Braille Messenger: Adaptive Learning Based Non-Visual Touch Screen Text Input for the Blind Community Using Braille

U B H S Udapola
Department of Statistics & Computer Science
Faculty of Science
University of Kelaniya
+94 71 415 1818
hansiudapola2@gmail.com

S R Liyanage
Department of Software Engineering
Faculty of Computing and Technology
University of Kelaniya
+94 71 649 8614
sidath@kln.ac.lk

ABSTRACT
'Braille Messenger' is an Android application (app) that is designed to facilitate the communication between users via the Braille alphabet through the medium of Short Messaging Services (SMS). This application is mainly envisaged to be used by the visually handicapped community. The proposed method is comprised of three gesture types for braille communication. The gestures can be generated with either two fingers, three fingers or six fingers. The app can be used by the blind and blind-deaf users with tactile feedback in the form of vibrations. Simple drawing patterns were selected as commands instead of long-tap and double tap operations. The app automatically suggests a word with the highest auto completion probability when a user types five characters, using vibration methods. A standard 6-bit encoding scheme was used to convert braille characters to text. A novel static algorithm was introduced in this research to detect drawn patterns by users. A feed-forward algorithm was formulated using the K-NN algorithm to detect the input fingers when a braille character is typed, and the K-Means algorithm was used to track the user's input fingers with time. 'Braille Messenger' was tested at different typing speeds. Average typing speed and accuracy were analysed on a sample of five blind users. The maximum typing speeds for the designs with two fingers, three fingers and six fingers were 5.4WPM, 9.6WPM and 18.9WPM respectively. The average typing speeds were recorded as 3.74WPM, 7.28WPM and 13.29WPM for the designs with two fingers, three fingers and six fingers respectively. An accuracy of 97.4% was recoded for braille character detection when using Bayesian Touch Distance with Nearest Neighbour and K-Mean algorithms. An accuracy of 94.86% was recorded for the drawn pattern commands with the proposed static mathematical algorithm. The proposed method currently only supports English language communication. The basic standard for Grade-II Braille alphabet commonly used by the blind community in Sri Lanka was used in this research. Future work would include implementing the app as a system wide keyboard, introducing error correction and support for other languages.

Categories and Subject Descriptors
H.5.2 [**Information interfaces and presentation**]: Graphical user interfaces (GUI) – *Input devices and strategies, screen design and user centered design.*

I.2.2 [**Applications and Expert Systems**]: Natural Language Interfaces

Keywords
Blind, Braille, Smart Mobile Devices, Text Entry Method, Universal Design.

1. INTRODUCTION
In present day, mobile phone has become an essential tool of communication in life. Among the plethora of different techniques for communication, SMS is the most pervasive. However, this method is challenging for the blind and visually impaired (VI) users. Looking at the history of the smartphone, the Ericsson GS88 (Mccarty, 2017) was the first which was manufactured with a physical keyboard in 1997. Therefore, blind users could manage to use the phone if they could physically feel the keys. The difficulties faced by the blind users increased with the advent of the Touch Sensitive Displays as they lost their sensitivity of keys that were there on a physical keyboard. A special accessibility feature for the blind community became an essential part of the mobile phone when they were introduced to the market.

There were solutions introduced by the companies that developed operating systems alongside 3rd party developers (Bonner, Brudvik, Abowd, & Edwards) (Mascetti, Bernareggi, & Belotti, 2011) (Hatzigiannakoglou & Kampouraki, 2016). All the solutions were however restricted to the technical capabilities of the smartphone. The main restriction was the number of multi-touch inputs that could be detected by the phone.

Voiceover in iOS by Apple (iPhone - Accessibility, 2017), Narrator in Windows by Microsoft (Windows phone accessibility, 2017) and Talkback in Android by Google (Android accessibility , 2017) are some of examples that were introduced by the leading software companies who developed the Operating Systems for the leading brands in the world. When the accessibility function is activated, a user should double tap on any button to execute a task instead of a single tap, as the single tap just reads screen view.

The initial stages of the proposed App before the initial testing was published in (Udapola & Liyanage, 2016). In this study, a novel method for the Blind and VI users to communicate using

the Braille Alphabet is proposed. The app features the ability to customize the UI according to the preferences of the user and the technical specifications of the device. As most of the previously developed Apps were specifically aimed for the devices with 6 multi touch inputs, 'Braille Messenger' initially checks the number of multi touch inputs supported by the device using a calibration procedure.

Even though there are several mobile apps developed for blind and VI users such as, (Bonner, Brudvik, Abowd, & Edwards) (Mascetti, Bernareggi, & Belotti, 2011) (Hatzigiannakoglou & Kampouraki, 2016) blind-deaf users were unable to use them since those apps were designed with the screen reading/Text-To-Speech (TTS) Manager. 'Braille Messenger' includes a method to facilitate blind-deaf users to read and write messages using braille with tactile feedback in the form of vibrations. Moreover 'Braille Messenger' facilitates any type of Multi-touch mobile device holders to type in Braille by introducing three different non-visual touch screen text entry methods by considering the number of touch points available in their mobiles.

1.1 Objectives

Major objectives of the proposed system is to provide a more accurate, more user friendly, high speed, user customizable typing mechanisms for the blind and blind community with different disabilities, depending on the multi-touch capability of a mobile device.

An integrated app with three different gestures was developed to achieve these objectives by introducing three different methods of typing (using one-hand design with two or three fingers and using two-hand design with six fingers). Some simple pattern drawing commands were introduced to execute basic tasks in lieu of taping on a button.

1.2 Current Study

In the current study, a self-updating blind dictionary (B-Dictionary) was developed. When a user adds a word that has more than five letters, the most matching and most frequent word is suggested. If the word is not available in the dictionary's data file, the typed word is added to the dictionary.

2. RELATED WORK

The related work for this research considered looking into the literature on current implementations for blind and VI users, UI and UX, text entry methods, input finger detection and K-NN classifications.

2.1 Universal Design

Blindness can be divided into two types, (Vision 2020 Sri Lanka, n.d.) which are full and partial blindness. Full Blindness means, inability to see anything at all, and partial blindness refers to having limited vision. The fully blind identifies the environment by touching because they do not possess any sense of dimensions or objects located around them. The mobile technology in the modern society has become such an essential part of human day to day life. Most of the mobile apps are designed for sighted people. Nevertheless, 'Design for Usability' clears that difference without considering the user's visibility (Sierra & Togores, 2012). Under the concept of 'Low Vision Mobile App Portal', some authors have provided a way to access mobile applications for visually impaired users. This research mainly focusses on the concept of "Universal Design" which is widely used in several areas/fields. Universal Design concept is closely related to the meaning of accessibility or usability. This does not imply only about the accessibility to a computer or a mobile device, but with the fast-developing technology, accessibility of the blind can be extended to include computers and mobile devices.

In 1999, the Web Accessibility Initiative [WAI] published the Web Content Accessibility Guidelines [WCAG] to improve the accessibility of disabled people. But with the development of mobile phones with touch displays, accessibility of mobile for visually impaired & blind users was fell due to the loss of physically feedback. So, some (Sierra & Togores, 2012) researchers have designed special apps for visually impaired or blind users to access the touch screen mobile devices. Low Vision Mobile Portal is one of the mobile apps that provide the facility to access important apps such as phone calls, messages, contacts, and calculator etc.

2.2 Human-Computer Interaction Design

In line with the concept of 'Mobile Accessibility', designing the user interface (UI) was given high priority. Application of Human/User -Computer Interaction Design concepts assist in designing effective User Interfaces especially for blind users (Arrigo & Cipri, 2010). Apple with iOS operating system, have considered the concepts of user experience and accessibility in mobile apps, when designing apps for the blind or VI users. (Sierra & Togores, 2012).

User experience refers to observing and analyzing the users' experiences to evaluate the effectiveness of a product design. Which is considered the most valuable aspect of Human-Computer Interaction (Marcus, 2013)

Accessibility in mobile apps generally implies that the app should be accessible by as many users as possible. For instance, narrator, voice over, voice control, speak recognition, auto text and tactile buttons are some of features that have been introduced to users with special needs to access the mobile (Craddock).

D. McGookin et al. have suggested a set of guidelines for mobiles with touch displays (McGookin, Brewster, & Jiang, 2008) that can be applied to maximize accessibility.

2.3 Text Entry Methods

All the apps that have been described above relate to the blind or VI people. But these apps were rarely used by blind users and regarding to an online survey (Leporini, Buzzi, & Buzzi, 2012), blind users mostly use mobiles (iPad/iOS) to take mobile calls (92.7%), read SMS (90.9%), write SMS (87.3%) and to listen to music (80%). But most of them (52.7%) are not satisfied about the usability of the keypad and 72.7% of blind users preferred an editing mode with a single tap.

According to Leporini et al. blind users prefer to the QWERTY keyboard rather than using multi tap keyboards (Leporini, Buzzi, & Buzzi, 2012). However, there are many text entry methods such as, Multi Tap. Nav Touch, Braille Type (Oliveira, Guerreiro, Nicolau, Jorge, & Goncalves, 2011) and Robust Entry Technique which were used in Eyes-Free Text Entry type in No-Look Notes (Bonner, Brudvik, Abowd, & Edwards).

QWERTY

In the basic QWERTY method, a user has to move his/her finger on the touch keypad, and then the system reads the letter that the user has touched, and the double tap types the letter.

There is another approach using the digital QWERTY keyboard that is used in iPhones known as 'Voiceover' (Oliveira, Guerreiro, Nicolau, Jorge, & Goncalves, 2011). In this approach the user

should set his/her finger on the desired key and then tap anywhere on the screen with a second finger. Afterwards, lift the first finger and then double tap anywhere on the screen. These are the steps that must be followed to enter a letter using voice over (iPhone - Accessibility, 2017).

Multi-Tap

In multi-tap method also, a mechanism similar to QWERTY is employed. But here the targeted number of keys on screen are less than QWERTY. The user must select the group of keys by double tap and multi-tap to select a letter.

Navigational Touch

In the method of navigational touch, the user needs to move the finger up and down (vertically) to select a group of letters and move the finger left and right (horizontally) to select a letter. Here letters are grouped into five and each group starts with a vowel 'a, e, i, o, u'.

Eyes-Free Text Entry

In the eyes-free text entry method, a user must tap on a group to enter a letter. As an example, if user needs to input 'B' then user need to select 'ABC' group by touching on that particular group of letters. To select a specific letter through the group or to open a group the user should tap a second finger anywhere on the screen. The next screen shows all the letters in a given group category vertically. Then user can move his/her finger up or down to select the individual letter. The user must tap his/her second finger anywhere on the screen to confirm the selection.

Braille Type

The Braille type method is the most familiar method for the blind and variants of Braille type input method can be found. This is known as the first approach with having less number of screen targets and here each cell known as braille cell and user should long touch on cell to mark the cell and the double tap anywhere on the screen to tell that single braille character has finished the typing. When user marked a cell then it denotes a single bit otherwise 0.

Braille type has been applied in (Alnfiai & Sampalli, 2016) with an extension known as 'SingleTapBraille'. In the Braille type design, a user should mark on the virtual cell area. But, in SingleTapBraille design, user does not need to find a specific location or area to mark the cell (Oliveira, Guerreiro, Nicolau, Jorge, & Goncalves, 2011). Here they developed their algorithm by using the factors of Number of dots in each character, the x, y coordinates of the touch points and distances between two points. As an example, if user need to input 'b' then user need to mark two points vertically anywhere on the screen. The relationship per the number of dots activated by input are as follows:

- Character with 1 tap anywhere on the screen - always known as letter 'a' or number 1.
- Character with 2 taps - |X1-X2|<error; D is a specific value for this status.
- Character with 3 taps - |X1-X2|<error; |Y1Y3|<error; D is a specific value for this status.
- Character with 4 taps - |X1-X2|<error; |X3-X4|<error; X2<X3; |Y1-Y3|<error; |Y2-Y4|<error; Y3<Y2, D is a specific value for this status.
- Character with 5 taps - |X1-X2|<error; |X3-X2|<error; |X4-X5|< error; |Y1-Y4|<error; |Y2-Y5|< error; D is a specific value for this status.

So, using the above results the typed character can be classified and output. A 'Sliding Rule' is also used in this method to run special characters such as, New Line [Enter], White Space, Backspace, Switch keyboards among Uppercase keyboard, Numerical keyboard, lowercase keyboard and symbols (punctuation marks) key board.

Furthermore, there are more approaches to type braille letters on screen. One of them is 'PerkInput Text Entry Method' (Azenkot, Wobbrock, Prasain, & Ladner, 2012). This is similar to Perkin Braille Device input method. A user must put 6 fingers on the screen and long touch on screen to add the point. The screen is shown in Figure 1.

Figure 1 - PerkInput method in touch screen

In PerkInput, Azenkot et al. have introduced the theoretical concept of IFD 'Input Finger Detection' which is based on a multi-touch signal detection technique. In this method, a single finger input is denoted as one bit '1' or '0'. These methods can be used with either both hands or with a single hand. If the user needs to use both hands, then the user has to touch on screen only once to enter a single braille character, otherwise a user has to touch twice to enter a single braille character.

There is another new technique to type in braille in touch screen which known as 'TypeInBraille'. In this technique (Mascetti, Bernareggi, & Belotti, 2011) a user needs to input pairs of dots at a time as shown in figure 2.

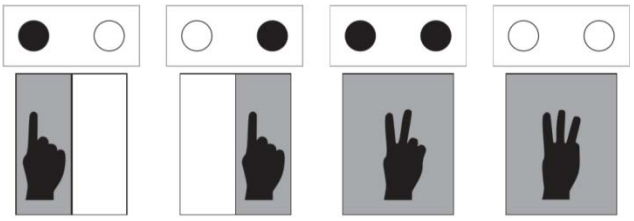


Figure 2 - Technique of TypeInBraille

In 'TypeInBraille' method a user has to touch three times to enter a single letter. Usually a single braille letter is denoted with a 3 x 2 matrix. A user must fill a single row in this method. When considering a single row, a single dot is denoted using a single touch in left/right side. If a row has both dots, then two fingers touch and no dots for a row are denoted by three fingers. A flick/slide rule is used to denote 'End Character' and 'Blank space'.

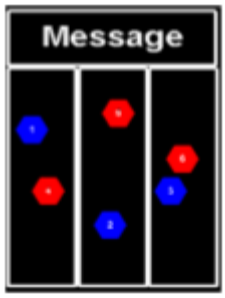


Figure 3 - Technique of BrailleOne text entry method

As well as 'BrailleOne' (Hatzigiannakoglou & Kampouraki, 2016) the research team of Paul, also an extension of this techniques. In this method, a user needs to touch twice to enter a single letter. In blue it represented one, two and three inputs. And in red it represented four, five and six. This method also has done in a previous research (Azenkot, Wobbrock, Prasain, & Ladner, 2012) however using in different way of Input detecting. In BrailleOne they input was detected by view wise. Assume a user names a Linear Layout A, B & C from left to right, then if a user taps on Linear Layout A then it detects it as 1 or 4 inputs due to group of categories (1,2,3 & 4,5,6). Similarly, if user taps on Linear Layout B then it detects it as 2 & 5. But in 'PerkInput Braille' (Azenkot, Wobbrock, Prasain, & Ladner, 2012) method use IFD technique. Here once user touch on screen with his/her 3 fingers then it detects its (x, y) coordinates of reference point and then using Maximum Likely Hood function they corrected the next time touch points with the previously touched reference point.

'Braille Key' (Subash, Nambiar, & Kumar , 2012 4th International) is another approach of typing braille. In this approach, they have divided the screen into four parts as 2 x 2. The first line is reserved for text entry and the left area, one touch selects the point one, two touches, the point two, and long touch, the point three. The same applies to the right, for points four, five and six. The ENTER and DELETE buttons are on second line.

In Google Play Store, there is an alternative keyboard known as "Swift Braille" by the Swift Team. In this method, the user need to draw the patterns from one dot to another.

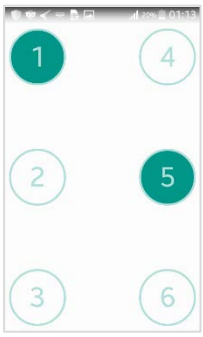


Figure 4 - Typing letter 'e' using Swift Braille

Figure 4 shows how user type braille character 'e' using Swift Braille. Here user starting at dot 1 position and then move his/her finger to the dot 5.

First two methods of QWERTY and Multi Tap that are described above are much difficult for blinds that is why researchers introduced new techniques just like third and fourth methods that are mentioned and they have analysed each method on factors of fast/speedy and easy/user friendliness.

Nav Touch & No-Look Notes are much different when compared with normal text entry methods of QWERTY & Multi Tap. Per their analysed data No-Look Notes (Bonner, Brudvik, Abowd, & Edwards) has proved their method is faster than 'Voice Over method' (Text entry speeds are 0.66 WPM for Voiceover and 1.32 WPM for No-Look Notes) and QWERTY (Oliveira, Guerreiro, Nicolau, Jorge, & Goncalves, 2011) is the fastest method (2.1 WPM with 0.7 of SD) of typing. Slowest method (1.49 WPM with 0.43 of SD) is typing braille (a user need to touch 6 times at maximum to enter a single braille character). But typing braille was the most accurate way of typing letters. However, they have come up with some difficulties for each method and 'Timeouts & Lose track of text' were identified difficulties for typing braille.

In (Siqueira & Silva, 2016) has analysed the most of braille methods which exists on today. Per their research 'Braille Touch' the method of typing braille using 6 fingers simultaneously and the 'TypeInBraille' are the fastest way of braille typing. But 'Braille Touch' has the maximum of error rate and 'PerkInput' method is most accurate way.

2.4 Input Finger Detection (IFD)

Input through the touch screen can be modelled for transmission of information through the noisy channel (Azenkot, Wobbrock, Prasain, & Ladner, 2012). In IFD mechanism at the first user should set his/her *n* reference points on the screen. As an example, when user insert long press by inserting *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 user's *i*th finger touch the screen for pre-given target point, then bit of that point will be one (1) otherwise will be zero (0). As mentioned earlier touch input given to the device with some inconsistency, since the user will not hit exact reference point with every subsequent touch. The device receives the noisy, encoded message which analogous from inconsistency through noisy channel and the input method decodes it using their detection algorithms.

There are three sources of noise in their model:

1. Hand repositioning. When the user repositions the hand on the screen, the current touch points are no longer near the reference points. This was addressed by simply set new reference points that reflect the new position of her hand.

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. The Maximum Likelihood (ML) to detect which finger corresponds to which point while accounting for the distribution of points around the target reference points.

3. Hand drift. When user touches the screen repeatedly usually our fingers moving little bit (drifting) from the target point. This was addressed by tracking the reference points after each touch to minimize the error of decoding.

2.5 Character Classification

Usually a Braille character is denoted as a 3 x 2 matrix and it is numbered 1 to 6 from top to bottom, left to right as shown in Figure 5. In PerkInput typing method fingers are numbered as in Figure 6.

Figure 5 - Braille Cell

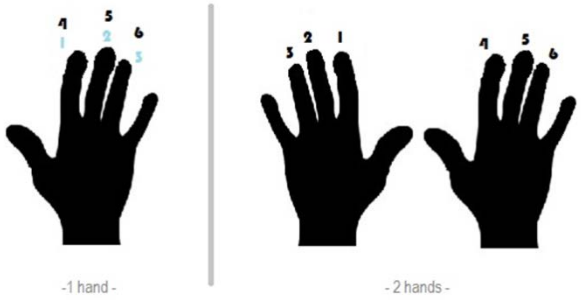


Figure 6 - Mapping Fingers in one hand and both hands

2.6 Tracking Reference Points

When the Hand Drift was addressed in the previous Section 2.4, ML functions were used for target selection criteria. They have assumed that users keep their hands next to the reference points. But this is not true in reality. Therefore, in this solution 'First-Order Phased-Locked Loop' has been used to track references points.

After the input finger is detected, where each touch reference points were moved by a fraction of distance between touch point and the corresponding reference point. Here they assumed that each individual finger touch is correlated with the other fingers. Therefore, the concept has been formalized into the following equation:

$$R_{n+1} = k.C.E_n + R_n$$

Equation 1

, where R_n is the vector of references points at time *n*, E_n is the vector of differences between references point and touch point at time *n*, *k* is a scalar constant which known as '*adaption coefficient*' (Smaller values of *k* are reducing the effect of tracking) & *C* is the '*correlation coefficient matrix*'.

k and *C* values, are they derived through the experiments and they as follow;

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

Equation 2

2.7 Sensor Vibration

Blind-deaf communication is more important when a message is received through a smart touch phone as most apps are built only to read the message through system voice feedback.

Blind-Shell (Svobodnik, Novak, & Cerman, 2013) is a launcher app that is created especially for the visually impaired. Through this facility is provided for the blind to perform basic operations such as SMS, Contacts and setting an alarm.

V-Braille (Jayant, Acuario, Johnson, Hollier, & Ladner, 2010) is a technique that is capable of reading Braille letters through a fully touch screen using vibrations. Jayant et al. have used different signals for Braille cells. The Figure 7 shows how vibration signals work when reading Braille letters. The dotted line has low vibration than a straight-line vibration.

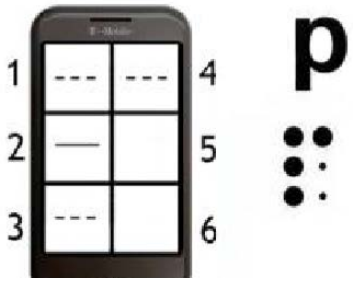
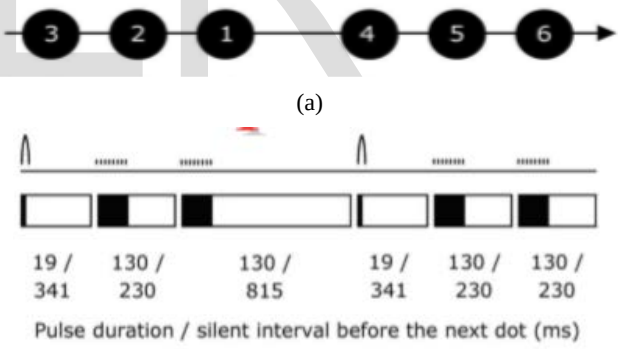


Figure 7 - Reading the lowercase letter 'p' (V-Braille) (Jayant, Acuario, Johnson, Hollier, & Ladner, 2010)

'Braille Scan', 'Braille Rhythm' and 'Braille Sweep' (Rantala, et al., 2009) also embody different methods to read Braille characters. Braille Scan is similar to V-Braille above mentioned, where when a user moves his finger from top to bottom the vibration happens only once per dot and again go back to the top and the last 3 dots of braille character in 2nd column are read. 'Braille Sweep' method is almost like the Braille Scan method but here the braille dots' positioning is different from the Braille Scan method. Here dots are positioned as three, two and one dots horizontally as well as four, five and six placed horizontally. Here the reading direction is different from the Braille Scan as this method uses vertical finger movement to read the text. Figure 8–(a) represents the numerical order of braille dots for Braille Rhythm method. In the Braille Rhythm method, the characters are read using temporal tactile patterns. This produces tactile pulses in numerical order from one to six (1-6/left to right).



(a)

(b)

Figure 8 - (a) Braille Sweep numerical order & (b) Braille Rhythm (Braille Sweep) (Rantala, et al., 2009)

Figure 8 – (b) denotes the pattern/ Rhythm for braille character 'c'. Character 'c' consists of dots one (1) and four (4). However, one and four has a higher pulse than other dots. Due to their data analysing part mean recognition accuracies and the mean reading time (Character Per Seconds) are as follows. 'Braille Scan' is more accurate, but 'Braille Rhythm' is a much faster method.

Table 2-1 - Analysis of Data after 3 sessions (Rantala, et al., 2009)

Method of Braille Reading	Recognition Accuracy (%)	Reading Time (cps)

Braille Scan	97	0.18
Braille Sweep	91	0.20
Braille Rhythm	92	0.27

3. METHODOLOGY

3.1 Typing Braille Characters

Under the literature from the previous chapter, several types of text entry methods were developed for touch screens to blinds or VI people by researchers. Among them 'Perk-In-Brailler' (Azenkot, Wobbrock, Prasain, & Ladner, 2012) is the fastest way of to insert characters. Nevertheless, due to the proposed system, three types of different gestures were developed to use with two, three or six fingers.

Design A: Type a single braille character using two fingers & needs to tap thrice to insert a single braille character. Target the devices which have only basic multi-touch capability of points of two.

Design B: Type a single braille character using three fingers & needs to tap twice to insert a single Braille character. Target the devices which have multi-touch capability with less than six points but greater than two points.

Design C: Type a single braille character using 6 fingers & by a single tap can insert a single Braille character. Target the devices which have best multi-touch capability of points of ten or more than 6 points.

With the customizable UIs, the restriction of hand positioning on the screen in Design B & C is avoided. Therefore, user has the freedom to register the reference points (Dots of a Braille character) as they were preferred. Following Figure 9 shows sample UI with references points (Design B & C) for each design.

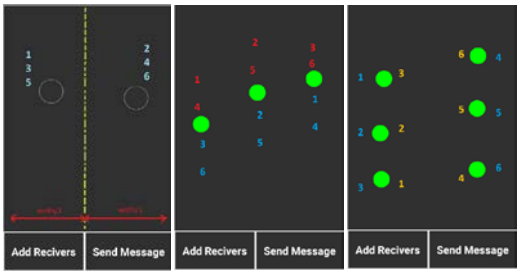


Figure 9 - Text Entry methods (Design A, B, C)

3.2 Input Finger Detection

In each design, tapped finger need to be identified and two different ways were used in design A and design B&C. In design A, simply screen is divided into two spaces and if left side is tapped, then it counted as left column and if right side is tapped then it counted as right column. And if a row has no marked cell then draw pattern command to denote empty cell row. Thrice a time, inserted braille code is converted into the alphanumeric text characters. Following Figure 10 demonstrates an example of inserting a braille character 'm'.

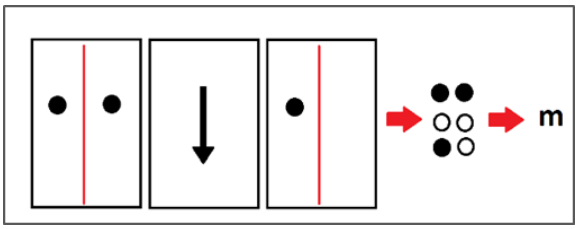


Figure 10 - insert braille character 'm'

With the increases of number of fingers which were used to insert a braille character in design B & C, above mentioned simple method cannot be used further. Therefore, more user friendly and accurate method is introduced. For that, K-NN classification algorithm is used to detect finger which is inserted by the user.

3.2.1 K-NN Classifier

K-NN classifier is a non-parametric method that is used to classify the objects. In design B, three different classes have been defined and user's registered references points have taken as centroid of each class.

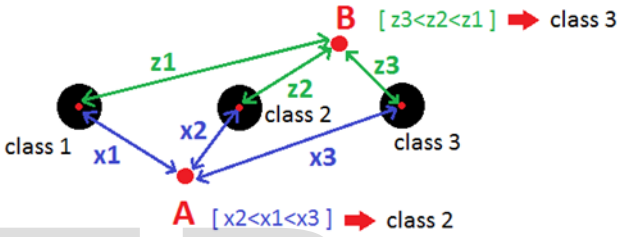


Figure 11 - Classify Point A & B using Nearest Neighborhood algorithm

Different types of distances have been used with K-NN classification algorithm to calculate distance between user touch points and reference points, namely Euclidean distance, city block distance and Bayesian touch distance.

Euclidian Distance

Let A (x, y) be the user touch point and registered reference point P_i (x_i, y_i) for class i (where i=1,2,3). Then calculate Euclidean distance using following equation.

$$d(A, P_i) = \sqrt{(x_i - x)^2 + (y_i - y)^2}$$

Equation 3

City Block Distance

Let B (x, y) be the user touch point and registered reference point P_i (x_i, y_i) for class i (where i=1,2,3). Then calculate city block distance using following equation.

$$d(B, P_i) = |x_i - x| + |y_i - y|$$

Equation 4

Bayesian Touch Distance

s- coordinates of the finger touch, t- target, W- Width of the t, c- center of the t

α & σ_a^2 are constants which they find out through separate experiments.

$$BTD(s,t) = \frac{(s-c)^2}{2(\alpha W^2 + \sigma_a^2)} + \frac{1}{2} ln(\alpha W^2 + \sigma_a^2)$$

Equation 5 (Zhai & Bi, 2013)

With some simplifications, they have generalized above equation up to 2-dimentional circular targets. Also, here they have used the proved results which 'distribution of 2-dimentional target selections approximately follow a bivariant 'Gaussian Distribution'. Then above equation has become to $BTD_2(s, t)$ as follows.

$$BTD_2 = \frac{1}{2}\left[\frac{(s_x-\mu_x)^2}{\sigma_x^2}+\frac{(s_y-\mu_y)^2}{\sigma_y^2}\right]+ln\sigma_x+ln\sigma_y$$

Equation 6 (Zhai & Bi, 2013)

Then they have replaced μ_x, μ_y, σ_x & σ_y with some estimations as follows.

$$BTD_2 = \frac{1}{2}\left[\frac{(s_x-c_x)^2}{\alpha_x d^2+\sigma_{a_x}^2}+\frac{(s_y-c_y)^2}{\alpha_y d^2+\sigma_{a_y}^2}\right]+\frac{1}{2}ln\left(\alpha_x d^2+\sigma_{a_x}^2\right)+\frac{1}{2}ln\left(\alpha_y d^2+\sigma_{a_y}^2\right)$$

Equation 7 (Zhai & Bi, 2013)

Where α_x, σ_{ax}, α_y & σ_{ay} are constants that they have obtained by separated experiments. Those values are as follows.

Table 3-1 (Zhai & Bi, 2013)

α_x	$\sigma_{a_x}^2$	α_y	$\sigma_{a_y}^2$
0.0075	1.68	0.0108	1.33

Zhai & Bi (Zhai & Bi, 2013) has introduced this distance for target selection. When converted that model into my problem model, targets are known as registered reference points. That distance can be calculated using Equation 7. d (width of target) in Equation 7 is taken as 1 for each reference points.

Among the distances which are mentioned above, BTD is selected to classify the class using Nearest Neighbour Classifier due to the higher accuracy that have been recorded.

3.3 Tracking Finger Points

Cause of finger drifting there would be a greater error between previously registered references points and the currently touched point d. Therefore, two different algorithms were used to update reference points. They were K-Mean and First-Order Phased-Locked loop algorithms.

K-Means algorithm

K-means algorithm is generally used to cluster noisy data, to constraint data to lie on the surface of a high dimensional unit sphere and for directional noisy data. General k-mean algorithm is an iteratively running process to partition 'n' number of observed data into number of given groups. (Ramler, 2008) Moreover, among different types of clustering algorithms K-Mean is the most simplest algorithm (Complexity - O(n)) and, K Means is found to work well when the shape of the clusters is hyper spherical (Kaushik, n.d.). Even though KNN classifier known as a *lazy learner* By combined with K-Mean centroid of the clusters updated and classes

Let's assume that observed references points (centroid of each clusters) at time n are $\hat{f}_{1,n}$, $\hat{f}$ $\hat{f}_{2,n}$, $\hat{f}$ $_{3,n}$, $\hat{f}$ $_{4,n}$, $\hat{f}$ $_{5,n}$, $\hat{f}$ $_{6,n}$. In General form, this can be denoted as $\hat{f}_{i,n}$ for i = 1, 2, 3, 4, 5, 6 and let's take d_n as the newly taped point and assume it belongs to the class j then by k-mean algorithm updated the reference point (centroid of the class j) $\hat{f}_{j,n+1}$ as follows;

$$\bar{r}_{j,n+1} = \left(\frac{m\bar{r}_{j,n}+d_n}{m+1}\right)$$

Equation 8

Where $\bar{r}_{j,n}$ is; $\bar{r}_{j,n} = \frac{\sum_{k=1}^{m} r_{j,k}}{m}$; $m \leq n$;

m – number of members belongs to class j at time n.

First-Order Phased-Locked Loop

First-order phased-locked loop was used by Azenkot (Azenkot, Wobbrock, Prasain, & Ladner, 2012) and Equation 1 & Equation 2 were used to updated references points. In this case, they have considered each finger are correlated with other fingers.

Both algorithms' accuracies have been equal and therefore K-mean algorithm was selected as reference points tracking algorithm cause of its simplicity.

3.4 Sense of vibration to read text

When 'Blind-Deaf' mode is activated, vibration patterns were used to read text and to give the feedback of typed text. The 'Braille Rhythm', a vibration technique (Rantala, et al., 2009) was used to input character and to read character. Following time periods and breakpoints patterns were implemented to build vibration patterns for each braille character.

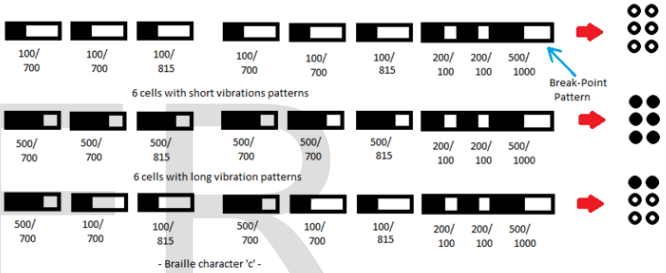


Figure 12 - Vibration Patterns with Break-Points

In this approach, basically 7.13 - 9.53 (seconds) time was taken to read a character. In Chapter 7, I have described the correctly identifying rate with participation of pseudo blind-deaf.

3.5 Braille Character Classification

In braille conversion, 6-bit (Figure 5) encoding scheme has been used to convert braille to text and sixty-three (63) braille characters which categorized under 7 lines in braille system has been used in 'Braille Messenger' including nineteen (19) punctuation marks & ten (10) digits. (Appendix)

The numerals 0-9 and the English characters "a, b, c, d, e, f, g, h, i, j" has the same braille characters. Numerals are distinguished from the above English characters with the presence of the "⠼" braille character. If "⠼" character appears prior to a braille character, it must be considered as a number (activate number mode) and after every WHITE SPACE or NEW LINE number mode will turn off.

3.6 Detecting Drawn Patterns

3.6.1 Using Gesture Detector

In Android Developing Environment there is an inbuilt function to detect draw patterns which known as 'Gesture Detector'. In this approach, there was an app called 'Gesture Builder' which developed by Google to save patterns as .txt file and then load that .txt file into Gesture Library in Android Studio. Then calculate the prediction score for drawn pattern compared to saved gestures.

When returning predicted pattern, it delayed around 1.5 seconds. Also, accuracy of the predicted pattern from Android-inbuild feature was less than the accuracy of predicted pattern from mathematical algorithm that was introduced in this research. Cause of prediction delay most of the times, braille inserting task not performed correctly.

3.6.2 *Using Static Mathematical Algorithm*

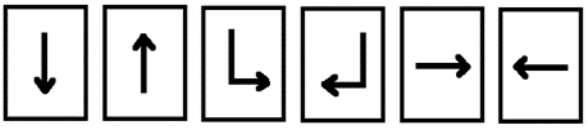


Figure 13 - Drawing Commands A, B, C, D, E, F

The proposed mathematical algorithm consists of the following steps to detect drawing pattern commands.

1. Store the all the points of drawing commands
2. By considering first and the last point of drawing command, check whether drawing lines moving side. Here considered slope between first and last points as well as difference between 2 points.

Take *difx* as the difference between the x coordinates of the first and last points, and *dify* as the difference between the y coordinates of the first and last points. Let *m* be the magnitude value of the gradient of the line drawn between first and last points of drawn command. The right, left, up, down and angle variable can be set using the following criteria as shown in Figure 14 – a;

If m less than or equal 0.5

    If difx is negative

        Right is True

    Else

        Left is True

    End If

Else If m less than 3.0

    Angle is True

Else

If dify is negative

        Down is True

    Else

        Up is True

    End If

End If

End If.

3. If ANGLE is true, then check for command C & D, if left is true then check for command F, if right is true then check for command E, if down is true then check for command A, if up is true then check for command B.
4. As examples, 'Checking for command C' (Figure 14 - concept of detecting drawing command patterns - b) & 'Checking for command B' have been considered.
    a. Checking for command C – Here find out the breaking point as shown in (Figure 14 - concept of detecting drawing command patterns-b). x-coordinates of pattern before the breaking point should lie between the [x_1-e, x_1+e] limit and y-coordinates of pattern after the breaking point should lie between the [y_n-e, y_n+e] limit. Otherwise variable OUTOFSHAPE set to be true. Moreover, here have been checked for reverse drawn lines and if it was, REVERSE set to be true.
    b. Checking for command B – Here x-coordinates of all points in drawn line should lie between the [x_1-e, x_1+e] limit otherwise OUTOFSHAPE set to be true. And here also have been checked for reverse drawn lines and if it was, REVERSE set to be true.
5. Then by using the Boolean values of variables UP, DOWN, LEFT, RIGHT, ANGLE, REVERSE & OUTOFSHAPE detect the command or not and if it was a command then returned the command name.

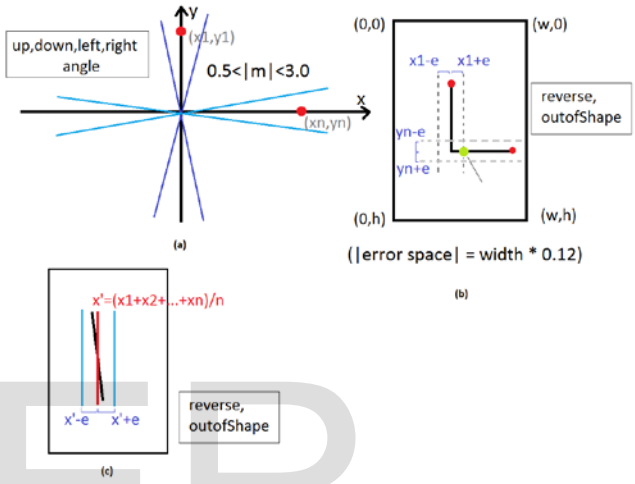


Figure 14 - concept of detecting drawing command patterns

4. RESULTS AND DISCUSSION

In this section, the proposed design with three customized gestures are discussed. Figure 9 shows Text Entry methods for the Designs A, B, and C. In this scenario, different algorithms were tested to detect the input finger for design B and design C. K-NN classifier with K=1 (Nearest Neighbor Classifier) using different distance measurements were attempted.

BTD, Euclidean and City Block three different distance measures that were used to calculate the distances in the Nearest Neighbor classifier. The app was implemented and tested for speed of typing, average typing speed and for accuracy with a sample of 5 blind users. Highest accuracy was recorded for the BTD at 97.54%. For other distances, Euclidean & City-Block distances observed accuracy at 95.38% & 92.3% respectively. Moreover, we can conclude that the accuracy level of the BTD is in between 93.63% and 97.14% at 95% level of confidence.

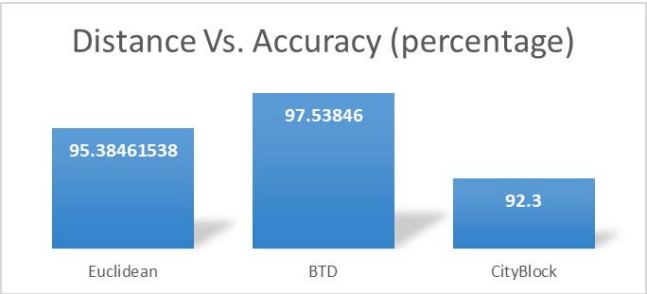


Figure 15 - Distances Vs text entry accuracy rate

After the evaluation of accuracy, Bayesian Touch Distance has been selecting as the distance for nearest neighbour classifier. Then accuracies were tested against the method which is used Maximum Likelihood function and 96.89% of accuracy was recorded for ML. Since accuracy of Nearest Neighbour Classifier with BTD was higher than the method that used ML function, the Nearest Neighbour Classifier with Bayesian Touch distance was chosen to implement in the algorithm to detect input finger when inserting braille characters for design B and C.

4.1 Text Entry Speed

After 5 sessions of testing with participation of 5 users, the maximum speed of each text entry was recorded at 5.4WPM, 9.6WPM and 18.9WPM for designs with finger 2 (Design A), 3 (Design B) and 6 (Design C) respectively. Results of the experiment were concluded that the typing speed of Design-A (one-hand 2 finger) lies between 3.37WPM and 4.11WPM, Design-B (one-hand 3 finger) lies between 6.58WPM and 7.98WPM and Design-C (two-hand 6 finger) lies between 11.69WPM and 14.89WPM at 95% level of confidence.

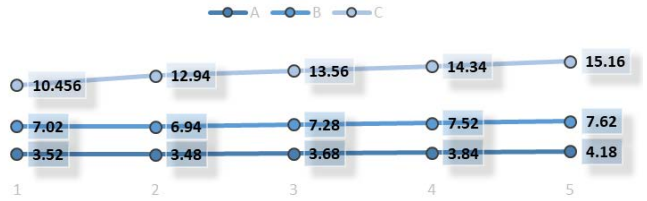


Figure 16 - Entry rate over sessions

As is observed from the graph, text entry speeds of each design have gradually increased over the number of sessions. Each design is observed highest speed at the session 5 and at that session speed has been increased by 19%, 9% and 45% when compared to the session 1 for design A, B, and C respectively. Which is concluded a higher learnability of my proposed app.

4.2 Accuracy of Text Entry

In this section, uncorrected error rates for each design is analysed over the time (number of sessions). Below Figure 17 demonstrates the error rates over the time. For the design A, 99.99% of accuracy (error rate = 0.01%) was recorded. Likewise, for average uncorrected testing, 2.46% of uncorrected error rate for design C and 1.15% of uncorrected error rate for design B were recorded. And, as is observed from the graph, there is a decline in Design-C and B. This showed that, this approach improves the learnability of the proposed app.

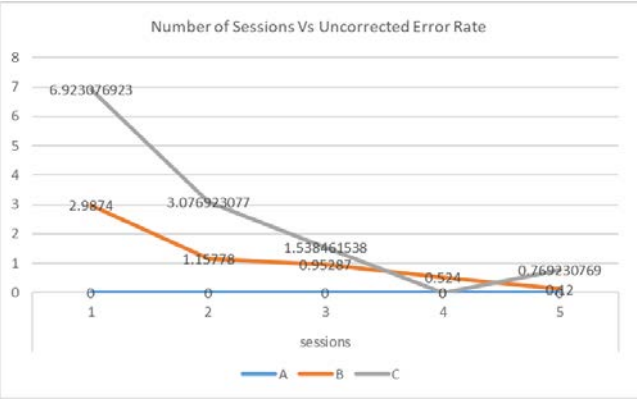


Figure 17 - Uncorrected Error rate over time

4.3 Drawn Pattern Detection

In proposed solution, some of drawing pattern commands were introduced to execute commands. As Android was chosen for the development, there was an in-built functionality to detect drawing patterns. So, in this study, the accuracy for the novel mathematical algorithm was measured and checked against its accuracy against the Android in-built functionality of 'Gesture-Detector'. Figure 18 represents the accuracy level for the 'Drawn Pattern Commands' for the mathematical algorithm that was introduced in this research while Figure 19 represents the accuracy level for the 'Drawn Pattern Commands' for the Android in-built functionality of 'Gesture-Detector'.

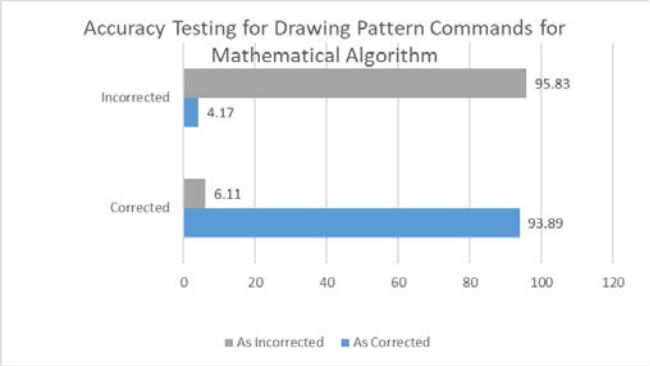


Figure 18 - Accuracy rate of drawing command patterns for mathematical algorithm

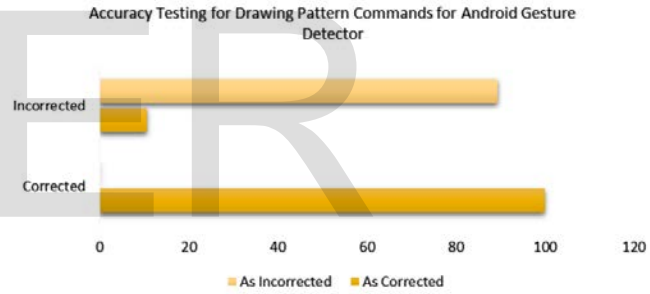


Figure 19 - Accuracy rate of drawing command patterns for Android Gesture Detector

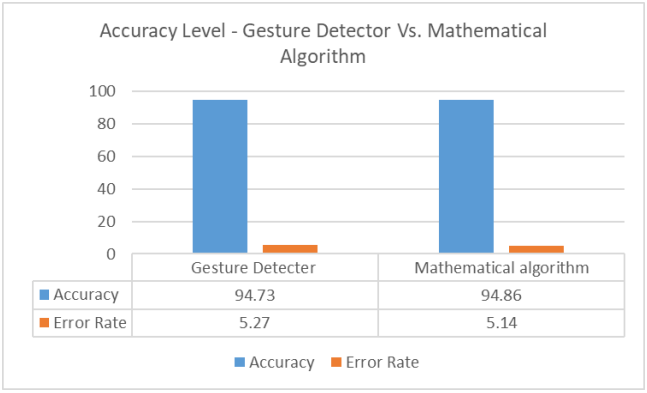


Figure 20 - Accuracy Level of Gesture Detector over Mathematical Algorithm

When comparing the accuracy level of mathematical algorithm over the Android's inbuilt feature 'Gesture Detector' there is no significant difference between them. Also, 94.86% accurately detect the drawn patterns from the proposed novel mathematical algorithm. But when considering the time taken to detect, Gesture

Detector took more time to detect drawn pattern commands than the proposed method. Moreover, using gesture detector leads to increased error rates in typing when tested on blind users. Therefore, novel mathematical algorithm is used to detect drawn pattern commands.

5. CONCLUSION

'Braille Messenger' is an integrated mobile application which is designed to send and receive text messages using Braille via short message service. The app can support both blind and the blind-deaf communities.

The app is designed with the user in mind, giving the user a chance to select any gesture what they preferred and through this entry method could to reach higher typing speed than other ordinary typing methods that blinds using.

K-NN and K-Means algorithms were employed with Bayesian Touch Distance to classify the inserted braille character on gesture. A novel static algorithm was also implemented to execute drawn pattern commands. The app demonstrated a higher accuracy for character and pattern detection and recognition. Furthermore, the app is able to predict words when a user types more than 5 letters.

The Braille Messenger recognized and converted the Braille to text with an accuracy of 97.54% while the drawn pattern commands were detected and recognized at an accuracy of 94.86%. When considered about the learnability, a higher rate of learnability was observed within a limited number of sessions.

In this research, 'Braille Messenger' was implemented as an integrated application to send and receive text messages using SMS Manager. But this customizable typing views can also be developed as a customizable keyboard. Thus, enabling a blind user to type with Braille when using any application on the phone.

Currently the app supports only English language, which can be extended to more languages like Sinhala and Tamil.

Error correction functionality is another important task that can be useful for the blind community when typing Braille. In future, the app can be implemented with an error correction functionality in an effective way.

6. References

Alnfiai, M., and Sampalli, S. (2016). SingleTapBraille: Developing a text entry method based on braille patterns using a single tap. *The 11th International Conference on Future Networks and Communications (FNC 2016).*

Android accessibility . (2017, June). Retrieved from Android - Google Support: https://support.google.com/accessibility/android/answer/6006564?hl=en

Arrigo, M., and Cipri, G. (2010). Mobile Learning for All. *Journal of the Research Center for Educational Technology (RCET), 6*(1), 94-102.

Azenkot, S., Wobbrock, J. O., Prasain, S., and Ladner, R. E. (2012). Input Finger Detection for Nonvisual Touch Screen Text Entry in Perkinput . *Graphics Interface Conference.* Toronto, Ontario, Canada.

Bonner, M., Brudvik, J., Abowd, G., and Edwards, W. K. (n.d.). *No-Look Notes: Accessible Eyes-Free Multi-Touch Text Entry.* Atlanta, USA.

Craddock, G.M. ed., 2003. Assistive Technology: Shaping the Future: AAATE'03 (Vol. 11). IOS press.

Hatzigiannakoglou, P. D., & Kampouraki, M. T. (2016). An Accessible Keyboard for Android Devices as a Means for Promoting Braille Literacy . *IJIM, 10*(2), 77-78.

iPhone - Accessibility. (2017, June). Retrieved from Apple official web site: https://www.apple.com/accessibility/iphone/

Jayant, C., Acuario, C., Johnson, W. A., Hollier, J., & Ladner, R. E. (2010). *VBraille: Haptic Braille Perception using a Touch-screen and Vibration on Mobile Phones.* Orlando, Florida, USA: ASSETS'10.

Leporini, B., Buzzi, M. C., & Buzzi, M. (2012). *Interacting with Mobile Devices via VoiceOver: Usability and Accessibility Issues.* OZCHI.

Marcus, A. (2013). Design, User Experience, and Usability: Design Philosophy, Methods, and Tools. *Second International Conference, DUXU 2013.* Las Vegas, NV, USA, July 21-26, 2013.

Mascetti, S., Bernareggi, C., & Belotti, M. (2011). *TypeInBraille : A Braille-based Typing Application.* Dundee, Scotland, UK: ASSETS'11.

Mccarty, B. (2017, May). *The History of Smartphones.* Retrieved from Thenextweb: https://thenextweb.com/mobile/2011/12/06/the-history-of-the-smartphone/#

McGookin, D., Brewster, S., & Jiang, W. (2008). *Investigating Touchscreen Accessibility for People with Visual Impairments* . Lund, Sweden: NordiCHI 2008.

Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J., & Goncalves, D. (2011). *Blind People and Mobile Touch-based Text-Entry: Acknowledging the Need for Different Flavors.* Dundee, Scotland, UK: ASSETS'11.

Ramler, I.P., 2008. Improved statistical methods for k-means clustering of noisy and directional data. Iowa State University.

Rantala, J., Raisamo, R., Lylykanas, J., Surakka, V., Raisamo, J., Salminen, K., Hippula, A. (2009). Methods for Presenting Braille Characters on a Mobile Device with a Touchscreen and Tactile Feedback. *IEEE TRANSACTIONS ON HAPTICS, 2*(1), 28-39.

Sierra, J. S., & Togores, J. S. (2012). Designing Mobile Apps for Visually Impaired and Blind Users. *ACHI 2012 : The Fifth International Conference on Advances in Computer-Human Interactions.*

Siqueira, J., & Silva, C. R. (2016). Braille Text Entry on Smartphones: A Systematic Review of the Literature. *IEEE 40th Annual Computer Software and Applications Conference.*

Subash, N., Nambiar, S., & Kumar , V. (2012 4th International). Braillekey: An alternative braille text input system: Comparative study of an innovative simplified text input system for the visually impaired. *Intelligent Human Computer Interaction (IHCI).*

Svobodnik, P., Novak, D., & Cerman, M. (2013). *BlindShell - User Interface for Visually Impaired Users.* Karlovo Namesti 13, Prague 2 120 00, Czech Republic .

Udapola, U. S., and Liyanage, S. R. (2016). Braille Messenger: SMS Sending Mobile App for Blinds Using Braille. *Kelaniya International Conference on Advances in*

Computing and Technology (KICACT - 2016), (pp. 68-69). Colombo.

Vision 2020 Sri Lanka. (n.d.). Retrieved 5 20, 2016, from http://www.vision2020.lk/blindness&vision.html

Windows phone accessibility. (2017, June). Retrieved from Microsoft official web site: https://support.microsoft.com/en-us/help/10664/windows-phone-accessibility-on-my-phone

Ye, H., Malu, M., Oh, U., & Findlater, L. (2014). *Current and Future Mobile and Wearable Device Use by People With Visual Impairments* . Toronto, Ontario, Canada.: Copyright © 2014 ACM 978-1-4503-2473-1/14/04.

Zhai, S., & Bi, X. (2013). Bayesian Touch – A Statistical Criterion of Target Selection with Finger Touch. *UIST'13*. St. Andrews, United Kingdom.