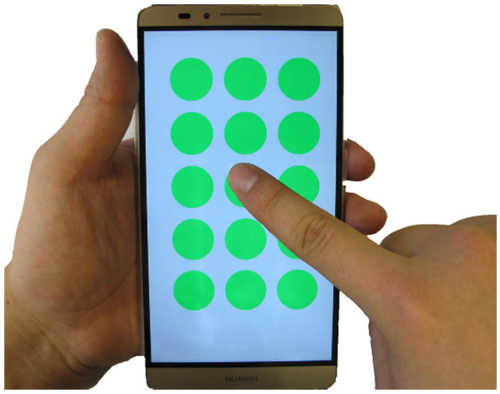

Figure 4. A 5x3 grid of dots used to recognize the gesture drawn by the user is computed and centered where the finger touches the screen.

has incorrectly inputted a letter. For example, to draw 'C', the user simply needs to drag her finger horizontally, on the screen; and to draw an 'E', the user would drag her finger downwards diagonally left to right. If the user wants to type 'C', but hears a 200 HZ sine-wave audio sound, then she would know that the gesture was drawn incorrectly.

Previous research suggests that reading each letter during typing can decrease the text entry speed. Specifically, Mackenzie and Castellucci [14] showed that providing feedback after typing a word, rather than after each letter, can increase the typing speed. Inspired by this idea, our system does not provide letter level feedback. Instead, it only reads aloud the entire word after the user types a space to complete a word. The system, however, does provide an immediate audio feedback ("*no*"), if the sketched gesture cannot be recognized as any letter. Additionally, our design does not support correction while drawing the gesture. To correct an error, the user must delete the wrong gesture and type it again.

3.2 Implementation

In this section, we discuss how BrailleSketch recognizes a gesture as a Braille code and how it maps Braille dots. Then we describe how BrailleSketch implements the auto-error correction and supports the input of additional keys (*i.e.,* space, delete, and enter).

3.2.1 Sketch Recognition

We implemented BrailleSketch as an Android application. We used HUAWEI Ascend Mate7 phone running Android version 4.4.2 as the testing device. It has a 6.0-inch screen. The height and width are 157 mm and 81 mm respectively, and the resolution is 1080 × 1920 pixels. BrailleSketch allows the user to touch anywhere on the screen to start a gesture. When the user touches the screen, BrailleSketch computes a 5 rows × 3 columns grid of dots centered at the touch point (see Figure 4). We set the size of the grid to be 5×3 so that the touch point can be any dot of a 3×2 Braille code. For example, in Figure 4, depending on the direction that the user draws the gesture next, the touch point can be the bottom right dot of the 3×2 Braille code or it can also be the top left dot of the Braille code, *etc*. The diameter of each dot in the grid is 2/15 of the screen's height (*i.e.,* the longer edge), which is 2.1 cm on the testing phone. This is slightly larger than the average width of the pad of the index finger for adults (1.6 to 2 cm) [7]. The distance between the centers of two adjacent dots is 1/6 of the screen's height.

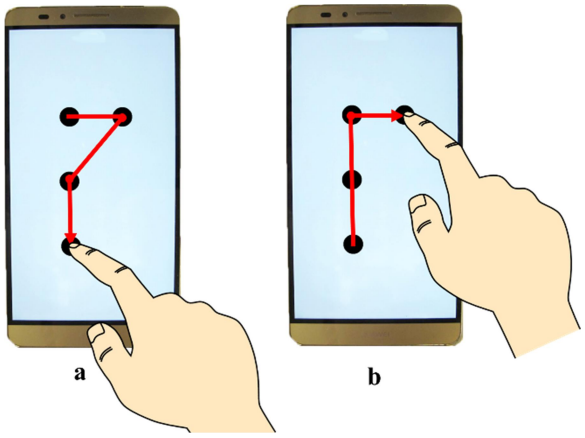

Figure 5. Two different ways of sketching the Braille code for the letter "P" beyond the one shown in the Figure 1.

BrailleSketch supports different ways of typing a letter. For example, Figure 1 shows one way to type the letter 'P' by starting from the top right corner dot, moving up to the top left corner dot, and finally ending at the bottom left corner dot. Alternatively, the user can also start from the top left corner dot, move to the top right corner dot, then move diagonally to the middle left dot, and finally end at the bottom left corner (see Figure 5 a). The user can also start from the bottom left dot, move up to the top left dot, then move horizontally to the right and end at the top right dot (see Figure 5 b). Thus, BrailleSketch allows the user to sketch Braille codes flexibly. The Grade 1 Braille alphabet design enables our method to use the subsequent dots in a gesture to determine the position of the first touchpoint. For example, if the subsequent dots in the pattern are above the dot added at the initial touch point, then the system determines the first touchpoint must be in one of the bottom two rows.

3.2.2 Auto-Correction

BrailleSketch implements an auto-correction algorithm [14] to correct errors in typed words. The algorithm works as follows: if the word typed by the user is in the dictionary, then it is left alone. The algorithm uses a dictionary that contains 10,000 words taken from the British National Corpus [24]. If a word is not in the dictionary, it computes the minimum string distance (MSD) between the inputted word and all words in the dictionary. The system then generates three lists of words with MSD equals 1, 2 and 3. It then sorts each list of words based their frequency. It combines the three lists and moves all the words that have the same length as the inputted word to the front of the combined list. The word at the top of the list is used to replace the inputted word. It is also possible that the algorithm does not find any match to the word. In that case, the auto-correction leaves the typed word as is. Future implementations of the autocorrection algorithm will explore how to include a dictionary with all words from the British National Corpus.

3.2.3 Additional Keys

Aside from the 26 English letters, BrailleSketch also allows the user to type three additional keys: *Space, Delete* and *Enter* (see Figure 6). When the keyboard is active, we override the functionality of the *volume* buttons. The user can type *Space* to finish a word by pressing the *volume down* button (located on the right side of the phone). She can type *Enter* by pressing anywhere on the screen for more than 2000 ms. Finally, she can delete a letter by performing swipe right gesture that leaves the screen.

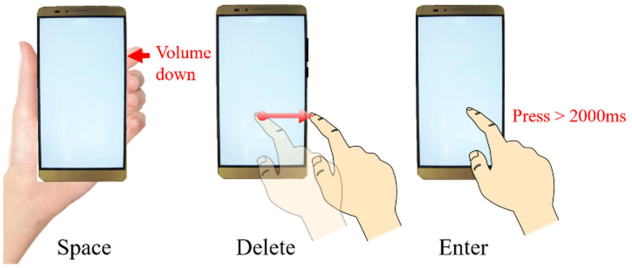

Figure 6. Three additional keys implemented: pressing volume down key to type the Space; swiping to the right to delete a letter; pressing anywhere on the screen for more than 2s to type "Enter".

4. EVALUATION

4.1 Method

We next investigated the typing performance that participants with visual impairments were able to achieve with BrailleSketch. We describe the participants and the procedure involved in our evaluation of the BrailleSketch method.

4.1.1 Participants

We recruited 10 participants with visual impairments (2 males and 8 females) from CNIB (Canadian National Institute for the Blind) via emails and snowball sampling. The average age of the participants was 50.3 (SD=11.4). The average year of experiencing Braille was 32.3 (SD = 8.1). All participants were legally blind. Four of the participants were blind since born. All participants have used Braille-based typing devices before the study.

4.1.2 Procedure

Our study consisted 5 test sessions (1 to 5). Before the first test session, we provided participants with a training session (~10 minutes) to learn and practice typing using BrailleSketch. Then during each test session, BrailleSketch presents the participant with one text phrase at a time by reading it aloud using the Android's text to speech synthesizer. We asked participants to type this phrase as fast and accurately as possible using only lower-case letters and spaces, with no punctuation or numbers. Participants can press the volume up button located on the right side of the phone to request the system to speak the phrase out again. We created five sets of phrases by shuffling the standard set of 500 English phrases which was developed by Mackenzie and Soukoreff [13]. In this standard set, the frequency of each letter is highly correlated with the English language. Each test session used a different shuffling of the phrase set, and all participants were presented with the same shuffling of the phrases in the same order.

Each test session consists of two parts. First, participants were asked to type as many phrases as possible for 15 minutes. After each 15-minute typing session, we also asked the participants to type 3 additional phrases in order to be able to directly compare our results with a Heni *et al.*'s gesture-based text input method for people with visual impairments[10].

We arranged a 5-minute break between each two test sessions to let the participants relax. After the participants completed all test sessions, we interviewed them about problems that they might have encountered, and to collect their feedback about BrailleSketch.

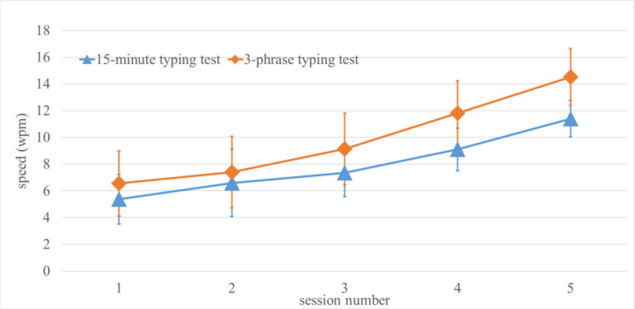

Figure 7. Typing speed for each session of the two typing tests. The horizontal axis is session number and the vertical axis is speed (wpm). Bars show standard deviations.

We compensated each participant $65 for completing the study. We also offered a $50 gift card to the participant who had the highest average speed across all five sessions to motivate them to keep typing as quickly and accurately as they could throughout the entire study.

4.2 Results

The ten participants completed a total of 50 15-minute typing test sessions and 50 3-phrase typing test sessions. We analyzed the speed, error rate, gesture per character of the data collected using the StreamAnalyzer [26]. We treated the test session as one independent variable (IV) with 5 levels (*i.e.*, five test sessions) and the type of typing test as the second IV with 2 levels (*i.e.,* the15-minute typing test and the 3-phrase typing test). We performed a 2-way repeated measure ANOVA to examine whether the changes between sessions and the difference between two typing tests were statistically significant or not. When a statistically significance was found, we performed pairwise comparisons with Bonferroni correction to identify the pair(s) that exhibited significant difference. We reported partial eta square as the measure of the effect size in addition to the p-value.

4.2.1 Speed

Figure 7 shows the average typing speeds in wpm for each session of the two typing tests. For the 15-minute typing tests, the speed in the first test session was 5.37 wpm (SD = 1.86). It grew session after session and reached 11.39 wpm (SD = 1.36) in the last session. The best performance in the final session was 13.30 wpm, while the worst performance was 9.748 wpm.

For the 3-phrase typing tests, the speed in the first test session was 6.56 wpm (SD = 2.43). It also grew session after session and reached 14.53 wpm (SD = 2.13) in the last session. The best performance in the final session was 19.32 wpm, while the worst performance was 11.53 wpm.

There was a significant effect of the type of typing test on the text entry speed ($F_{1,9}$ = 31.02, p < .001, η_p^2 = .78). There was a significant effect of the test session on the text entry speed too ($F_{4,36}$ = 121.5, p < .001, η_p^2 = .93). Pairwise comparisons show that the differences between each two sessions were statistically significant (p < .05).

4.2.2 Error Rates

We calculated the average uncorrected, corrected, and total error rates for two typing tests. We first computed the average error rates per session per person. We then averaged all participants' error rates for each session to compute the average error rate for

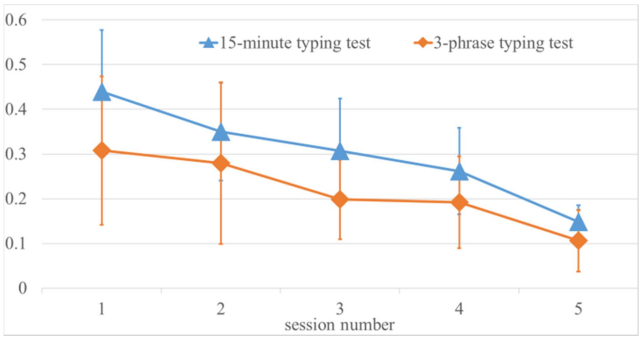

Figure 8. Total error rate for each session of the two typing tests.

each test session. The total, uncorrected, and corrected error rates are shown in Figure 8, Figure 9 and Figure 10 respectively.

4.2.2.1 Total Error Rates

Figure 8 shows the total error rates for the two typing tests over the five test sessions. For the 15-minute typing tests, the total error rate decreases from 0.439 (SD = 0.139) in the first session to 0.308 (SD = 0.166) in the last session. For the 3-phrase typing tests, the total error rate decreases from 0.148 (SD = 0.036) in the first session to 0.106 (SD = 0.069) in the last session.

There was a significant main effect of the typing test on the total error rate ($F_{1,9}$= 13.78, p < .01, η_p^2= .61). There was a significant effect of the test session on the total error rate too ($F_{4,36}$= 22.64, p < .01, η_p^2= .72). Pairwise comparisons show that the differences between the two sessions were significant (p < .05) except between session 1 and session 2 (p = 0.96) and between session 2 and 3 (p = 0.66).

4.2.2.2 Uncorrected Error Rates

Figure 9 shows the uncorrected error rates for the two typing tests over the five test sessions. For the 15-minute typing tests, the uncorrected error rate decreases from 0.105 (SD = 0.0908) in the first session to 0.047 (SD = .036) in the last session. For the 3-phrase typing tests, the uncorrected error rate decreases from 0.0804 (SD = 0.100) in the first session to 0.03 (SD = 0.040) in the last session.

There was a significant main effect of the typing test on the uncorrected error rate ($F_{1,9}$= 5.36, p < .05, η_p^2= .37). There was a statistically significant effect of the test session on the total error rate too ($F_{4,36}$= 3.12, p < .05, η_p^2= .26). Pairwise comparisons show no significant differences between any two sessions.

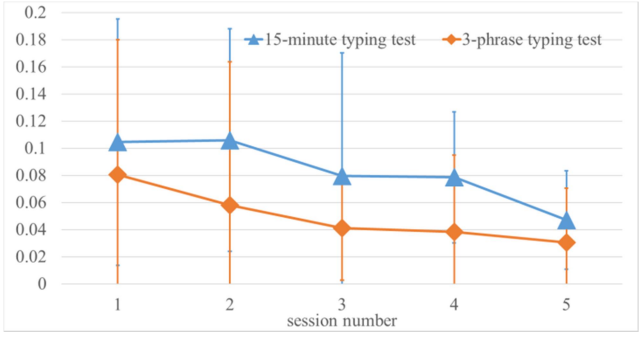

Figure 9. Uncorrected error rate for each session of the two typing tests.

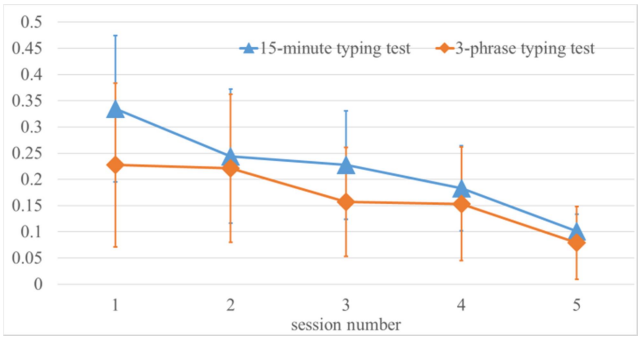

Figure 10. Corrected error rate for each session of the two typing tests.

4.2.2.3 Corrected Error Rates

Figure 10 shows the corrected error rates for the two typing tests over the five test sessions. For the 15-minute typing tests, the corrected error rate decreases from 0.33 (SD = 0.14) in the first session to 0.10 (SD = 0.03) in the last session. For the 3-phrase typing tests, the uncorrected error rate decreases from 0.23 (SD = 0.16) in the first session to 0.079 (SD = 0.07) in the last session.

There was a significant effect of the typing test on the corrected error rate ($F_{1,9}$= 13.85, p < .01, η_p^2= .61). There was a significant effect of the test session on the corrected error rate too ($F_{4,36}$= 16.09, p < .01, η_p^2= .64). Pairwise comparisons show that the differences between the last session and the first three sessions were significant (p <.05). There was a trend towards significance between the last session and the fourth session (p=.053) and between the fourth session and the first session (p=.070). There was no significant difference between any other sessions.

4.2.3 Gesture Per Character (GPC)

GPC denotes the number gestures taken on average to input a letter. GPC provides an indication of the accuracy and efficiency of an input method. A high GPC value indicates low accuracy and low efficiency. The closer the GPC value is to 1, the more accurate and efficient the method is.

Figure 11 shows the GPC for two typing tests over 5 test sessions. For the 15-minute typing tests, GPC decreases from 2.16 (SD = 0.506) in the first session to 1.24 (SD = 0.096) in the last session. The best GPC was 1.05 (SD = 0.055) in the last session. For the 3-phrase typing tests, the uncorrected error rate decreases from 1.81 (SD = 0.79) in the first session to 1.193 (SD = 0.145) in the last session. The best GPC was 1 in the last session.

There was a trend towards significance between the two typing

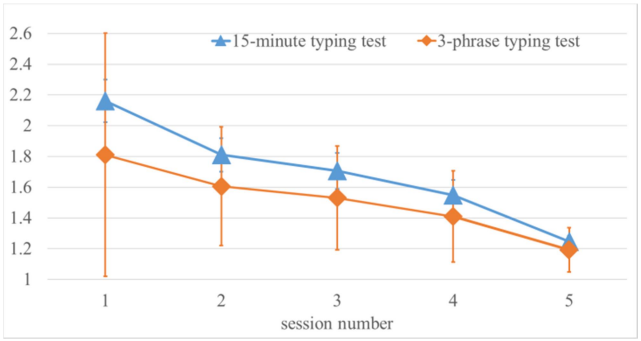

Figure 11. GPC for each session of the two typing tests.

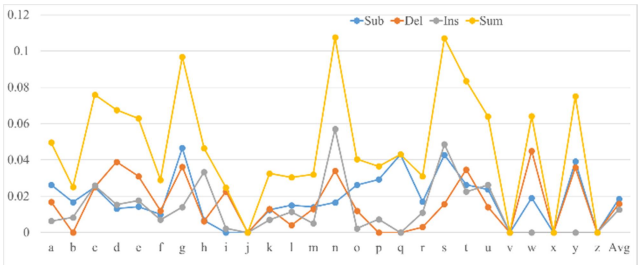

Figure 12. Three types of uncorrected error per letter in the last session of the 15-minute typing tests. "Sum" represents the total uncorrected error rate for each letter. "Avg" represents the average uncorrected error rates of all letters.

tests ($F_{1,9}$= 4.98, p = .053, η_p^2= .36). There was a significant effect of the test session on the GPC too ($F_{4,36}$= 15.61, p < .001, η_p^2= .64). Pairwise comparison results show that the differences between the last session and the first three sessions were significant (p <.05). There was a trend towards significance between the last session and the fourth session (p=.052). There was no significant difference between any other sessions.

4.2.4 Unpacking the Errors

We analyzed the data from the last session of the 15-minute typing test to gain a better understanding of the errors that participants made when using BrailleSketch. We used the data from the 15-minute typing tests, because participants typed more phrases in the 15-minute typing tests than the 3-phrase typing tests and also made significantly more errors. Additionally, statistical analysis shows a significant difference or a trend towards significant difference between the last session and any of the first four sessions.

4.2.4.1 Detailed Uncorrected Errors

Figure 12 shows the different uncorrected errors per letter in the last session of the 15-minute typing tests. From the figure, we can see that *Substitution* error was the most common type of error. It was followed by *Deletion* error and then *Insertion* error. N, S and G were the top-3 letters with the highest total uncorrected errors. G, S, and Y were the top-3 letters with the highest *Substitution* errors. On the other hand, I, V, and X were the top-3 letters with the lowest *Substitution* errors.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	193	0.6	0.11	0.67	0.56	0.54	0	0.06	0	0	0	0.01	0	0	1.9	0	0	0.06	0	0.41	0.22	0	0.01	0	0.11	0
b	1.2	134	0	0	0	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0.27	0	164	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0.23	0	0	0	0	0.07	0
d	0.4	0	0	103	0.25	0	0	0	0	0	0	0	0	0.14	0	0	0	0.33	0	0.5	0	0	0	0	1	0
e	0.68	0	0	0	155	0	0	0.03	0	0	0.5	0	0	0	0	0	0.4	0	0	0	0.08	0	0.1	0	0.33	0
f	0.4	0	0	0.48	0.4	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0.08	0	0	0	0	0	0
g	0	0	0.5	0	0	0	132	0	0	0	0	0	0	6.5	0	0	0	0	0.24	0.31	0.1	0	0	0	0	0
h	0	0	0	0	0	0	0	121	0	0	0	0	0.17	0	0	0	0.2	0	0.33	0	0	0	0	0	0	0
i	0.3	0.2	0	0	0	0	0	0	36	0	0	0	0.12	0	0.53	0	0	0	0	0	0.06	0	0	0	0	0
j	0.5	0	0	0	0	0	0	0	0	14	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
l	0.03	0.3	0	0	0	0.1	0.5	0.5	0	0	0	112	0	0	0.29	0.01	0	0.33	0	0.54	0.01	0	0.5	0	0	0
m	0	0	0.1	1	0	0	0	0	0	0	0.6	0.1	38	0	0	0	0.4	0	0	0	0	0	0	0	0	0
n	0	0	0	0.25	0	0	0	0	0	0	0	0	0	234	0	0	0	0.33	0.5	0	0	0	0	0	0	0
o	0	0	0.61	0.19	0.55	0	0	0	0	0	0	0.1	0.17	0	64	0	0	0.14	0.75	0.29	1	0	0	0	0.1	0
p	0	0	0	0	0	0.2	0.5	0	0	0	0	0	0	0	0	182	0	0	0	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.5	52	0	0	0	0	0	0	0	0	0
r	0.67	0	0	0.13	0.04	0	0	0	0	0	0	0.33	0	0	1	0	0	64.4	0	0.21	0	0	0	0	0.67	0
s	0.2	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	4.5	0	0	93	0	0	0	0	0	0.25	0
t	0	0	0	0	0	0.5	0	0	0	0	0	0.33	0	0	0	0	0	0	2.5	126	0	0	0	0	0	0
u	0	0	0	0	1.25	0	0	0	0	0	0	0	0	0	0	0	0.4	0	0	0	47	0	0.5	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0	0	0	0	10	0	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	74	0	0	0
x	0	0	0.06	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0	0	17	0	0
y	0	0	0	0	0	0	3.4	0.3	0	0	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	63	0
z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12

Figure 13. The confusion matrix of the substitution errors of the last session of the 15-minute typing test.

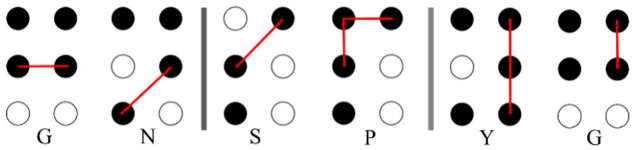

Figure 14. The three letters with the highest substitution errors and what they were mistyped as. Red lines represent the part of the sketched gesture path that contributed to the confusion between these pairs of letters.

We further analyzed the most common type of error, the *Substitution* error, by computing a confusion matrix [26]. Figure 13 shows the confusion matrix of the *Substitution* errors for all letters in the last session of the 15-minute typing tests. Each cell in row x and column y of the matrix contains the number of occurrences of a substitution error where the presented letter x was transcribed as y. The counts were weighted by all possible alignments and thus may not be integers [12]. We found that: 1) all letters, as expected, appeared unevenly in the testing phrases; 2) most letters fell in the diagonal line, which were correctly transcribed; 3) G, S, and Y had the highest *Substitution* errors and were mistyped as N, P and G respectively. Figure 14 shows the Braille codes of these letters. The difference between G and N and the difference between S and P were the direction of the initial (or final) part of the gesture path (see the first two groups in Figure 14). The difference between Y and G was the length of a vertical line in the gesture paths (see the last group in Figure 14).

4.2.4.2 Detailed Corrected Errors

Corrected errors happen when participants delete incorrectly typed letters and change them. We computed the corrected error rate of each letter by dividing the number of corrections for a letter by the total number of times that the letter appeared in the testing phrases. Figure 15 shows the corrected error rate of each letter in the last session of the 15-minute typing test. We found that the letters with the highest corrected error rate were: N, V, and Z. The letters with the lowest three corrected error rates were: J, Q, and X.

4.2.5 Subjective Feedback

Participants were generally positive about using BrailleSketch. For example, one participant commented that "*Typing with one finger is nice. I found that it is easy to use, even though I am used to typing with the Perkins Brailler. I am getting faster after getting used to drawing on the phone. It is also helpful for people to learn Braille*." Participants also felt excited about the potential uses of BrailleSketch. For example, one participant suggested the need to integrate a method such as BrailleSketch with home appliances, such as the microwave, because more and more appliances are equipped with a touch screen that are difficult to use without sight. More generally, participants encouraged us to release BrailleSketch for the iOS system so that more people with visual impairments could download and use it.

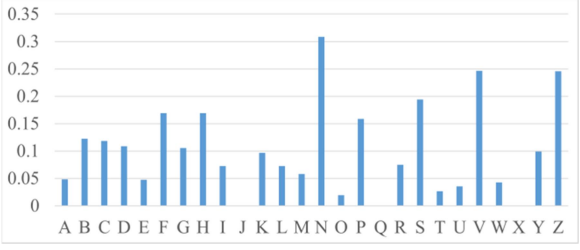

Figure 15. The corrected error rate per letter for the last session of the 3-phrase typing tests.

5. DISCUSSION

In this section, we first discuss our system's performance in comparison to other gesture based text entry methods. We then discuss the speed and error rate achieved with BrailleSketch and compare them with other text entry methods for people with visual impairments. Afterwards, we analyze the performance of the automatic error correction algorithm that we used in BrailleSketch. Finally, we discuss the effect of the delayed audio feedback that BrailleSketch used and the intuitiveness of the method.

5.1 Gesture based Text Input Methods

Like BrailleSketch, EdgeBraille [17] and MoonTouch [10] are two input methods that allow people with visual impairments to draw gestures that are interpreted into text. EdgeBraille, similar to BrailleSketch, leverages the user's familiarity with the Braille alphabet. However, EdgeBraille requires that users draw gestures that touch areas along the edges of the touchscreen to specify the dots in the intended Braille code; this requires long travel distances on the touchscreen. On the other hand, MoonTouch requires that users learn the Moon alphabet. However, MoonTouch allowed the users to draw gestures of any size anywhere on the screen. This enabled MoonTouch users to input text faster than EdgeBraille users (at 10.14 wpm [10] *vs*. 7.17 [17] respectively).

In this work, we examine how to develop a gesture-based method that leverages the users' familiarity with the Braille alphabet without requiring the user to target specific locations on the screen to support accurate input. We designed the evaluation procedure to include the 3-phrase typing test following the 15-minute typing tests in order to formally compare BrailleSketch with MoonTouch [10]. In particular, Hani *et al.*'s evaluation of the MoonTouch system included 5 evaluation sessions in which participants were asked to type three phrases as quickly and accurately as possible after "learning sessions" for "learning the system and interacting with it during a time interval of at least 15 min and at most 20 min" [9]. Participants typed at an average speed of 10.14 wpm in the fifth session with 1.22 GPC when using MoonTouch. In contrast, participants typed at an average speed of 14.53 wpm with 1.19 GPC when using BrailleSketch.

5.2 Other Braille-based Text Input Methods

Table 1 shows the average speed and error rate of other Braille-based text input methods from the literature and our BrailleSketch. We also added training time and study time whenever we could find them from the related papers. The speed achieved with BrailleSketch (14.53 wpm) outperformed the other methods listed except BrailleTouch's rate achieved by expert and moderate performance groups. The typing speed with BrailleTouch in the last three sessions were not significantly different from each other [25], which suggested that the reported speed probably has begun to plateau. However, the typing speed between any two sessions with BrailleSketch system was significantly different ($p < .05$). This shows that typing speed achieved with BrailleSketch has not begun to plateau by the last session yet and still has the potential to increase if participants keep using it. On the other hand, the error rate made by participants using BrailleSketch (0.11) had already dropped lower than that of BrailleTouch.

Table 1. Speed and error rate of Braille-based input method.

Braille-based text entry method	Speed (wpm)	Error rate	Training Time (min)	Study Time (min)
BrailleType [20]	1.49	0.07	10 ~ 15	N/A
SingleTapBraille [1]	4.71	0.11	20	40 ~ 60
Perkinput [4]	6.05	0.04	75	525
TypeInBraille [15]	6.3	0.03	10	N/A
EdgeBraille [17]	7.17	0.08	5	N/A
BrailleEasy [23]	9.82	0.10	40	40
BrailleTouch [25] (poor performance group)	9.4	0.39	N/A	450
BrailleTouch [25] (moderate performance group)	21	0.33	N/A	450
BrailleTouch [25] (expert performance group)	23.2	0.15	N/A	450
BrailleSketch (our system)	14.53	0.11	10	~100

5.3 Audio Feedback

Inspired by previous research that showed removing immediate per letter feedback could increase text entry speed [14], BrailleSketch does not speak out the typed letter immediately after users type it. Instead, it only speaks out the entire word after users finish typing it.

To examine the effect of the reduced audio feedback strategy, we also implemented BrailleSketch with the immediate letter-level audio feedback that reads aloud the letter immediately after it was typed. We tested it with one participant using the same study protocol. The participant was born blind and has used Braille extensively for over 30 years. The typing speed of his last session using the system with immediate letter-level audio feedback was 8.37 wpm, which was lower than the lowest speed of any participants in the last session when using BrailleSketch without immediate letter-level audio feedback (11.53 wpm). The total error rate of his last session in this test (0.113) was roughly equal to the average total error rate of BrailleSketch without immediate letter-level audio feedback (0.106). These results suggest that the reduced audio feedback in the current BrailleSketch implementation (*i.e.,* no immediate letter-level audio feedback) improved its text entry speed without increasing the error rate. However, we only tested the audio feedback with one participant. Further evaluation with more participants are needed to further validate the effect of different audio feedback strategies.

In our current design, BrailleSketch provides an immediate audio feedback ("*no*") to the users, if the sketched gesture cannot be recognized as any letter (erroneous input). Our evaluation shows that participants did not confuse the word "no" with erroneous input because words are only read aloud after being completed with *Space* or *Return*. However, a better audio feedback, such as non-word sounds, should be considered in the future.

While designing the audio feedback to indicate the direction the users are sketching their gesture paths, we tested different sound frequencies internally before choosing 1000 HZ and 200 HZ. However, we did not systematically evaluate how different audio feedback may affect text entry performance, which is an important future topic of discussion regarding input methods for people with visual impairments.

5.4 Automatic Error Correction

To evaluate the effect of the auto-correction algorithm that our system used, our system also recorded the transcribed texts before using the auto-correction algorithm. We compared the transcribed phrases before and after the auto-correction was applied with the presented phrases in the 3-phrase typing tests. We found among the total 770 presented words, 62 words were mistyped before using auto-correction. The auto-correction properly corrected 34 words, which was 55% of the mistyped words. Of the 28 words that auto-correction failed to correct, 15 were changed to incorrect words and 13 were left as-is. Future works will examine how to improve the auto-correction algorithm.

5.5 Intuitiveness of BrailleSketch

BrailleSketch can be used quickly by people who already know the Braille alphabet because it allows users to create their own mappings of the Braille pattern for letters into gestures without any training needed. The system can recognize different gestures for the same letter based on the sketched paths after completed (Figure 5). Whichever is the most natural mapping for the user, they can draw it. For many other Braille typing methods, users need to know how to type Braille in Perkin-like methods.

To correctly type letters, participants either know the gestures that they can use or are able to guess it. Thus, to some extent, our study tested the intuitiveness of BrailleSketch. We also analyzed the errors caused by touching too high/low on the screen through the five test sessions. The average total error rate of all participants in all test sessions due to touching too high/low on the screen was 0.85%. This low error rate suggests that participants were able to land or adjust their finger to avoid the edges of the phone during the typing test.

6. CONCLUSION

In this paper, we presented BrailleSketch, a Braille-based text entry method that allows users to input text by sketching a path that connects all the dots in the intended Braille. The average speed achieved using BrailleSketch by participants in our study was 14.53 wpm with 10.6% error and 1.19 GPC. We showed that the speed has not begun to plateau by the last session yet and still has the potential to increase. Our results show that gesture-based Braille text entry method with reduced audio feedback and auto-correction is potentially better than most tapping and typing methods for people with visual impairments who already know how to read Braille. Our approach does not require people with visual impairments to learn an additional alphabet and is faster than other alphabet based gesture methods.

7. ACKNOWLEDGES

This research was approved by the Social Science, Humanities, and Education Research Ethics Board (REB) at the University of Toronto under the protocol reference #32798. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

8. REFERENCES

[1] Alnfiai, M. and Sampalli, S. 2016. An Evaluation of SingleTapBraille Keyboard. *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility - ASSETS '16* (New York, New York, USA, 2016), 161–169.

[2] Alnfiai, M. and Sampalli, S. 2016. SingleTapBraille: Developing a Text Entry Method Based on Braille Patterns Using a Single Tap. *Procedia Computer Science*. 94, (2016), 248–255.

[3] Azenkot, S. and Lee, N.B. 2013. Exploring the use of speech input by blind people on mobile devices. *Proceedings of the 15th International ACM SIGACCESS Conference on*

Computers and Accessibility - ASSETS '13 (New York, New York, USA, 2013), 1–8.

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

[5] Bonner, M.N., Brudvik, J.T., Abowd, G.D. and Edwards, W.K. 2010. No-Look Notes: Accessible Eyes-Free Multi-touch Text Entry. *Proceedings of International Conference on Pervasive Computing* (2010), 409–426.

[6] Braille profiling project - RNIB: *https://www.rnib.org.uk/sites/default/files/braille_profiling.doc*. Accessed: 2017-05-05.

[7] Dandekar, K., Raju, B.I. and Srinivasan, M.A. 2003. 3-D finite-element models of human and monkey fingertips to investigate the mechanics of tactile sense. *Journal of biomechanical engineering*. 125, 5 (2003), 682–691.

[8] Frey, B., Southern, C. and Romero, M. 2011. BrailleTouch: Mobile Texting for the Visually Impaired. *Universal Access in Human-Computer Interaction. Context Diversity*. Springer Berlin Heidelberg. 19–25.

[9] Guerreiro, T., Lagoá, P. and Santana, P. 2008. NavTap and BrailleTap: Non-visual texting interfaces. *Proceedings of the Rehabilitation Engineering and Assistive Technology Society of North America Conference* (2008).

[10] Heni, S., Abdallah, W., Archambault, D. and Uzan, G. 2016. An Empirical Evaluation of MoonTouch: A Soft Keyboard for Visually Impaired People. *Proceedings of International Conference on Computers Helping People with Special Needs* (2016), 472–478.

[11] Lee, S., Hong, S.H., Jeon, J.W., Choi, H.-G. and Choi, H. 2004. Design of Chording Gloves as a Text Input Device. Springer, Berlin, Heidelberg. 201–210.

[12] MacKenzie, I.S. and Soukoreff, R.W. 2002. A character-level error analysis technique for evaluating text entry methods. *Proceedings of the second Nordic conference on Human-computer interaction - NordiCHI '02* (New York, New York, USA, 2002), 243.

[13] MacKenzie, I.S. and Soukoreff, R.W. 2003. Phrase sets for evaluating text entry techniques. *CHI '03 extended abstracts on Human factors in computing systems - CHI '03* (New York, New York, USA, 2003), 754.

[14] MacKenzie, S. and Castellucci, S. 2012. Reducing visual demand for gestural text input on touchscreen devices. *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts - CHI EA '12* (New York, New York, USA, 2012), 2585.

[15] Mascetti, S., Bernareggi, C. and Belotti, M. 2011. TypeInBraille: a braille-based typing application for touchscreen devices. *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility - ASSETS '11* (New York, New York, USA, 2011), 295.

[16] Mascetti, S., Bernareggi, C. and Belotti, M. 2012. TypeInBraille: Quick Eyes-Free Typing on Smartphones.*Computers Helping People with Special Needs* (2012), Springer, Berlin, Heidelberg. 615–622.

[17] Mattheiss, E., Regal, G., Schrammel, J., Garschall, M. and Tscheligi, M. 2014. Dots and Letters: Accessible Braille-Based Text Input for Visually Impaired People on Mobile Touchscreen Devices. Proceedings of the *International Conference on Computers for Handicapped Persons (2014)*. Springer, Cham. 650–657.

[18] Myung-Chul Cho, Kwang-Hyun Park, Soon-Hyuk Hong, Jae Wook Jeon, Sung Il Lee, Hyuckyeol Choi and Hoo-Gon Choi A pair of Braille-based chord gloves. *Proceedings. Sixth International Symposium on Wearable Computers,* 154–155.

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

[20] Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J. and Gonçalves, D. 2011. BrailleType: unleashing braille over touch screen mobile phones. *In Proceedings of the INTERACT 2011* (2011), 100–107.

[21] Perkins Braillers: *http://www.perkinsproducts.org/about-perkins-braillers*. Accessed: 2017-04-15.

[22] Romero, M., Frey, B., Southern, C. and Abowd, G.D. 2011. BrailleTouch: Designing a Mobile Eyes-Free Soft Keyboard. *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services - MobileHCI '11* (New York, New York, USA, 2011), 707.

[23] Šepić, B., Ghanem, A. and Vogel, S. 2014. BrailleEasy: One-handed Braille Keyboard for Smartphones. *Studies in health technology and*. (2014), 1030–1035.

[24] Silfverberg, M., MacKenzie, I.S. and Korhonen, P. 2000. Predicting text entry speed on mobile phones. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '00* (New York, New York, USA, 2000), 9–16.

[25] Southern, C., Clawson, J., Frey, B., Abowd, G. and Romero, M. 2012. An evaluation of BrailleTouch: mobile touchscreen text entry for the visually impaired. *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services - MobileHCI '12* (New York, New York, USA, 2012), 317.

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