

**ACCESSIBLE TOOLS ON TOUCHSCREEN DEVICES FOR  
BLIND AND VISUALLY IMPAIRED PEOPLE**

by

Mrim Alnfai

Submitted in partial fulfilment of the requirements  
for the degree of PhD

at

Dalhousie University  
Halifax, Nova Scotia  
June 2018

© Copyright by Mrim Alnfai, 2018

## TABLE OF CONTENTS

TABLE OF CONTENTS.....	ii
List OF TABLES .....	viii
LIST OF FIGURES.....	ix
ABSTRACT .....	xii
LIST OF ABBREVIATIONS USED.....	xiii
<b><u>CHAPTER 1: INTRODUCTION.....</u></b>	<b>1</b>
1.1 RESEARCH OBJECTIVE.....	4
1.2 RESEARCH QUESTIONS.....	5
1.3 OVERVIEW OF THE THESIS CONTRIBUTIONS .....	6
1.4 THESIS OUTLINE .....	7
<b><u>CHAPTER 2: BACKGROUND AND LITERATURE REVIEW .....</u></b>	<b>10</b>
2.1 BRAILLE PATTERNS.....	10
2.2 RELATED WORK .....	14
2.2.1 EXISTING CALCULATORS FOR BLIND PEOPLE .....	14
2.2.2 EXISTING TEXT INPUT METHODS FOR BLIND .....	16
2.2.3 EXISTING AUTHENTICATION METHODS FOR BLIND .....	25
<b><u>CHAPTER 3: FRAMEWORK FOR ASSISTIVE TECHNOLOGY .....</u></b>	<b>36</b>
3.1 OVERVIEW OF THE FRAMEWORK FOR THE PROPOSED ASSISTIVE TOOLS.....	36
3.2 FRAMEWORK OVERVIEW .....	37
3.3 FRAMEWORK COMPONENTS .....	38
<b><u>CHAPTER 4: TEXT INPUT METHODS.....</u></b>	<b>43</b>
4.1 THE SINGLE TAP BRAILLE TEXT ENTRY METHOD.....	43
4.1.1 TYPING AND EDITING ON A TOUCHSCREEN .....	43
4.1.1.1 GRADE 1 .....	44
4.1.1.2 GRADE 2 .....	45
4.1.2 ALGORITHM ADVANTAGES .....	48
4.1.3 TYPING AND EDITING OPERATIONS .....	49
4.2 SINGLE TAP BRAILLE IMPLEMENTATION.....	51

<b>4.2.1 EXPERIMENTAL TOOLS .....</b>	<b>51</b>
4.2.1.1 ANDROID OS .....	51
4.2.1.2 ANDROID STUDIO .....	52
4.2.1.3 MICROSOFT EXCEL .....	53
<b>4.2.2 EXPERIMENTAL SERVICES .....</b>	<b>53</b>
4.2.2.1 TALKBACK SERVICE .....	53
<b>4.2.3 EXPERIMENTAL MODEL .....</b>	<b>55</b>
4.2.4 SINGLETAPBRAILLE INTERFACE .....	69
<b>4.3 EVALUATION METHODOLOGY .....</b>	<b>69</b>
<b>4.3.1 RECRUITMENT OF PARTICIPANTS .....</b>	<b>70</b>
<b>4.3.2 METHODOLOGY .....</b>	<b>71</b>
<b>4.3.2.1 STUDY PROCEDURE .....</b>	<b>71</b>
<b>4.3.3 RESULTS AND DISCUSSION.....</b>	<b>73</b>
<b>4.3.3.1 RESULTS .....</b>	<b>73</b>
4.3.3.1.1 QUANTITATIVE RESULTS .....	73
1. <i>SPEED</i> .....	73
2. <i>ACCURACY</i> .....	74
<b>4.3.3.2 QUALITATIVE FEEDBACK .....</b>	<b>75</b>
4.3.3.2.1 <i>PRE-QUESTIONNAIRE</i> .....	75
4.3.3.2.2 <i>USABILITY QUESTIONNAIRE</i> .....	77
4.3.3.2.3 <i>INTERVIEW</i> .....	77
4.3.3.2.4 EVALUATION OF GRADE 2 ENCODING IN SINGLETAPBRAILLE.....	79
<b><u>4.3.3.3 DISCUSSION .....</u></b>	<b><u>81</u></b>
<b>4.3.3.3.1 SPEED AND ACCURACY .....</b>	<b>82</b>
<b>4.3.3.3.2 KEYBOARD FEATURES.....</b>	<b>83</b>
<b>4.4 BRAILLEENTER INPUT METHOD .....</b>	<b>84</b>
<b>4.4.1 THE FRAMEWORK OF BRAILLEENTER TEXT ENTRY METHOD .....</b>	<b>84</b>
<b>4.4.1.1 TYPING FUNCTION.....</b>	<b>85</b>
<b>4.4.1.2 BRAILLEENTER ALGORITHM .....</b>	<b>85</b>
<b>4.4.1.3 TYPING NUMBERS .....</b>	<b>87</b>
<b>4.4.1.4 TYPING PUNCTUATION .....</b>	<b>88</b>

4.4.1.5 EDITING FUNCTION .....	88
4.4.1.6 USING VOICE FEEDBACK.....	89
4.4.1.7 KEYBOARD ADVANTAGES.....	89
4.5 BRAILLEENTER KEYBOARD IMPLEMENTATION .....	89
4.5.1 EXPERIMENTAL MODEL .....	90
4.5.1.2 BRAILLEENTER KEYBOARD INTERFACE .....	96
4.6 PRELIMINARY EVALUATION.....	97
4.6.1 PARTICIPANTS.....	98
4.6.2 APPARATUS.....	99
4.6.2.1 SWIFT BRAILLE KEYBOARD .....	100
4.6.2.2 TEXT PHRASES.....	101
4.6.3 EXPERIMENTAL DESIGN.....	103
4.6.3.1 PROCEDURE .....	104
4.6.4 EVALUATION METRICS .....	106
4.6.5 RESULTS AND DISCUSSION.....	108
4.6.5.1 QUANTITATIVE RESULTS .....	109
4.6.5.1.1 SPEED.....	109
4.6.5.1.1.1 KEYBOARDS' SPEED ACROSS TEXT TYPES .....	109
4.6.5.2 ACCURACY.....	110
4.6.5.2.1 ERROR RATE .....	110
4.6.5.2.2 CORRECTED ERROR RATE .....	112
4.6.5.2.3 UNCORRECTED ERROR RATE .....	112
4.6.5.2.4 ERROR RATE ACROSS TEXT TYPES .....	113
4.6.5.2.5 LEWIS' RATING QUESTIONNAIRE RESULTS.....	114
4.6.5.2 QUALITATIVE RESULTS .....	116
4.6.5.2.1 PRE-QUESTIONNAIRE .....	116
4.6.5.2.2 INTERVIEW .....	116
4.6.5.2.2.1 BRAILLEENTER STRENGTHS AND LIMITATIONS.....	116
4.6.5.2.2.2 BRAILLEENTER SUGGESTIONS .....	118
4.6.5.2.2.3 SWIFT BRAILLE STRENGTHS AND LIMITATIONS.....	118
4.6.5.2.2.4 SWIFT BRAILLE SUGGESTIONS.....	119

4.6.5.2.2.5 OBSERVATION .....	120
4.6.5.2.2.6 PREFERENCE.....	121
4.6.6 DISCUSSION .....	122
4.6.6.1 SPEED AND ACCURACY .....	122
4.6.6.2 USABILITY AND PREFERENCE .....	124
4.6.6.3 COMPARATIVE ANALYSIS WITH OTHER INPUT METHODS FOR VISUALLY IMPAIRED .....	127
4.7 CONCLUSIONS .....	131
<b><u>CHAPTER 5: NUMB ERINPUT METHOD.....</u></b>	<b>133</b>
5.1 THE DESIGN OF THE BRAILLETAP CALCULATOR.....	133
5.1.1 REPRESENTING BRAILLE NUMBERS USING BRAILLETAP .....	136
5.2 BRAILLETAP CALCULATOR IMPLEMENTATION .....	137
5.2.1 USER INTERACTION MODULE.....	138
5.3 PILOT STUDY .....	145
5.4 CONCLUSION.....	147
<b><u>CHAPTER 6: BRAILLEPASSWORD METHOD.....</u></b>	<b>149</b>
6.1 RESEARCH CHALLENGES .....	149
6.2 BRAILLEPASSWORD AUTHENTICATION TECHNIQUE.....	150
6.2.1 DESIGN PRINCIPLES .....	151
6.2.2 THE BRAILLEPASSWORD DESIGN.....	151
6.2.3 BRAILLEPASSWORD INTERFACES .....	154
6.2.3.1 REGISTRATION INTERFACE .....	155
6.2.3.2 LOGIN INTERFACE .....	156
6.2.3.3 PASSWORD RECOVERY .....	157
6.2.4 INTERACTION TECHNIQUES.....	158
6.3 BRAILLEPASSWORD IMPLEMENTATION .....	159
6.3.1 BRAILLEPASSWORD AUTHENTICATION MODULE .....	159
6.4 BRAILLEPASSWORD ENTROPY .....	170
<b><u>6.5 USER STUDY .....</u></b>	<b>171</b>
6.5.1 APPARATUS.....	172
6.5.2 PARTICIPANTS.....	173

<b>6.5.3 EXPERIMENTAL DESIGN</b> .....	<b>174</b>
<b>6.5.3.1 STUDY PROCEDURE</b> .....	<b>175</b>
<b><u>6.5.4 RESULTS</u></b> .....	<b><u>178</u></b>
<b>6.5.4.1 QUANTITATIVE RESULTS</b> .....	<b>178</b>
<b>6.5.4.1.1 REGISTRATION TIME</b> .....	<b>178</b>
<b>6.5.4.1.2 LOGIN TIME</b> .....	<b>179</b>
<b>6.5.4.1.3 PASSWORD ENTRY SPEED</b> .....	<b>179</b>
<b>6.5.4.1.4 FAILURE RATE</b> .....	<b>180</b>
<b>6.5.4.1.5 PASSWORD ACCURACY AND MEMORABILITY</b> .....	<b>180</b>
<b>6.5.4.1.6 PASSWORD STRENGTH</b> .....	<b>181</b>
<b>6.5.4.1.7 GUESSING OF PASSWORDS FROM CAMERA VIDEOS</b> .....	<b>182</b>
<b>6.5.4.2 QUALITATIVE RESULT</b> .....	<b>183</b>
6.5.4.2.1 PRE-QUESTIONNAIRE.....	183
<b>6.5.4.2.2 INTERVIEW</b> .....	<b>183</b>
<b>6.5.5 DISCUSSION</b> .....	<b>191</b>
<b>6.5.6 STUDY LIMITATION</b> .....	<b>197</b>
<b>6.6 CONCLUSION AND FUTURE WORK</b> .....	<b>198</b>
<b><u>CHAPTER 7: DISCUSSION AND CONCLUSION</u></b> .....	<b><u>200</u></b>
7.1 DISCUSSION .....	200
7.2 CONCLUSION .....	202
7.3 FUTURE WORK .....	205
<b>APPENDICES</b> .....	<b>208</b>
APPENDIX A: BACKGROUND QUESTIONNAIRE.....	208
APPENDIX B: POST QUESTIONNAIRE OF SINGLETAPBRAILLE .....	210
APPENDIX C: INTERVIEW OF SINGLETAPBRAILLE.....	211
APPENDIX D: LETTERS OF APPROVAL FOR THE SINGLETAPBRAILLE ST...212	
APPENDIX E: BACKGROUND QUESTIONNAIRE OF BRAILLEENTER.....	215
APPENDIX F: LETTERS OF APPROVAL FOR THE BRAILLEENTER ST.....	216
APPENDIX G: POST QUESTIONNAIRE (LEWIS QUESTIONNAIRE) .....	219
APPENDIX H: INTERVIEW OF BRAILLEENTER KEYBOARD .....	220
APPENDIX I: BACKGROUND QUESTIONNAIRE OF BRAILLEPASSWORD ....	221

APPENDIX J: INTERVIEW OF BRAILLEPASSWORD .....	223
APPENDIX K: LETTERS OF APPROVAL FOR THE BRAILLEPASSWORD.....	224
<b><u>REFERENCES.....</u></b>	<b>208</b>

## LIST OF TABLES

TABLE 1. PREFIXES AND SUFFIXES FORMS IN GRADE 2.....	13
TABLE 2. CONTRACTION AND SHORTEN WORDS FORMS IN GRADE 2.....	13
TABLE 3. REMOVING THE VOWELS IN WORDS IN GRADE 2 BRAILLE .....	13
TABLE 4. BRAILLE KEYBOARD INTERACTION TECHNIQUES .....	50
TABLE 5: PARTICIPANTS’ DEMOGRAPHIC DATA.....	70
TABLE 6. BRAILLEENTER KEYBOARD INTERACTION TECHNIQUES.....	87
TABLE 7. EDITING INTERACTION METHODS IN BRAILLEENTER.....	88
TABLE 8: PARTICIPANTS’ DEMOGRAPHIC DATA .....	98
TABLE 9. TOUCHSCREEN KEYBOARDS DESIGNED FOR BLIND .....	99
TABLE 10: THE TWELVE TEST PHRASES USED .....	102
TABLE 11: AVERAGE TYPING SPEED OF BOTH KEYBOARDS .....	110
TABLE 12: RESULTS OF LEWIS’ RATING QUESTIONNAIRE .....	115
TABLE 13: RESULT SUMMARY .....	125
TABLE 14. AN OVERVIEW OF INPUT METHOD PERFORMANCE .....	128
TABLE 15. AN OVERVIEW OF METHODS’ QUALITATIVE FEATURES .....	129
TABLE 16. CATEGORIZATION OF BRAILLE NUMBERS .....	134
TABLE 17. BRAILLE KEYBOARD INTERACTION TECHNIQUES.....	137
TABLE 18: THE TIME SPENT TO USE THE BUTTON CALCULATOR.....	146
TABLE 19: EDITING INTERACTION TECHNIQUES IN BRAILLEPASSWORD .	158
TABLE 20: SWITCHING BETWEEN AUTHENTICATION INTERFACES .....	159
TABLE 21. PARTICIPANTS’ DEMOGRAPHIC DATA .....	174
TABLE 22. REGISTRATION INFORMATION FOR THE TRAINING PHASE .....	176
TABLE 23. STUDY SESSIONS’ TASKS FOR AUTHENTICATION METHODS ...	178



## LIST OF FIGURES

FIGURE 1: AN OVERVIEW OF THESIS CONTRIBUTIONS .....	7
FIGURE 2. THE BRAILLE CODE.....	10
FIGURE 3. THE ALPHABET PATTERNS IN BRAILLE .....	11
FIGURE 4. PATTERNS OF COMMON PUNCTUATION MARKS IN BRAILLE ....	11
FIGURE 5. PATTERNS OF NUMBERS IN BRAILLE .....	11
FIGURE 6. FRAMEWORK FOR ASSISTIVE TOOLS .....	37
FIGURE 7. SAMPLE OF CATEGORIZATION OF BRAILLE CHARACTERS .....	45
FIGURE 8. EXAMPLE OF WHOLE WORDS ABBREVIATIONS IN GRADE 2.....	46
FIGURE 9. SAMPLE OF TYPING SHORTEN WORDS FORMS IN GRADE 2.....	47
FIGURE 10. ALGORITHM FLOWCHART.....	47
FIGURE 11. TALKBACK SERVICE.....	54
FIGURE 12. THE DESCRIPTION OF GESTURES RECOGNITION ALGORITHM...60	
FIGURE 13. SNAPSHOT OF SWIPING GESTURE RECOGNITION .....	60
FIGURE 14. THE DESCRIPTION OF BRAILLE CHARACTERS ALGORITHM ....	63
FIGURE 15. SNAPSHOT OF BRAILLE CHARACTERS ALGORITHM.....	64
FIGURE 16. SNAPSHOT CODE OF GRADE 2 ABBREVIATION METHOD.....	65
FIGURE 17. SWITCHING BETWEEN THE KEYBOARD LAYOUT ALGORITHM.65	
FIGURE 18. SNAPSHOT OF SWITCHING BETWEEN THE KEYBOARD CODE...66	
FIGURE 19. ADD SPACE METHOD.....	66
FIGURE 20. SNAPSHOT OF ADD SPACE CODE.....	66
FIGURE 21. BACKSPACE METHOD.....	67
FIGURE 22. SNAPSHOT OF BACKSPACE METHOD CODE .....	67
FIGURE 23. TYPING METHOD .....	67
FIGURE 24. SNAPSHOT OF WRITE TEXT METHOD CODE.....	68
FIGURE 25. SPEAK METHOD .....	68
FIGURE 26. SNAPSHOT OF SPEAK METHOD CODE.....	68
FIGURE 27. SINGLETAPBRAILLE INTERFACE .....	69
FIGURE 28. ENTRY SPEEDS FOR SINGLETAPBRAILLE AND QWERTY.....	74
FIGURE 29. AVERAGE ERROR RATE FOR SINGLETAP AND QWERTY .....	75
FIGURE 30. TYPING THE LETTER “B” IN BRAILLEENTER .....	86

FIGURE 31. AN EXAMPLE OF TYPING A NUMBER IN BRAILLEENTER.....	87
FIGURE 32. TYPE SYMBOL “,” IN BRAILLEENTER .....	88
FIGURE 33. BRAILLE CHARACTERS RECOGNITION ALGORITHM.....	95
FIGURE 34. SNAPSHOT OF BRAILLE CHARACTERS RECOGNITION CODE.....	95
FIGURE 35. SNAPSHOT OF BRAILLE PUNCTUATIONS RECOGNITION .....	96
FIGURE 36. TIME HANDLER.....	96
FIGURE 37. BRAILLEENTER INTERFACE.....	97
FIGURE 38. SWIFT BRAILLE KEYBOARD.....	101
FIGURE 39: EXPERIMENTAL SETUP.....	104
FIGURE 40: TYPING SPEED FOR BOTH KEYBOARDS.....	109
FIGURE 41: AVERAGE TYPING SPEED OF BOTH KEYBOARDS.....	109
FIGURE 42: AVERAGE ERROR RATE OF BOTH KEYBOARDS.....	111
FIGURE 43: AVERAGE TOTAL ERROR RATE FOR BRAILLEENTER .....	111
FIGURE 44. AVERAGE CORRECTED ERROR RATE FOR BRAILLE ENTER.....	112
FIGURE 45. UNCORRECTED ERROR RATE FOR BRAILLE ENTER.....	113
FIGURE 46: KEYSTROKES PER CHARACTER RATE FOR BRAILLEENTER.....	113
FIGURE 47. TOTAL ERROR RATE BASED ON PHRASES TYPE.....	114
FIGURE 48: SUGGESTION FOR IMPROVEMENTS OF SWITCH BRAILLE.....	126
FIGURE 49. ALGORITHM FLOWCHART .....	136
FIGURE 50. GESTURE RECOGNITION ALGORITHM .....	140
FIGURE 51. CODE OF GESTURE RECOGNITION ALGORITHM .....	140
FIGURE 52. NUMBER RECOGNITION ALGORITHM .....	142
FIGURE 53: SNAPSHOT OF NUMBER RECOGNITION ALGORITHM CODE ....	142
FIGURE 54. CHANGE THE CALCULATOR OPERATIONS .....	143
FIGURE 55. SNAPSHOT OF SWITCHING BETWEEN CALCULATOR CODE .....	143
FIGURE 56. CLEAR METHOD .....	144
FIGURE 57. SNAPSHOT OF CLEAR METHOD CODE.....	144
FIGURE 58. BRAILLETAP USER INTERFACE .....	145
FIGURE 59: SAMPLE OF BRAILLE LOWERCASE LETTERS .....	152
FIGURE 60. EXAMPLE OF INSERTING CAPITAL A IN BRAILLEPASSWORD .	153
FIGURE 61. EXAMPLE OF TYPING IN THE BRAILLEPASSWORD .....	154

FIGURE 62: TYPING PUNCTUATION IN BRAILLEPASSWORD .....	154
FIGURE 63: REGISTRATION PAGES .....	156
FIGURE 64. PASSWORD RECOVERY PAGES .....	158
FIGURE 65. REGISTER STEPS IN THE BRAILLEPASSWORD.....	160
FIGURE 66. REGISTRATION AND LOGIN ALGORITHM .....	162
FIGURE 67. XML CODES OF REGISTER AND LOGIN INTERFACES .....	162
FIGURE 68. THE JAVA CODE OF THE SIGNUP FUNCTION .....	163
FIGURE 69. SNAPSHOTS OF LOGIN IN FUNCTION CODE .....	165
FIGURE 70. VIBRATION FUNCTION .....	165
FIGURE 71. XML CODE OF WELCOME INTERFACE .....	165
FIGURE 72. A SNAPSHOT OF REGISTER DATABASE .....	166
FIGURE 73. LOGIN PHP CODE.....	166
FIGURE 74. REGISTER PHP CODE .....	166
FIGURE 75. CONNECTING PHP FILES TO ANDROID APP CODE .....	167
FIGURE 76. CONNECTING PHP FILE TO MYSQL DATABASE .....	168
FIGURE 77. THE JAVA CODE OF VALIDATION FUNCTION .....	168
FIGURE 78. THE CODE OF BRAILLE CAPITAL SIGN .....	169
FIGURE 79. THE CODE OF BRAILLE NUMBER SIGN .....	169
FIGURE 80. MOVING BETWEEN FIELDS .....	170
FIGURE 81. THE TRADITIONAL AUTHENTICATION APPLICATION.....	173
FIGURE 82. REGISTRATION TIME FOR AUTHENTICATION METHODS.....	179

## **ABSTRACT**

Input methods on touchscreen devices are often developed without taking into account the needs of people with no or low vision. Because of this, there has been a strong research interest in the development of techniques for touchscreen accessibility for blind people, such as integrating the screen reader in smartphone devices. However, the QWERTY keyboard and the button calculator with VoiceOver and many other proposed touchscreen keyboards are in many ways inaccessible to blind and visually impaired people because they primarily require finding key locations on a touchscreen and both hands are needed to interact with keyboard interfaces. The primary objective of this research is to create, design and evaluate button-free user interfaces that allow blind users to overcome the limitations of button-based input methods, as well as mitigate privacy and security concerns. This research presents a model of designing accessible button-free input methods on smartphone devices that assist blind and visually impaired users. This thesis outlines the progressive development and evaluation of button-free user interfaces designed to make touchscreen devices on mobile phones more accessible for blind and visually impaired individuals. The first of these, SingleTapBraille, makes inputting text faster and more accurate than with the most common standard option, QWERTY keyboard with VoiceOver. Similarly, BrailleTap makes the entry of numbers and arithmetic symbols easier than with the standard smartphone calculator. We also introduce BrailleEnter, which further improves the functionality of SingleTapBraille and BrailleTap. Lastly, we address the security concerns associated with systems that increase accessibility for visually impaired individuals, specifically the password vulnerability associated with the auditory feedback provided by the VoiceOver service. We address this with BraillePassword, which provides input feedback to the user via haptic feedback that only the user can access. Four user studies are conducted with blind and visually impaired people. Data is generated through qualitative and quantitative methods. Findings indicate that the developed Braille tools significantly enhance accessibility, and has the potential to be valuable to blind users. The proposed model allows users to overcome outstanding accessibility challenges as well as mitigate privacy and security risks they face when interacting with smartphone devices.

## **LIST OF ABBREVIATIONS USED**

CNIB	Canadian National Institute for Blind
PIN	Personal Identification Number
IDE	Integrated Development Environment
ADT	Android Development Tools
APK	Android application package
OS	Operating system
XML	eXtensible Markup Language

## **CHAPTER 1: INTRODUCTION**

The touchscreen on smartphone devices allows users to interact naturally and manipulate data on the screen without other tools. Touchscreens also provide a high degree of flexibility that allows designers to create interfaces uniquely developed based on users' requirements and preferences. Another great advantage of smartphone devices is that it offers both computer and telephone features including camera, Internet access, calendar, notes and contact list. The advantages and capabilities of touchscreen smartphone devices have led many people with different abilities and needs to choose a touchscreen device over mobile phones with physical buttons. The visually impaired constitute a growing smartphone consumer group. A 2015 National Health interview survey reported that 23.7 million American adults age 18 and older are living with vision loss (American Foundation for the Blind, 2017). People with no or low vision, like the rest of the population, are increasingly using smartphone devices.

Despite all these advantages, touchscreens pose significant challenges with regard to input methods for blind people. Vision impairment makes it difficult for blind people to type characters via the QWERTY keyboard, which is the standard keyboard on touchscreen devices (Alnfiai, Sampalli, 2016; Azenkot et al., 2014). One of these difficulties is requiring users to locate a particular key on a touchscreen. This requires users to spend a long time memorizing the location of keys. Another limitation is the time and focus required for blind users to listen to all the letters while VoiceOver speaks them out as his/her finger passes over the screen. This situation becomes more challenging when a user wants to type text in a public space (Alnfiai, Sampalli, 2016).

In addition to being time consuming and error prone, inserting text with Talkback

poses privacy and security threats. When the TalkBack service reads aloud a character under the users' fingers while logging into an online account, it increases the threat of eavesdropping on users' private information including user names and passwords. In this case, the assistive technology (TalkBack) allows a person who is nearby to hear a blind user's password. If a blind person wears a headphone to prevent others from listening to their private information, the earplugs might prevent blind persons from knowing if there is someone nearby. This leads to a harmful security threat referred to as a shoulder attack. Another security issue arises when a blind user attempts to overcome the slowness of the input method by selecting short, easy to insert passwords (Azenkot et al., 2014). Some blind users rely on trusted sighted persons to help them access authentication mechanisms by giving them their private information (Sauer et al., 2010).

Indeed, input methods on touchscreen smartphone devices present major challenges for blind people. Most blind people want to use smartphone devices to complete a variety of tasks such as browsing the Internet, sending messages, emailing, accessing online banking, and storing contact information. Most of these tasks require users to interact with a touchscreen keyboard. Unfortunately, the most widespread input methods can exclude this population from the numerous advantages of performing tasks on touchscreen devices. Therefore, many researchers have developed various text entry methods for blind people (Alnfai, Sampalli, 2016; Azenkot et al., 2014; Mattheiss et al., 2015; Frey et al., 2011; Mascetti et al., 2011). However, many of these developed input methods require users to visually locate an object location or are designed mainly based on button interaction (Frey et al., 2011; AlBanna, 2016). Other input methods require users to use both hands to interact with keyboard interfaces because they are designed based on the

multi-touch approach (Frey et al., 2011; Mascetti et al., 2011). Hence, developing an accessible input method is still one of the biggest issues facing blind users of touchscreen devices. In short, there is a clear need for an accessible input method for blind people on touchscreen devices.

In response to these limitations, we proposed novel button-free input methods specifically designed for blind users that aim to overcome both usability and accessibility challenges. The core idea is to permit users to type on a touchscreen using Braille, which is the fundamental writing system used by visually impaired people. These input methods eliminate the need to visually locate buttons on a screen, and are mainly designed based on touch which is a sense that blind people can use very well. The new input methods significantly reduce the challenges that blind users face by eliminating annoying sounds, as well as the need to locate a specific location, requiring both hands to interact with the screen, and reducing security and privacy threats. The proposed input methods allow users to type letters, numbers, punctuation, lower case and capital letters, and includes an editing function.

The input methods rely on gesture patterns, as opposed to tap locations, allowing visually impaired users to type anywhere on a screen without taking into account the position of any interaction made on the input method interface.

The goal of this research is to support the following thesis statement:

Touchscreen input methods that use a gesture on a button-free user interface allow blind users to overcome the limitations of button-based input methods as well as reduce privacy and security concerns.



## **1.1 Research Objective**

The primary objective of this research is reducing the accessibility challenges that blind people face when interacting with touchscreen devices. One of these challenges is locating a specific key position on a touchscreen. Thus, our aim is to create button-free input methods to allow blind users to interact anywhere on a screen without needing to locate any object on a screen to insert numbers or text. Another objective of this research is to enable blind users to use only one hand (index finger) while interacting with a screen because it is the most accessible way for them, as explained in a previous study (Paisios, 2012).

This research also aims to overcome the limitations of the current input methods on touchscreen devices such as the QWERTY keyboard and the standard calculator. These input methods require users to visually locate a key on the screen, which is a time-consuming task for blind users. They move their fingers over a screen sequentially searching for the desired character, while moving their fingers over a screen, which might cause them to press the wrong key. This means these methods are prone to typing errors even if blind users use an assistive technology such as a TalkBack service to navigate through an input method. On the contrary, some of the serious difficulties that face blind users cause by assistive technology. The TalkBack service poses harmful privacy and security threats.

The primary objective of this research is to overcome the abovementioned limitations by designing input methods that meet blind users' abilities. It also aims to fully understand how blind users interact with the proposed methods to specify the design principles of the most accessible input interface for people with visual impairment. To do

so, an evaluation study was conducted to allow users to use both the proposed methods as well as current input methods. These steps helped us present an insightful guideline for developers who need to develop accessible interfaces on a touchscreen device for this population.

By designing the input methods and analyzing them, we will be able to tackle the most serious risk, which is password problems. Our goal is to carefully improve the proposed input method based on the evaluation findings. Then, we designed a security solution keeping in mind the disability of blind users as well as the limitations of both touchscreen and assistive technologies. We aimed to design a novel authentication method for web applications using the proposed input method as an entry pad after eliminating aural feedback. After that, we conducted a usability study for the proposed authentication method to determine how effective the new authentication method is in preventing “shoulder-surfing” attacks.

## **1.2 RESEARCH QUESTIONS**

1. Does a button-free interface based on Braille patterns with a single touch interaction technique improve the accessibility of number and text input method for people with visual impairment?
2. How do blind users experience a single touch interaction technique based on Braille patterns with voice feedback to enter numbers or texts? Do the proposed input methods allow users to input text faster and more accurately than the button-based keyboard e.g soft QWERTY keyboard?

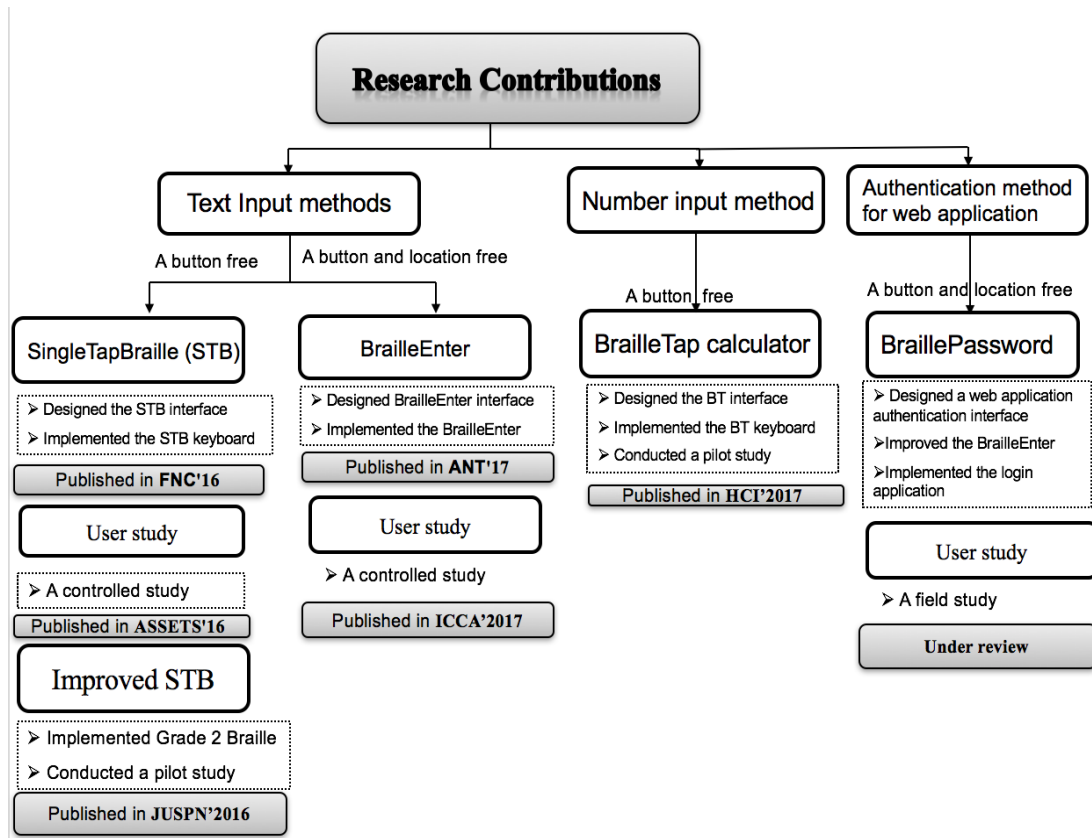
3. Does a button- and location- free input method on a touchscreen overcome the limitation of button-based input methods like QWERTY keyboard or Swift Braille keyboard?
4. How do people with visual impairment experience a single touch interaction technique with haptic feedback to enter their credentials?
5. Does the single touch interaction with vibration feedback on a button-free interface with non-visual text entry method reduce the risk of observation attacks (shoulder attack) for people with visual impairment? Can gestural interaction method with haptic feedback reduce observation attacks including audio and visual attacks?

### **1.3 Overview of the thesis contributions**

This research is motivated by the belief that it addresses the accessibility and security challenges that blind and visually impaired face using smartphone devices. Such challenges include, for example, typing text, calculating a simple equation, or securely logging into an online account. Visually impaired individuals struggle to do these basic activities due to the need to visually locate objects on a touchscreen layout.

To overcome these challenges, these individuals must invest additional time and energy into memorizing the input method layout. There is currently no application that overcomes the current input methods' limitations for individuals with limited or no vision. To fill this void, this research presents accessible input methods for touchscreen devices that utilizes tap and swipe gestures to type characters based on Braille patterns. This research also presents user studies with blind and visually impaired participants to fully understand the end users' requirements and figure out the feasibility of the button-free input

methods.



**Figure 1. An overview of thesis contributions**

Figure 1 shows the research components that have been completed and where have been published.

### 1.4 Thesis outline

The remainder of the dissertation is organized as follows. In **Chapter 2** we provide a review on the background of Braille. Then, we provide the related works and the past research that has been published on touchscreen input methods, calculators and authentication methods that have been developed for blind and visually impaired people.

**Chapter 3** introduces the framework of the assistive tools that proposed to improve the accessibility for visually impaired people. **Chapter 4** presents SingleTapBraille, a new single touch technique for text entry on touchscreen devices that allows users to type each character by tapping the screen with one finger. The system translates the user's taps into their corresponding character based on Braille character patterns. We evaluated SingleTapBraille with eight blind people at the Canadian National Institute for Blind (CNIB) by comparing the performance of SingleTapBraille to that of the standard QWERTY keyboard in combination with VoiceOver. This chapter also presents BrailleEnter, which was designed to allow users to type letters, numbers, and punctuation by pressing the screen to represent raised Braille dots and by briefly tapping to represent unraised dots anywhere using one finger, with input translated based on Braille coding. In an initial comparison of BrailleEnter and SingleTapBraille with two blind users, participants typed sets of phrases using both keyboards. BrailleEnter was more accurate than SingleTapBraille, and it has the potential to make smartphone devices fully accessible for blind people. We again utilized a comparative evaluation methodology to evaluate BrailleEnter's usability and performance, including speed and accuracy. Following this evaluation, we conducted a user study, which compared the usability and performance of BrailleEnter with that of Swift Braille, a button-based input method that represents Braille dots as six buttons on a touchscreen. The research presented in Chapter 4 has been published in four papers (Alnfai & Sampalli 2016a; Alnfai & Sampalli, 2016b; Alnfai & Sampalli 2017b; Alnfai & Sampalli 2017c).

**Chapter 5** discusses the design and initial evaluation of BrailleTap, a new number input method for touchscreen devices. To insert a number with BrailleTap, a blind user

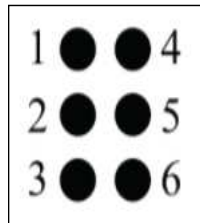
taps a number based on the Braille number codes. The system interprets the Braille code and returns its corresponding numerical value. To insert an operation, a user swipes vertically to select the desired arithmetic operations. In addition to enabling the entry of numbers and mathematical symbols, the system computes equation and reads the results using audio feedback. During an initial evaluation, participants typed sets of equations using both BrailleTap and the standard calculator available on touchscreen devices. Results indicate that BrailleTap is faster and more accessible for blind users than the standard calculator. The research presented in Chapter 5 has recently been published (Alnfiai & Sampalli, 2017a).

Following our previous evaluation study, we realized that there is a need to improve the web application touchscreen authentication process to better meet the needs of blind and visually impaired people. In **Chapter 6** we present BraillePassword, a new web application authentication technique for touchscreens that is accessible and protected against visual and oral threats. A user performs long taps and brief taps anywhere on a screen based on Braille character patterns, with no visual or audio feedback. Instead, it provides haptic feedback that facilitates error correction without the security risks associated with auditory or visual feedback. This chapter shows the results of our evaluation study to examine the performance of this new authentication method. **Chapter 7** includes the thesis' discussion and conclusion.

## CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

### 2.1 Braille patterns

Braille is the writing system that permits blind and visually impaired people to do both writing and reading tasks. Blind people use their fingertips to touch the raised dots of Braille characters. Each character in Braille is represented as a cell that contains six dots and are organized in two columns and are ordered from one to six, the first column contains dots one, two, and three, and the second column contains dots four, five, and six (American Foundation for the Blind, 2017), (as shown on Figure 2). Thus, there are 64 unique Braille patterns codes. Since blind readers' needs vary, three different Braille Grades were developed, which are Grade 1, 2 and 3. In this paper, we will only describe and implement Grade 1 and 2 because they are the most commonly used.



**Figure 2. The Braille code**

#### 2.1.1 Grade 1

In Grade 1, Characters consist of some combination of these six dots. Each Braille pattern in Grade 1 represents only one character. Grade 1 is used by novice blind users who are just starting to learn basic Braille. Grade 1 is used to print some books and magazines, but these documents are very large because Grade 1 requires a fair amount of space. Each character contains a combination of raised and unraised dots. Figure 3 illustrates the Braille

code that represents each letter, with small dots showing the unraised dots and the large black dots showing the raised dots in Braille code.

⠁	⠃	⠉	⠇	⠑	⠋	⠎	⠕	⠗	⠞
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				

**Figure 3. The Alphabet patterns in Braille**

⠂	⠆	⠒	⠒	⠒	⠒	⠒	⠒	⠒	⠒	⠒
,	;	:	.	!	()	?“	*	”	!	.

**Figure 4. Patterns of Common Punctuation Marks in Braille**

Punctuation marks in Braille are also represented by a combination of raised and unraised dots (American Foundation for the Blind, 2017). Figure 4 illustrates the Braille code that represents the punctuation marks, with small dots showing the unraised dots and the large black dots showing the raised dots in Braille code.

⠠	⠡	⠢	⠣	⠤	⠥	⠦	⠧	⠨	⠩	⠪
1	2	3	4	5	6	7	8	9	0	

**Figure 5. Patterns of numbers in Braille**

Braille numbers 0-9 are made using the same patterns as the first ten letters of the alphabet in Braille, “a” through “j” (Figure 5). Distinguishing between Braille letters and numbers is done by placing a prefix before the alphabet pattern “a” through “j”. A number prefix is an activation of dots 3, 4, 5, and 6 (American Foundation for the Blind, 2017).



Here, number 1 is represented by typing number sign plus Braille code for letter “a” and number 2 is represented by typing the number sign plus Braille code for the letter “b”. If users want to type larger numbers, they can type the number sign once. For example, typing 375 is represented by placing number sign plus Braille codes for “c” “g” “e” letters.

Decimal numbers are represented by using the comma (American Foundation for the Blind, 2017). The comma in Braille is the activation of dot 2 in numbers, for example, 375.53 is represented by placing number sign plus Braille codes for “c” “g” “e” “,” “e” “c”.

### **2.1.2 Grade 2**

Grade 2 Braille is used as an alternative to Grade 1 Braille to save space in large documents (Royal National Institute of Blind People, 2016), and is used by people who are experts in Braille. The main purpose of Grade 2 Braille is to save space in printed Braille documents because it allows for a shortened form of a word, for example letter “c” indicates the word “can”. It is also faster to read and write using Grade 2 Braille than Grade 1 Braille, thus most printed materials like books and magazines are written with Grade 2 Braille. There are many Braille code combinations that have been used to represent the shortened form of common words (2016).

1. Grade 2 represents a group of common letters in particular Braille patterns, for example, it includes common suffixes or prefixes in individual Braille signs (as shown in Table 1).

**Table 1. Prefixes and suffixes forms in Grade 2**

Words	code	Words	code	Words	code
ar	⠠⠗	ing	⠠⠊⠎	ow	⠠⠗
dis	⠠⠗	ed	⠠⠑	sh	⠠⠗

2. It represents entire commonly used words into single Braille character contractions (as shown in Table 2).

**Table 2. Contraction and shorten words forms in Grade 2**

Words	code	Words	code	Word	code
but	b	it	x	peopl	p
can	c	just	j	quite	q
do	d	knowledge	k	rather	r
every	e	like	l	for	⠠⠠
from	f	more	m	so	s
go	g	not	n	still	⠠⠠
have	h	this	⠠⠠	that	t
in	⠠⠠	out	⠠⠠	will	w
us	u	you	y	the	⠠⠠

3. It shortens words by removing most or all of the vowels in them (as shown in Table 3)

**Table 3. Removing the vowels in words in Grade 2 Braille**

Words	Contractions	Words	Contractions
about	ab	after	af
above	abv	afternoon	afn
according	ac	afterward	afw
across	acr	again	ag
always	alw	Braille	brl
blind	bl	deceive	dcv
either	ei	good	gd
friend	fr	great	grt

The abbreviations of words in Grade 2 Braille are not restricted to semantics, and can be used for nouns or verbs (Royal National Institute of Blind People, 2016). For example, in

Grade 2, the abbreviation of (can) is (c) and it can be typed as a verb or a noun. Sometimes, users should spell words to avoid confusion.

## **2.2 Related work**

### **2.2.1 Existing calculators for blind people**

Apple and Android have applied screen readers to smartphone devices to improve accessibility for blind people. Users can move their finger over the screen and the TalkBack service speaks the object name under the user's finger. Despite this improvement, inserting numbers or mathematics operations using the standard calculator with the TalkBack service is challenging (Alnfai, Sampalli, 2016 a; Alnfai, Sampalli, 2016 b; Ruamviboonsuk et al., 2012). The basic TalkBack calculator has one layer, with four operations on the right of side and the numbers 0-9 on the left side. The result of each calculation is displayed on the top of the screen. The user has to move his finger over the calculator screen and TalkBack reads out numbers or operations that are located under the user's finger. Thus, the user has to move his finger over the screen until he finds the desired number. If the user enters a wrong number, the user is required to find a clear button to delete the error. Thus, the main drawback of this method is that it is time consuming. This technique also consumes time when the user makes mistakes since she has to reenter the numbers and operations from the beginning. The main strength of the TalkBack calculator is that it provides audio feedback that helps users ensure they enter the correct number.

In addition to the standard keyboard with TalkBack, researchers have proposed several other text entry techniques (Hesselmann et al., 2011; Mattheiss et al., 2015; Mascetti et al., 2011; Frey et al., 2011; Oliveira et al., 2011; Azenkot et al., 2014), which

also predominantly require users to find a specific object location. Several developers have also introduced accessible text entry methods to overcome navigation problems, but they require users to remember particular gestures to enter text or numbers (Ruamviboonsuk et al., 2012; Shabnam et al., 2014).

Limited research has been done in the field of accessible number entry for blind users, and only one method has been integrated with an application for a particular aim, in this case an accessible calculator mainly for this population (Ruamviboonsuk et al., 2012). Previous work by Hesselman et al. (2011) developed a number entry method on a large screen, where a blind user taps on the screen with the number of fingers of the entry number. Thus, in this application, users need to use two hands to interact with a touchscreen. It also allows audio feedback to help users confirm they have entered the intended number. Drawbacks of this technique are that it is limited to entering numbers only from 1 to 10 and requires users to use two hands to interact with a screen. Another input method was proposed by Azenkot (2013), which is called DigiTaps. This method was designed mainly for small screens where a user only needs three fingers to interact with the screen, but this application requires several steps to just input one digit. After inserting a digit, the app provides haptic and optional audio feedback.

The H4-writer is a text and number entry method introduced by MacKenzie et al. (2011), which uses optimal prefix-free codes based on Huffman codes to insert numbers as well as text (Huffman, 1952). The H4-writer user interface has four buttons and the user need only use one thumb. Thus, it creates navigation issues and requires users to memorize the defined prefix codes. This is similar to Tapulator, a non-visual calculator, which was designed mainly for blind people. It uses prefix-free codes to enter numbers on a

touchscreen; users tap on touchscreens using one to three fingers and swipe the screen to insert a number. It requires users to perform specific gestures to perform operations, such as two consecutive swipes up to down to represent addition. The clearest advantages of this calculator are that it overcomes navigation problems by allowing users to tap anywhere on the screen and it provides voice feedback. Like BrailleTap, there are no numbers represented on the user interface; users can enter numbers on a screen based on Braille patterns. However, the major limitations of this method are that it is not easy to learn because users are required to memorize the defined prefix codes to enter a number or operation. It is also time consuming because users are required to complete more than three steps to enter a number.

### **2.2.2 Existing text input methods for blind people**

There have been several text input methods that have been developed or improved to meet blind people's needs. Identifying an optimal accessible input method is one of the challenges that face developers of touchscreen keyboards. Interaction with touchscreens is difficult for people with no vision due to several factors (Alnfiai, Sampalli, 2016). One of these factors is the absence of physical keys, so blind people cannot locate a key by touching or feeling. Another factor is that the communication of touchscreen devices is mainly based on visual feedback. The last factor is that the size of the touchscreen on smartphone devices is small and the keys are close to each other, thus the chance of tapping the wrong key is high. Thus, there have been many attempts to create accessible text input methods on touchscreens (Frey et al., 2011; Alnfiai, Sampalli, 2016; Mascetti et al., 2011; Jalaliniya et al., 2015; Oliveira et al., 2011). From 2005 until 2017, several text entry

methods on smartphone devices have been proposed in order to allow blind people to perform text entry more efficiently (Siqueira et al., 2016). Nevertheless, entering text on touchscreen devices is still a challenging task for blind people. Most of the developed text entry methods present accessibility and usability barriers for blind users. Most of these difficulties are caused by assistive technology limitations; keyboard interface limitations; and interaction technique limitations.

Some of these keyboards have suffered from assistive technology limitations, such as screen readers, sound recognition software and tactile QWERTY overlay. One of the biggest improvements is integrating screen readers in both iOS and Android smartphone devices. Blind users can therefore use the normal (QWERTY) keyboard with VoiceOver to be able to type text. Users can move their finger over the screen to find a particular character and perform a double tap to insert it. However, it is still inaccessible because of the time required for users to identify the location of a button by going through and listening to all the letters until they find the desired one. The voice reading out each character name as the user moves their finger over the screen can also be annoying.

Another significant improvement in accessibility for blind people is improving speech recognition services, such as Siri that allows users with no or low vision to insert text quickly and easily. Users can simply speak to their devices; however, blind people tend not to prefer these services because they find it awkward to speak their message aloud in a public place (Alnfiai, Sampalli, 2016). Also, most speech recognition applications are not adequately accurate and require the users to be in a quiet place to insert text (Nicolau et al., 2010).

Some companies provide a tactile overlay for the QWERTY keyboard on smartphone devices. The tactile QWERTY overlay is an added physical guide over the touchscreen that is recognized by the underlying application in order to provide tactile feedback for the users. Integrating tactile overlay with touchscreen can improve their accessibility to blind users and allow them to locate the keyboard keys easily as the overlay provides small bumps that can be touchable (Kane et al., 2015). However, this overlay provides limited functionality and requires extensive labor to customize an overlay for a particular application interface. It is also typically expensive, for example, the cost of the tactical overlay for the standard qwerty keyboard for an iPhone device is \$100 (Orf, 2015). To overcome this, some developers use plastics to create a tactile overlay that suit the requirements of individual users and applications. Even still, creating an overlay for a single interface is a laborious process.

Another contribution in the field of accessibility for blind people is the creation of refreshable Braille displays, which are electro-mechanical devices. They are used to display Braille characters and provide tactile feedback of the visual information that is displayed on a computer screen or smartphone device. The refreshable Braille device is connected to the computer or smartphone device via USB cable or via Bluetooth. The main usage of this device is that it produces Braille output on the Braille display using metal pins that raise and lower different combinations of pins in Braille cells to display the Braille characters. The key advantage of the refreshable Braille display is that it provides direct access to information that enables the users to check format, spacing, and spelling. Another advantage of this device is that it is quiet, as it provides only tactile feedback. However, these devices are extremely expensive. The price of Braille displays ranges from \$3,500 to

\$15,000, depending on the number of characters displayed. Braille displays vary in terms of how many cells are available, some of them provide 40, 70 or 80 characters. A 40-character display is sufficient for most jobs. Examples of jobs that could require a 70- or 80-character display are computer programmer and customer service representative. Another limitation associated with these devices is that it is bulky and users need to go through the installation stage.

Most of the available and proposed keyboards for blind people suffer from requiring users to locate an object on a touchscreen device such as keys on the QWERTY keyboard. There are also other keyboard interfaces such as BrailleType that are designed based on Braille, but they inherit the QWERTY keyboard limitations by requiring users to locate Braille dot positions on a screen (Oliveira et al., 2011). Another problem with the QWERTY keyboard interface is that the keyboard keys are located so close to one another due to the small size of a smartphone screen, which causes frequent errors (Nicolau et al., 2010).

Other proposed keyboards for blind people use different types of interaction methods with touchscreens, for example, using the multi-touch approach. Some of the interaction methods are not accessible for blind people and these methods require them to use both hands to interact with a screen, which is a difficult task because they usually carry something with the other hand (Alnfai, Sampalli, 2016; Paisios, 2012). Although often more accessible than the QWERTY keyboard, most of the available text entries for blind people suffer from one or more of these limitations.

Azenkot (2014) developed the Perkinput, a multi-touch text-entry method for touchscreen devices based on Braille coding. For large touchscreens, the Perkinput requires



the blind user to use two hands and three fingers from each hand to enter a character in one step. For small touchscreens, the user can use one hand to enter a character using three fingers in two steps. The first step inserts the dot 1, 2 and 3, and the second step 4, 5 and 6, as each finger represents a dot in the Braille pattern. The Perkinput interface eliminates the fixed button; thus, it is more accessible because it does not require the user to locate an object position on a screen. However, this keyboard does not completely overcome the navigation problem because a user has to calibrate their fingers over a screen registering his fingers' positions. This calibration is used to detect which fingers were used to touch the screen.

Mascetti, Bernareggi, and Belotti (2011) developed the TypeInBraille Keyboard, a touch and audio based text entry following Braille patterns. The keyboard interface has left and right sections. It requires at least three gestures (touches on the screen) to enter a character. Users insert Braille dots based on rows from the top to the bottom. Users touch the left or right section using a single finger to activate the Braille dot on the tapped section. Two taps on the screen using two fingers activates both dots in the right and left sections and three taps inactivate both left and right dots in a row. This keyboard supports voice feedback to enable users to confirm they entered the desired character. However, it requires users to memorize multiple gestures to insert Braille dots and perform other functions. These gestures are not easy to learn and it is difficult for blind people to perform because it requires users to insert dots in rows, unlike the typical way, which is in columns. Another weakness is that it requires users to use multiple fingers to make some of the gestures that require three fingers, which is somewhat difficult for blind people to perform (Paisios, 2012).

Another keyboard that follows the same interaction method approach is the GBraille keyboard (Maria et al., 2016). It uses both tap and swipe gestures to perform the keyboard functions and it requires at least three gestures to insert a letter. One tap on the left is used to insert a left dot and on the right to insert a right dot. Two taps using two fingers are used to insert both dots in a row. Swiping to the right inactivates both left and right dots in a row and swiping to the left removes the last character. It has the same strengths and limitations of the TypeInBraille keyboard.

The Eyedroid keyboard is similar to the TypeInBraille and GBraille keyboards (Jalaliniya et al., 2015). It is a swipe and audio based text entry, which follows Braille patterns. Users can swipe in different directions to activate and inactivate Braille dots to enter a character. The main advantage of this keyboard is that it does not require users to tap or swipe on a particular location and it supports audio feedback. However, it requires users to learn and memorize the defined gestures to insert Braille dots based on rows, which can be confusing for blind users because it does not follow conventional Braille coding and might be error prone.

Some of the keyboards suffer from both inaccessible user interfaces and inappropriate interaction methods. BrailleTouch (Frey et al., 2011) uses six keys to represent the Braille code and it uses a multi-touch interaction based on Braille shapes and it requires users to use two hands and three fingers from each to enter text on a screen. It also requires users to hold the smartphone device in landscape mode with the screen facing away from them, which is inconvenient and leads to privacy issues where others can see written text on a screen. This keyboard provides audio feedback of the inserted character. The main limitation of the keyboard is that it requires users to tap precisely on Braille dot

locations to insert a character. Apple's Braille screen input is similar to the MBraille and Braille Touch keyboards. It requires using two hands (multi-touch approach). The main advantage of this keyboard is that it allows users to type based on Braille, and it enables quick typing. However, It only works on landscape mode in small touchscreen devices. Another limitation is that users need to calibrate Braille dots each time they used the keyboard. A user stated that "I have found that I've had to recalibrate the dot positions a few times while practicing to get the hang of things" (AppleVis, 2014). Users need to place the first three fingers of first left hand, then right hand, on the screen. When users move their finger away from the previous calibration, they make typing errors. Another restriction addressed by the blind users is that they find the way of holding the device to be odd. Some of the users introduce the problem they face when they accidentally rotate their device and usually the dots are reversed (AppleVis, 2014).

Other existing keyboards require users to find the exact location of each object on a touchscreen. Most of these keyboards' interfaces are designed based on buttons. These keyboards are differentiated from each other based on the number of buttons presented on a screen, or the location of these buttons. Oliveira et al. (2011) developed a BrailleType keyboard that uses the location based approach. Its interface also uses six buttons that are organized in two columns representing Braille code. Audio feedback is used to read out the number of dots that have been touched or selected. A blind user must find the location of each Braille dot to activate it because the user interface is mainly designed based on buttons. Thus, finding the exact location of each dot is one of this keyboard's drawbacks.

Mattheiss et al. (2015) developed EdgeBraille, which is another input method that uses six buttons to represent the Braille code. The Braille dots are located on the corners

and edges of a screen. EdgeBraille allows users to swipe one finger along the screen's edges to draw a continuous line of Braille dots, selecting a combination of them in order to enter a character. Vibrio-tactile feedback notifies users about the activation and deactivation of Braille dots while the user slides his finger over the screen. Audio feedback has also been used to speak out the typed text. This keyboard is not accurate because it requires users to draw a line that connects the dots of each character. Blind users can not identify the exact positions of Braille dots which makes it harder to draw a line between them. Moreover, it only allows users to type letters, with no numbers or punctuation, and it does not allow users to edit their typing. Similarly, Li, Fan, and Truong (2017) proposed a gesture based text input method, which is called BrailleSketch. To input a character, a user draws a gesture of any size anywhere on the screen representing Braille code. Li et al. evaluated BrailleSketch without immediate letter-level audio feedback and showed that participants achieved 14.53 wpm with 10.6% total error rate. They also evaluated the BrailleSketch with immediate letter-level audio feedback and showed that typing speed was 8.37 wpm and the error rate was 11.3%. In addition, they found that typing some characters is not easy, such as N, V and Z where the user needs to draw the exact shape of the Braille character to differentiate it from other characters.

A very recent text entry method proposed for blind people is BrailleÉcran that uses a fixed layout and a tactile interface as a cover of a touchscreen (Siqueira et al., 2016). The BrailleÉcran interface represents Braille dots as six buttons and includes additional buttons for editing. Each button on the layout is represented by a tangible dot on the tactile interface that allows blind users to locate and press the exact location of a button. The main advantage of this keyboard is that it provides both vibration and audio feedback. It has not

been evaluated yet, but a possible disadvantage is that it requires users to purchase special equipment like a tactile cover. Another limitation is that some buttons have two functions, which might cause confusion. For example, when a user taps the buttons for volume, he/she obtains audio feedback, while when a user presses the buttons for volume, he/she closes the application. In addition, the buttons are close to each other, thus the chance of error is high. Similarly, Heni et al. designed MoonTouch, which is a text input method based on gestures (2016). It uses an enhanced version of the moon alphabet and it provides vocal and tactile feedbacks. The empirical evaluation result showed an average speed of 10.14 WPM and the error rate was not measured. Another keyboard that uses tactile interface is slate Master (Lee et al., 2017). It uses a tangible slate, stylus and an application uses a custom input interface, which mimics the use of the Braille Slate. The conducted a preliminary study with six Braille teachers to examine the design of the method and found that it can be a good application for learning how to write in Braille, but participants found it confusing where they do not know where they tap Braille dot as they are not able to receive tangible feeling as they do when they tap on a slate and a paper (old way) they can feel the embossed Braille dots on the paper. The speed and error rate were not measured.

BrailleKey was developed to overcome BrailleTouch's main limitation, which involves holding the mobile device upside-down (Subash et al., 2012). It uses four large keys, which are located on the corner of a screen. The two upper buttons are used for entering text based on the Braille code and the two lower buttons are used for editing. For the upper buttons, the one located on the left is used to insert the first column of Braille code (dots 1, 2 and 3); the one on the right is used to insert the second column of Braille code. For the upper left button, users can perform one touch to select the first dot, two

touches to select the second dot, and a long touch to select the third dot; this technique is applied for the second button as well. The main drawback of this keyboard is that it is designed based on buttons, thus it requires users to tap accurately on button locations.

Most of the existing Braille keyboards only support Grade 1 and not Grade 2, except the AccessBraille keyboard (Ludi, Timbrook, Chester, 2014), which uses a multi-touch approach and presents Braille dots as six columns on a touchscreen to allow blind users to use both hands to type on a small screen. This keyboard has the same drawbacks as the BrailleTouch keyboard (Frey, Southern, Romero, 2011).

Our main goal is to develop an accessible and usable number and text input methods for blind people, which allows interacting anywhere on a screen to enter a character. Following the same principle of our previous research and Paisios (2012), this app was designed to be used with one finger, as this is the most convenient and accessible way for a blind person to interact with a touchscreen. In addition, we allow users to interact with a touchscreen based on Braille.

### **2.2.3 Existing authentication methods for blind people**

The Internet and social media have become an essential part of most people's daily lives. People use the Internet for various purposes including socializing, sharing information, entertainment, education and shopping. The Internet presents its contents on websites for a desktop software application and on web applications for touchscreen platforms. Web applications require users to login to obtain access to online accounts such as emails, banking, healthcare or shopping applications. Logging into a web application by providing

a name and password is called an authentication process. Authentication is a process of ensuring the provided identification of a user matches the stored information in a database file of authorized users' information that is located in a local operating system or within an authentication server. It also can be defined as the procedure of verifying that only the authorized user can access to his/her accounts. The most common and widely used authentication methods are Personal Identification Numbers (PINs) and textual passwords.

Authentication with touch-based smartphones is a very challenging task for blind people due to usability, accessibility and security issues. Apple and Android have improved the accessibility of touchscreen devices by integrating screen reader service in both iOS and Android platforms and it works well with both the built-in apps and third-party apps (Accessibility, 2016; Android Accessibility, 2016). Using the PIN or textual passwords with the accessibility service are extremely difficult where users have to type the password sequences using the QWERTY keyboard. Thus, these authentication methods have posed security and usability issues for blind people (D'Arcy & Feng, 2006; Holman et al., 2008; Ma et al., 2012). Blind people used these methods with the support of TalkBack or VoiceOver services which pose serious security issues. For example, in these authentication techniques, a user moves his/her finger over the screen trying to locate a specific key on a screen. At the same time, the VoiceOver service reads aloud the name of the key under the user's finger. When the user locates the intended key, the user can double tap on the screen to insert it. In case someone is around the blind user, they can hear the user's private information including user name and password. Another security threat occurs when a user wears headphones to prevent people around him from listening to what has been typed on a screen. The earplugs reduce awareness of surroundings by preventing

a blind user from hearing or knowing if there is someone (specifically a shoulder surfer) around him/her. In this case, people might be able to see what keys the blind user pressed to insert the password (shoulder attack or video attack). Clearly, the PIN and textual passwords and VoiceOver are susceptible to observation, and thus security threats.

Furthermore, most existing authentication systems are inaccessible, for example, typing special characters increases time and difficulty for blind users, since they have to press a combination of keys to type such things as capital letters. This difficulty negatively impacts on the strength of password where blind users will tend to choose simple passwords, which are easier to crack and guess.

In short, current web authentication systems including PINs and textual passwords are not accessible for blind people because they require users to visually select the keys on a touchscreen. The limitations of PIN and textual passwords are due to the lack of accessibility and usability of touchscreen technology, and the inappropriate design of web application authentication interfaces and assistive technology. Thus, there is a need to consider usability and accessibility issues to overcome the security challenges that blind people encounter.

Numerous types of authentication methods for mobile phones have been proposed to make access to private information more secure. Fritsch, Fuglerud, and Solheim (2010) classified authentication methods into three types, which are knowledge-based, token-based and biometric authentication methods. Each of these authentication methods has its own pros and cons and none of them meet all users' needs and satisfaction. This is especially true for people who have disabilities. Graphical passwords are authentication methods that have been widely studied and many different ways have been proposed on



how to use this technique including choosing a sequence of images on a screen or images from a group for password, drawing a secret pattern or shape, and selecting a sequence of points in presented images (Sangore, Patil, Ramani, Pasare, 2014; Suo, Zhu, and Owen, 2005; Oorschot and Thorpe, 2008; Dhamija and Perrig, 2000). All of these graphical password techniques are inaccessible to individuals with no vision because it requires users to visually recognize and select number of images.

Alphanumeric passwords, which are defined as the traditional authentication method, such as PIN and textual passwords have been commonly studied on smartphone devices, but as mentioned, they are not accessible for blind people because it requires users to navigate and locate a specific character on the screen using TalkBack or VoiceOver service to help them select the intended character. Screen reader service broadcasts their private information while aiding them to navigate through the keyboard layout. As Cassidy et al. (2013) found, using the screen reader system to aid blind users while using the ATM interface reduces blind users' awareness of surrounding environment, meaning a blind user cannot hear if someone is nearby or behind him/her. This definitely increases the risk of observation attacks on this population. Similarly, using the Braille labels on the ATM interface to assist users to identify keys does not prevent the vulnerability to observation attacks where a bystander is able to see which key has been pressed (Helkala, 2012). Eiband et al. conducted a survey about shoulder attack, with 174 participants. The survey results indicate the existing of shoulder attacks in the real world and the results also show the serious need to improve password systems to reduce the risk of shoulder surfing.

Biometric authentication is an alternative method recently adopted on smartphone devices. Different types of biometric techniques including iris scans, hand, face, voice and

fingerprint recognition have been used for verification, eliminating the need to enter a password (Mudholkar, Shende, Sarode, 2012; Poh et al., 2016). However, biometric authentication methods might lead to unsolved security issues. The major security problem associated with the fingerprint service is that blind users might provide their fingerprints' scans to a system or a website when they are not able to see the system's or website's information (Fritsch, Fuglerud, Solheim, 2010). Blind users may not be able to use the iris scan because it mainly relies on visual cues and amputees cannot perform fingerprint scans (2010). Keane (2016) reported that facial biometrics for logging in are not an accessible method for blind people. Poh et al. (2016) conducted an experimental study and found that blind people struggle with taking selfies. Most of the face images were taken during the experiment were blurry or off-center due to the poor positioning of the camera. Voice authentication methods also suffer from distinct limitations in that they do not work when there is background noise or if the user has a throat infection. The voice authentication also leads to major privacy issues when people around user hear the user's private information (Fritsch, Fuglerud, Solheim, 2010).

There are other limitations that are associated with biometrics in general. Most people are not willing to provide their biometric data through a web application (Authentication technologies, 2009). Once the biometric information is hacked or stolen there is an insurmountable problem for the user. Passwords in other authentication methods can be reset or changed, but biometric information cannot be reset and this might lead to criminal activity (2009). Most available web applications do not rely completely on a biometric authentication method. Sometimes, these methods require users to insert alphanumeric passwords instead.

Therefore, in most cases, creating other authentication methods to meet blind people abilities is necessary. A lot of research discusses the importance of developing an accessible authentication method for blind people. Saxena and Watt (2009) highlighted that potential privacy and security threats remain largely unaddressed for blind people. They also addressed the serious need for an accessible authentication method for blind people. Similarly, Azenkot et al. (2012) also reported that most blind people are unaware of potential security threats based on a study with 13 blind participants. Furthermore, Dosono et al. (2015) conducted empirical studies exploring challenges and difficulties in authentication for users with no vision. Entering passwords correctly or verifying successful authentication are the major challenges that blind people face when logging in to a website.

However, there is little research that has proposed accessibility authentication method for people with no vision. Most of the proposed authentication methods for blind people are designed to thwart shoulder attacks for desktop software (Ali et al., 2016; Kuber, and Sharma, 2010; Wobbrock, 2009; Sae-Bae et al., 2012).

There is only one authentication method proposed on smartphone devices for blind people, which is called PassCords (Azenkot et al., 2012). It is a non-visual authentication method for touchscreen devices that uses a multi-touch approach where a user taps several times on a screen with one or more fingers. The algorithm used in this technique analyzes the number of finger contacts with the screen and determines which set of fingers touched the screen in order to define the password. At the beginning of each session, a user calibrates a set of reference points of the user's fingers anywhere on the screen. The reference points are used to identify the proximate position of the user's fingers on a screen. Azenkot et al. compared PassCords to the VoiceOverPINs in one study session for each

method with 16 blind people. The study findings were that the PassCords speed was faster than the VoiceOver PINs method and there was no significant difference in the failure rate of either method. The main causes of error in this method is that requiring users to calibrate without identifying the exact position of the user's fingers on a screen.

A very recent authentication method proposed for blind people is H4Plock (Ali, Kuber and Aviv, 2016) that uses a gestural and tactile mobile interface. Initially, the H4Plock requires users to select two passcodes including a primary and secondary passcode where a user can choose 1 to 4 gestures for each passcode. The H4Plock mobile interface is divided into four parts and the passcode can be inserted in one quadrant, some quadrants, and all the quadrants. A user will receive a vibration indicates which passcode should be inserted. This step will be repeated four times until the user inserts all gestures of a passcode. In each access attempt, the passcode is different due to the changes of the gestural sequence with the various vibration cues that are presented by algorithm. This makes observing or guessing the passcode sequence very difficult. The main advantage of this technique is that it provides vibration cues and allows users to use gestures instead of locating buttons. The evaluation findings with 17 sighted participants indicate that the H4Plock speed was 11 seconds to insert four-gesture passcodes and failure rate of this method was not calculated.

Kuber and Sharma (2010) designed a web-based tactile authentication method for desktop computers by displaying a grid of tactile stimuli through a tactile mouse. It allows blind users to authenticate access using their sense of touch. This method's interface has four grids of nine squares where a square has a unique pattern of raised pins presented via cells on top of the tactile mouse. A user chooses and memorizes four different pin patterns

from a set of thirty-six tactile stimuli. The chosen four patterns are used to form the user's password. An evaluation of this technique shows that most blind users are unfamiliar with using a mouse, Blind people are unfamiliar with how to navigate or perceive tactile feedback from a mouse. They depend heavily on using screen reader and keyboard keys. This indicates that using a mouse to navigate on the password interface is inaccessible for blind users. Therefore, in 2012, the authors improved this method by presenting tactile icons at one fixed point on the interface, each tactile icon is played for a short time. They used the VT Player device to display tactile feedback underneath the user's fingertips instead of tactile mouse. The main objective was to reduce the need to navigate using a mouse. The evaluation result shows that the accuracy rate was 81.8% and they spent 141.6s to authenticate access to the system and the navigation issue with the tactile icon still exists. The main limitation of this method is that users need training time to learn the tactile scanning process. Another limitation is that it was not implemented and tested on smartphone devices. It also requires extra equipment such as The VT Player device. Recently, Wolf, Kuber and Aviv (2017) described a preliminary study evaluating the accessibility and usability of using tactile feedback to help blind individuals enter their credentials using a PIN or 3\*3 pattern locks on a mobile authentication system. The study is still under progress but they found that tactile feedback is useful where it helps blind users to easily navigate the authentication interface. They also found that the tactile cues might improve privacy for blind users in some circumstances, in case they forgot their headphones or the phone is located in a pocket or bag and they want to interact with their phone privately. However, it requires blind users to carry extra equipment which is not preferable.

Wobbrock (2009) developed a rhythm-based authentication method, which is called TapSongs. TapSongs uses a single sensor such as button to enable users to tap a rhythm which is used as a password to verify the user. This method compares the inserted rhythm with a timing model already stored on a device. An evaluation of this technique shows that approximately 6-8 seconds is needed to authenticate a user and the strength of entropy was not evaluated. In a similar manner, Ali (2015) used Google Glass, which is a wearable device to help individuals with no vision to access online accounts on desktop computers where a user performs a sequence of directional gestures on a Google Glass's multi-touch pad and these inserted sequential gestures on Google glass will be stored and used to authenticate the user. The user study shows that TapSongs is not easy to crack, but the memorability of this method has not been measured. Similarly, Saulynas and Kuber conducted an exploratory study to evaluate the usability of using Brain-Computer Interface (BCI) and gestural headset to allow blind and visually impaired users to enter their passwords (2017). They found that using BCI and gestural technologies is slower and less accurate than entering four digit PINs even though it provides higher level of security against observation attacks.

In 2012, Sae-Bae et al. proposed an authentication method that used behavioral-biometric information gathered from multi-touch gestures of users' fingertips and their palms. It classifies movements and attributes of fingertips' and palms using a pattern recognition method. A user performs a multi-touch gesture using five fingers on a screen to create different gesture samples as passwords such as a pinch or zoom gesture. This technique will capture all x-y coordinates and time-stamps of five fingertips, then it compares the inserted multi-touch gesture with the registered patterns of the user in order

to verify the user. This method shows an improvement in the accuracy of interpreting multiple gestures.

Said, Kuber, and Murphy (2015) developed a sound-based user authentication interface, which is called AudioAuth. The AudioAuth Interface has 9 icons presented in 3x3 grids where the location of sounds on a grid is randomized. A user selects a sequence of sounds through a mouse. However, the main limitation of the AudioAuth method is that it is time consuming where users need quite a long period of time to scan over all the sounds in a grid and to distinguish the stored password sounds. The evaluation result shows that the accuracy rate was 80%. However, it has the same limitation as the tactile authentication, proposed by Kuber and Sharma (2010), where blind users are unfamiliar with using a mouse to select a sound icon.

In 2017, Balaji, Kuppusamy and Afzal proposed a screen locker based on Braille. The system interface presents the Braille dots as six large buttons located at the edges of the screen. Users select the Braille dots that form the desired character. It provides visual, oral and vibration feedbacks. This system has not been evaluated against security attacks. However, it seems this method has the same limitation as the QWERTY keyboard.

Several studies addressed the difficulties that blind people face logging in to an online account or while entering the credentials (Saxena and Watt, 2009; Azenkot et al., 2012; Kane et al., 2011). These include:

1. They were unable to find the location or box on a screen to type a username or a password. Blind users may be less precise in targeting specific areas of the screen.
2. They are unable to locate specific objects on a screen, for example, the link to “I forgot the password” or the help icon.

3. They are vulnerable to observation and audio attacks
4. They tend to ask for help from strangers or friends
5. The QWERTY keyboard digits are close to each other and that causes them to make errors.

Usability, accessibility, and security are closely linked and must be carefully considered to design a secure and usable user interface for authentication. In this work, our primary goal is to develop an accessible authentication method on touchscreen devices that overcomes most of the abovementioned difficulties for blind people. It will use gestures representing Braille code and allow users to type anywhere on a screen.



## **CHAPTER 3: FRAMEWORK FOR ASSISTIVE TECHNOLOGY**

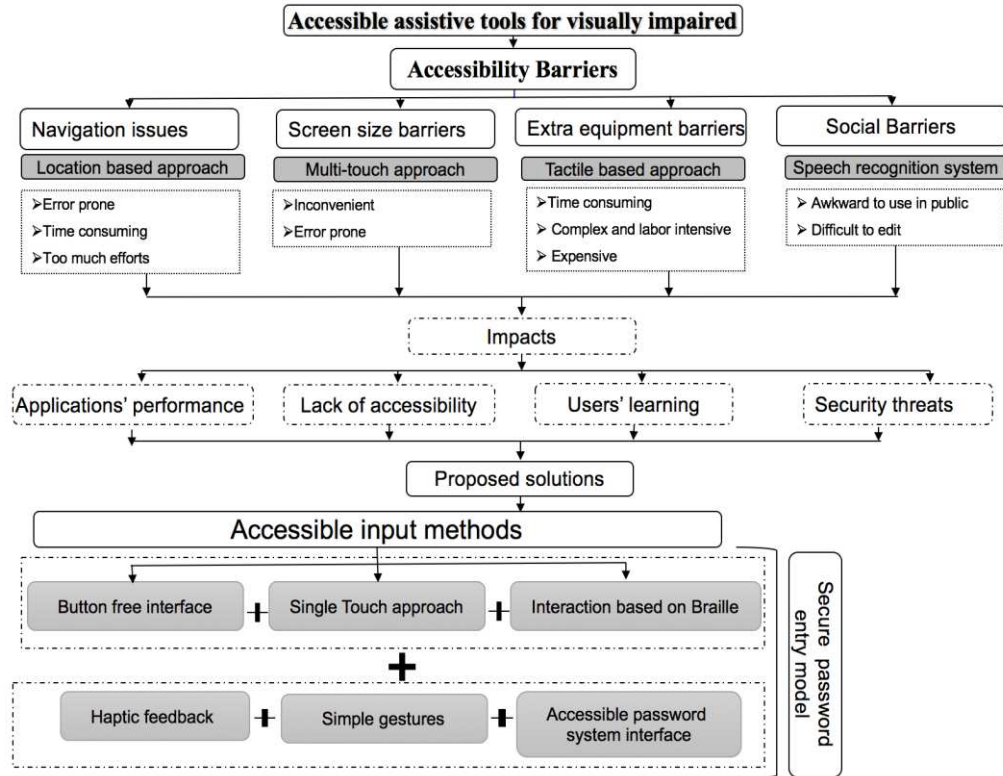
This chapter explains the framework for the proposed assistive tools to improve the accessibility of touchscreen devices for people with visual impairment.

### **3.1 Overview of the framework for the proposed assistive tools**

Recently, assistive software for visually impaired individuals has been developed for smartphones. This has increased accessibility and provided users with a rich unifying platform. For example, the integration of Text-to-Speech service into smartphone devices has enabled accessibility solution. However, visually impaired people still face some challenges and technical obstacles when using their smartphone devices. These obstacles include the touchable buttons, small screen size, and inconvenient interaction techniques. For these reasons, visually impaired people have difficulty finding keys easily on touchscreen devices. Additional assistive tools are needed to address both this issue and associated security risks.

In this research, we aim to remove these issues through the development of assistive applications. These applications enable users to type and calculate on touchscreens efficiently, with less effort and the ability to authenticate themselves securely. This research uses simple gestures on a button free user interface. This eliminates the limitations of button-based input methods and reduces privacy and security concerns. The goal of this research is to provide a conceptual framework that explain the research problems and the proposed solutions.

### 3.2 Framework overview



**Figure 6. Framework for assistive tools for visually impaired**

Figure 6 shows the framework that was utilized in this research in order to develop applications that allow blind and visually impaired users to overcome existing accessibility challenges. These include navigation, smartphone screen size, carrying extra equipment, and social barriers. These barriers result in reduced application performance, lack of accessibility, higher learning curve, and security threats. Recognizing these barriers and their associated impacts, this research developed a framework. These applications were designed in collaboration with the target population. This approach resulted in assistive tools that offer a more accessible and secure user experience.

### **3.3 Framework components**

There are many text input methods using different approaches that have been researched and developed for visually impaired people. Some of these input methods use a location based, multi-touch, or tactile based approach. Each approach has unique limitations that make typing on a touchscreen inconvenient and error prone for blind users. The existing location based keyboards require users to find a specific object location on a screen. Not only time consuming, it poses a navigation issue because the user has to move their finger over the keyboard layout trying to locate a particular key. Once the user has located the desired key, error is common because of the close proximity of keys on the keyboard layout. This issue particularly affects visually impaired people who need to move their fingers slowly over the keyboard layout with the support of the screen reader until they find the wanted letter. Then, users have to double tap to enter the desired character, but sometimes their fingers drift and hit the wrong key. As the keys are organized close to each other, it is very easy to accidentally hit the nearby key instead of the wanted one. Similarly, visually impaired people face the same difficulties while using the regular calculator to perform basic calculations in their daily life.

To compensate for the above problems, researchers have developed keyboards that use a multi-touch approach. Multi-touch keyboards require users to use two hands to type on the screen, but it is also inconvenient for blind users where one hand is usually occupied holding the smartphone and cannot be used for typing at the same time. In addition, the most accessible way for blind users is using one finger to interact with a touchscreen (Paisios, 2012). Another limitation associated with the multi-touch keyboards is that users

need to use three fingers of each hand to type Braille dots where the smartphone screen is small and does not suit interaction with two hands.

Other proposed keyboards require blind users to carry extra equipment like a tactile overlay to help them locate keys on the keyboard layout; this is not preferable for most blind users (Fard, Chuangjun, & Helgeson, 2011). The tactile overlay limits application functionality or users' abilities to interact with the screen or identify all the application buttons. Another reason blind people may not like this technique is that the tactile overlay for a particular touchscreen device requires extensive customization of the hardware. Thus, it is very expensive and complex to create an overlay as it requires intensive labour (He et al., 2017).

Due to the above-mentioned problems, more intuitive input methods of entering text and numbers is therefore essential in order to overcome the above obstacles that blind users have experienced.

A possible solution to the above problems is the design of novel entry methods based on Braille patterns that use a single touch approach to encode Braille patterns anywhere on a screen without any concern about location or buttons using only one finger. This input method has thus far been shown to be the most important tool for accessing applications on touchscreen devices. Therefore, we first developed a new text entry method called SingleTapBraille, a keyboard that eliminates the need to find a specific key location to input a letter, allowing information to be entered in a way that is fast and effective using a single thumb or finger based on Braille shapes. To improve the speed of the SingleTapBraille keyboard, we improved it by implementing grade 2 Braille, which employs shorten word forms. However, the SingleTapBraille utilizes algorithms based on

the relationships between dot positions on a screen. The user study revealed that blind users are not able to represent the Braille shapes correctly for some letters that have similarity in their Braille codes. Therefore, we proposed and developed BrailleEnter, another non-visual Braille input method that requires a user to utilize two gestures to encode Braille patterns of each character. This keyboard enables users to enter Braille dots anywhere on a screen using a single finger, and since the keyboard algorithm is not based on where a user taps, it eliminates the need for locating an object on the screen. Users can interact with the screen in one spot or any spot that is convenient for them that can allow people with vision loss to easily access different smartphone applications.

We evaluated the performance, strengths and weaknesses of the entry methods through user studies with seven to ten participants in each study. User studies' results indicate that typing anywhere using a single finger based on Braille patterns was preferred by blind users as it balances simplicity with accuracy. Insights from our user studies helped us further refine and improve the preferred entry method for visually impaired people.

The user studies also highlighted the need to create a calculator that meets blind people's requirements. We learned they faced the same problems when using the regular calculator that they have faced while using the QWERTY keyboards. Thus, we used the single touch approach based on Braille pattern to build an accessible calculator that overcomes navigation issues.

Location based input methods do not only pose efficiency and error challenges to visually impaired users; they also cause security threats by displaying or speaking out what keys the blind user pressed to enter the password. Thus, most web authentication methods that use these keyboards with VoiceOver are susceptible to observation attacks.

Furthermore, most existing authentication systems are inaccessible, for example, typing special characters increases time and difficulty for blind users, since they have to press a combination of keys to type such things as capital letters. This difficulty negatively impacts the strength of password where blind users will tend to choose simple passwords, which are easier to crack and guess.

Current web authentication systems including PINs and textual passwords are not accessible for blind people because they require users to visually select the keys on a touchscreen. The limitations of PIN and textual passwords are due to the lack of accessibility and usability of touchscreen technology, and the inappropriate design of web application authentication interfaces and assistive technology. Thus, there is a need to consider usability and accessibility issues to overcome the security challenges that blind people encounter.

Designing a web authentication method that protects against shoulder attacks is therefore essential in order to significantly reduce the security risk that blind and visually impaired people encounter while using smartphone devices. We therefore developed BraillePassword, a novel web authentication method on smartphone devices for blind users that cope with shoulder attack using the single touch approach and haptic feedback. BraillePassword proved to be more resilient to observation attacks. All the developed input methods are compared to commonly used input methods, and the developed web authentication method is compared to the PIN and textual password on touchscreens.

We designed entry methods that use a single touch approach based on Braille patterns where users are not restricted to locate an object on touchscreens. Our entry methods rely on the ability to type using their mother language (Braille) which does not

require time for them to learn. These methods, which employ a set of accessible gestures that can be easily performed by end users, lower the rate of errors that occur when users need to switch between layers or do editing. Additionally, entry methods which allow for exploration of the touchscreen that allows people with no vision to interact anywhere on the touchscreen without any concern about hitting undesired objects or opening up another screen enables them to learn the methods in a short time, and allows them to use the methods more easily. Ultimately, the interaction approach that uses a single touch approach based on Braille with haptic feedback provides an accessible and secure web authentication method that is resistant to observation attacks and allows for simple gestures and relatively effortless screen exploration. This research introduces a new interaction technique for input and a secure user interface for web authentication applications. This research allows future researchers to conceptualize the design of input and authentication methods in this manner.

## **CHAPTER 4: TEXT INPUT METHODS**

### **4.1 The SINGLETAPBRAILLE TEXT ENTRY METHOD<sup>1</sup>**

The proposed research seeks to overcome the limitations of existing input methods based on Braille. Our main goal is to develop a new text entry method that eliminates the requirement that users find specific locations in order to input letters, allowing information to be entered in a way that is fast and effective using a single thumb or finger. This is in accordance with Paisios (2012), who found that using one finger to interact with a touchscreen is the best, most accessible way for blind users. Similarly, a previous study on travelers conducted by Karlson, Bederson, Contreras-Vidal (1988) found that they usually hold bags with one hand and hold the mobile phone using the other hand. This situation is similar to the usual scenario for blind and visually impaired people who hold a cane or a guide dog with one hand and use the other hand to interact with a mobile phone.

The main objective of the present study is to identify the best features for users who are restricted to using only one hand to perform tasks on their mobile device. The design of our keyboard was guided by preliminary findings showing that the majority of users would prefer to use the thumb to interact with touchscreens. The description of our input method is detailed in the following section.

#### **4.1.1 Typing and editing on a touchscreen**

The SingleTapBraille system is based on Braille patterns, each with two columns and three rows. Our input method allows users to enter dots from left to right,

---

<sup>1</sup> Research Aptitude Work



corresponding to the way in which blind users read Braille. The first tap means a dot in the left column. Each Braille dot is activated individually using a single tap on a touchscreen. For example, if the user taps on any part of the screen once, this will represent the letter “a” and it means the user activated the first dot in Braille. When the user stops touching the screen for a short interval, the Braille pattern is interpreted and the resulting letter is typed and the software reads out the letter. With SingleTapBraille, users can hold the mobile phone with one hand because they need only one finger to enter Braille dots on the touchscreen. Users can also use their thumbs to enter Braille dots on the touchscreen.

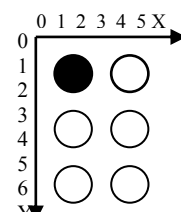
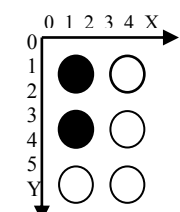
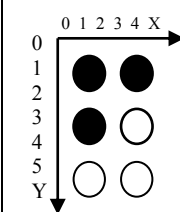
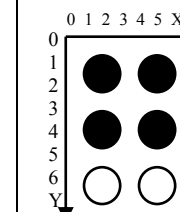
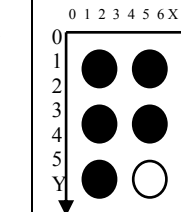
#### **4.1.1.1 Grade 1**

In our approach, the SingleTapBraille system analyze the relationship between active dots in each Braille character based on their coordinates and the distance between each dot and the following one(s). For example, if the user taps two taps below each other on a screen, this will represent the letter “b” and it means the user activated the first dot and second dot in Braille. The software will analyze the relationship between these two taps, and based on our algorithm, will identify the intended letter, number or punctuation mark and show it on the screen. The main concept behind our new input method is the clustering of activated Braille dots for each character. When the user stops touching the screen for a short interval, our algorithm clusters the entered taps and interprets them based on the number of dots. The algorithm processes each dot’s coordinates, as well as the space between each dot and the following one. Figure 7 illustrates how the algorithm works; the relationship between Braille dots is determined on the basis of the following values:

1. The value of coordinates (X, Y) for each dot.

2. The distance between dots,  $D$ , which means a defined value representing the distance between two dots.
3. The number of dots in each character

Here are some examples that explain how the proposed algorithm works:

Categorization of Braille characters based on number of dots				
Character has one dot	Characters have two dots	Characters have three dots	Characters have four dots	Character has five dots
Letter A, number 1	Letter b, number 2	Letter f, number 6	Letter g, number 7	Letter q
 <p>One tap anywhere in a mobile screen</p>	 <p><math> X1-X2  &lt; \text{error}; D \leq \text{a specific value}</math></p>	 <p><math> X1-X2  &lt; \text{error};  Y1-Y3  &lt; 0; D \leq \text{a specific value}</math></p>	 <p><math> X1-X2  &lt; \text{error};  X3-X4  &lt; \text{error}; X2 &lt; X3;  Y1-Y3  &lt; \text{error};  Y2-Y4  &lt; \text{error}; Y3 &lt; Y2, D \leq \text{a specific value}</math></p>	 <p><math> X1-X2  &lt; \text{error};  X3-X2  &lt; \text{error};  X4-X5  &lt; \text{error};  Y1-Y4  &lt; \text{error};  Y2-Y5  &lt; \text{error}; D &lt; \text{a specific value}</math></p>

**Figure 7. Sample of categorization of Braille characters based on number of dots**

The error parameter is a specific value that is used as a threshold for determining whether two floating point numbers are in a vertical or horizontal line. If the difference between  $x$  values of two taps is smaller than the error value, the algorithm will accept it because it's very likely that the user's taps are intended to be in the same vertical line.

#### 4.1.1.2 Grade 2

The main purpose for implementing Grade 2 Braille is to improve the keyboard speed. As indicated, when using Grade 2 Braille, contractions and abbreviations can be typed instead of whole words.

The implementation of Grade 2 follows the same approach of Grade 1 algorithm.

The algorithm will analyze the Braille sign of each contraction based on number of dots and the relationship between these dots in a single Braille sign.

Users select the Grade 2 Braille mode and they type the Braille sign of a word that they want to type. Our algorithm will check which word this Braille sign indicates and will spell out the complete word. At the same time, our application’s speak function will speak out the intended word. For example, if a user inserts the letter k Braille sign in Grade 2 mode, our keyboard will type out “knowledge” and it will speak out “knowledge” (see Figure 8).

<b>you</b>	<b>can</b>	<b>do</b>	<b>Knowledge</b>
<p>Letter y</p> <p>0 1 2 3 4 5 6 X 1 2 3 4 5 Y 8</p> <p><math> X1-X2  &lt; \text{error};  X3-X4  &lt; \text{error};</math>  <math> X4 X5  &lt; \text{error};  X3-X5  &lt; \text{error};</math>  <math>Y2-Y1 &gt; \text{gap}; Y5 &gt; Y1; Y5 &gt; Y3;</math>  <math>Y5 &gt; Y4;  Y5-Y2  &lt; \text{error}</math></p>	<p>Letter c</p> <p>0 1 2 3 4 5 6 X 1 2 3 4 5 Y 8</p> <p><math> Y1-Y2  &lt; \text{error}</math></p>	<p>Letter d</p> <p>0 1 2 3 4 5 6 X 1 2 3 4 5 6 Y</p> <p><math> Y1-Y2  &lt; \text{error};  X2-X3  &lt; \text{error}</math></p>	<p>Letter K</p> <p>0 1 2 3 4 5 6 X 1 2 3 4 5 6 -</p> <p><math> X1-X2  &lt; \text{error}; D &gt; \text{gap}</math></p>

**Figure 8. Example of entire words abbreviations in Grade 2**

However, the algorithm follows other steps to type words that their abbreviations have two letters or more (see Figure 9). Users tap the letters’ Braille codes and after inserting the abbreviation, our algorithm will find out the corresponding words and insert the complete words. At the same time, the speak function will read out the interpreted word. For example, a user types letter “a” and “b” and then adds space, the algorithm will check a corresponding word for the abbreviation “ab” and it will insert and speak “about”.

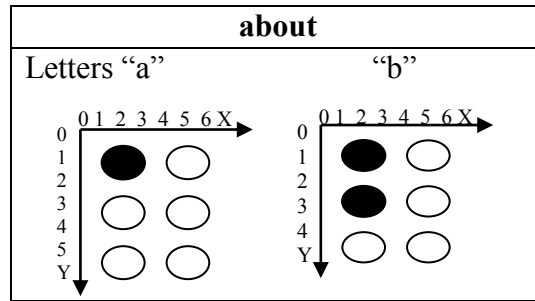


Figure 9. Sample of typing shortened words forms in Grade 2

Typing steps for Grade 2 abbreviations that contain two letters or more in the SingleTapBraille:

1. Typing the letters based on the Braille shape of each letter
2. Finding out the corresponding word
3. Replacing the abbreviation with the entire words
4. Reading out the inserted word

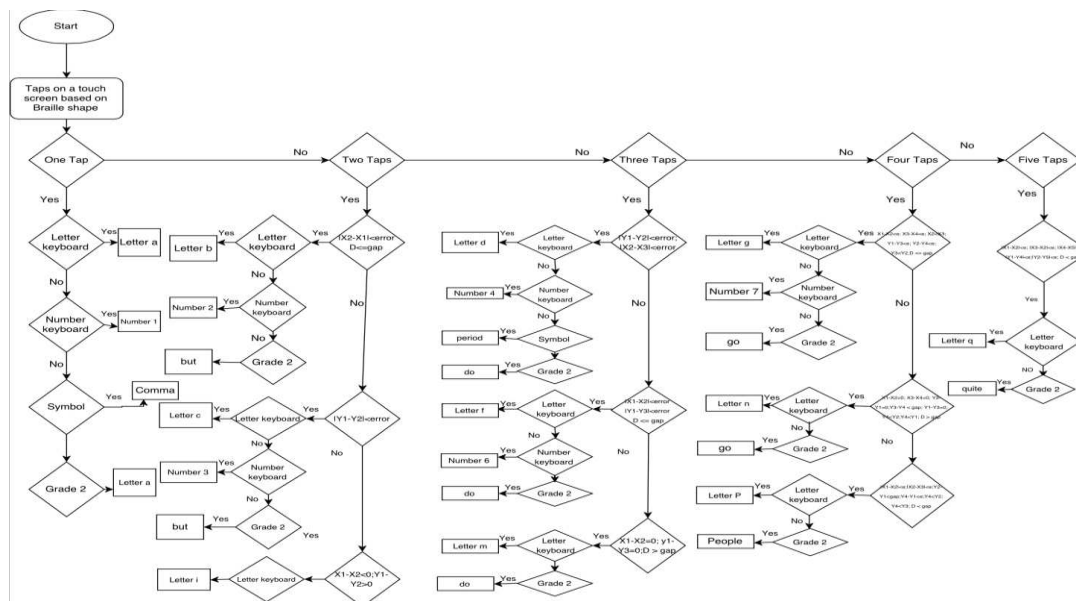


Figure 10. Algorithm flowchart

Figure 10 shows how the SingleTapBraille algorithm will check the number of taps on a screen and then it will analyze the relationship between each tap and the following one, and based on our algorithm, will identify the intended letter, number or punctuation mark and show it on the screen.

#### **4.1.2 Algorithm advantages**

The main advantage of our algorithm is that it does not restrict users by requiring them to find a specific object location on a screen, instead allowing them to tap the Braille dots anywhere on the screen based on Braille patterns. It does not have a button or other type of control that requires the user to precisely tap or click specific spots; it is gesture driven and has a corresponding voice output. Users do not have to tap on a specific location to activate a particular Braille dot. After the user taps using one finger (or thumb) to activate Braille dots, the software will analyze the relationship between those dots and represent the interpreted characters. The application uses audio feedback after a short interval, allowing the user to confirm that he/she entered the intended letter and understands the gestures being performed. As a result of letting users confirm what they are doing, the error rate will be significantly reduced. Furthermore, users can use one hand to hold the mobile phone and tap on the screen using only a thumb. Users can also operate our input method on a tablet using one hand to hold the tablet and another hand to enter text. Importantly, our application can be used both by stationary people and those on the move.

### 4.1.3 Typing and editing operations

There are some typing operations, such as space and backspace, that the Braille keyboard does not present using Braille dots, because they have specific buttons on the Braille machine that perform these functions. The SingleTapBraille method uses an assortment of flicking motions on the mobile screen in order to carry out certain functions, as detailed in Table 4.

The main advantage of this approach is that when the user performs a gesture, the software will speak out the gesture function's name. The SingleTapBraille method activates the Text-to-Speech function to speak out the registered interaction gestures that are used to facilitate typing, as shown in Table 4.

In order to change the keyboard mode, users can swipe vertically in a loop to activate a particular mode including lower case, upper case, numbers, symbols and Grade-2 modes. The first swipe will activate the first keyboard mode and TalkBack service will read out the keyboard mode. Then, if the users swipe vertically twice, they will obtain the second keyboard mode. For example, when the user performs the first vertically swipe from bottom to top, the app will speak "Upper case". In case the user does not want the upper-case mode, he/she can swipe again vertically to select the lower-case mode instead. Performing a vertical swipe from top to bottom will open the symbols mode. A second swipe from top to bottom will open the Grade 2 Braille mode. They can also swipe horizontally to edit their typing (see Table 4).

**Table 4. Braille keyboard interaction techniques and their purposes**

<b>Interaction techniques</b>	<b>purposes</b>
Swipe from bottom to top	Upper case, number mode
Swipe from top to bottom	Lower case, symbols mode, Grade 2 Braille
Swipe from left to right	Add space
Swipe from right to left	Backspace

In Braille, there are no special characters that represent capital or small letters (Pierce, 1995). In order to enter a capital letter using Braille, the user should activate dot number 6 in the pattern just before the letter that is meant to be capitalized (Paisios, 2012). None of the existing input methods have been able to help blind and visually impaired individuals type capital letters. In our method, users can simply swipe their finger from top to bottom to type capital letters. When the user swipes from top to bottom, the software will speak out the gesture function. The software also begins any sentence with a capital letter and then automatically shifts back to small letters. Swiping from the top down will allow the user to write a small letter.

As mentioned above, numbers follow the same order and have the same shapes as the first ten letters of the alphabet in Braille (Pierce, 1995). Consequently, the letter “a” sign represents number 1, letter “b” sign represents number 2, and letters “a-j-c” signs represents the number 193.

Using our input method, users can perform a defined gesture (a swipe from top to bottom) in order to enter numbers on a touchscreen.

Grade 2 Braille also follows the same Braille patterns order and shapes. A single Braille sign in Grade 2 indicates a word. Users can insert a Braille sign and the algorithm will replace it with the matched word.

The SingleTapBraille application interface enables the users to type based on Braille coding using a thumb or one finger. Visually impaired individuals can use the TalkBack accessibility service to listen to the entered letter after tapping on the screen. The SingleTapBraille interface has an edit text on the top of the screen, where the written text shows up. Users have the ability to tap anywhere under the edit text. The first time the user enters a given letter, it shows up as a capital letter automatically. User can switch between layers by swiping their fingers from top to down or vice versa. Users can also edit written text by swiping from right to left to remove the last typed letter.

## **4.2 SingleTapBraille implementation**

This section presents an overview of the major technologies being used within the SingleTapBraille keyboard. In the following sections, we describe the various experimental tools, the experimental services, and the experimental model for the developed application.

### **4.2.1 EXPERIMENTAL TOOLS**

In this section, we describe the experimental tools used in the SingleTapBraille application. The ways in which these technical tools were used is discussed in the ‘Experimental Model’ section.

#### **4.2.1.1 Android OS**

The Android smartphone operating system is open sourced and provides effective tools for developers to build high quality applications for all android device types. It allows users to edit, debug, build and deploy applications; thus the android operating system is considered



an appropriate environment for implementing our proposed applications for blind and visually impaired people. First, Android provides an informative document, tutorial and several websites share valuable knowledge about design and building such an application. Second, most available libraries for the android operating system are developed in the Java language. Third, most available android devices are offered to end users with a reasonable price thus there is many Android users. This means that any new application can quickly get published and downloaded by many users, resulting in abundant feedback (Vodička, 2011).

We implemented the SMS application in the Android platform because it is flexible by offering developers a low barrier to entry and allowing them to play with features and functionalities with their applications. The developed Braille applications can be implemented in any mobile platform, including iOS, Android, and Windows phones. The components of designing the user interface of this approach are available across all platform versions. In the implemented approach, we used only essential gestures to create a simple interface including short tap, long tap and swipe gestures that are easy to implement in other platforms. This approach does not require any element that is restricted to any specific platform release, and can be easily implemented across platform versions.

#### **4.2.1.2 Android Studio**

It is the formal Integrated Development Environment (IDE) for android operating system. It includes a workspace and a plug-in system for the development environment. It is a replacement for the Eclipse Android Development Tools (ADT) as the main IDE for native Android application development. The Java language is used to write numerous

functions involving Java editing, incremental compilation, code assist and an XML Editor (Android studio blog, 2017).

Android studio enables developers to design applications' interfaces and to build applications in the Android environment by offering the necessary API libraries like building, testing, and debugging tools. The distributed application extension is Android application package “.apk”. in addition, Android studio provides an emulator and source code for debugging sample applications to simplify testing build applications (Android studio blog, 2017).

#### **4.2.1.3 Microsoft Excel**

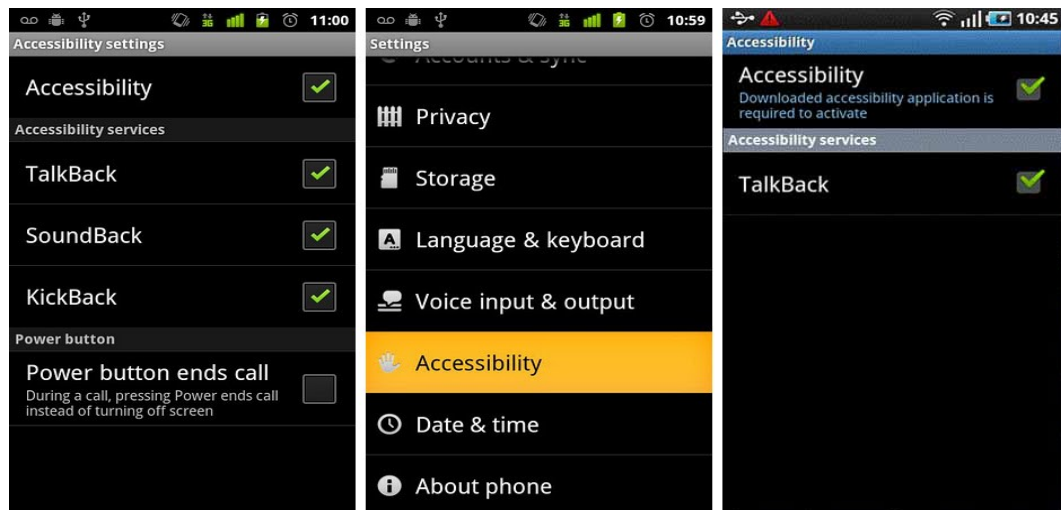
Microsoft Excel is a spreadsheet program used to process large amounts of collected data. The common usage of this program is to store and process commercial data using different functions, including basic mathematical operations (e.g., sums, averages, etc.). Microsoft Excel allows users to visualize the processed data in a graph, chart, diagrams in order to provide a brief knowledge about a trend of treated data. In this research, we used Excel to process the collected data from the conducted studies and provide an overview about the collected data.

### **4.2.2 Experimental services**

#### **4.2.2.1 TalkBack service**

Google's Android Accessibility Service provided TalkBack service, which is a screen reader pre-installed in the android devices. The main objective of this application is to provide oral feedback for the end users while moving their finger over the screen device. It also helps blind and visually impaired users by informing them of the result of any action

they make on the screen, such as (opening an application) and events (e.g. notifications) through verbal feedback. It also informs the users about which button they have touched and/or selected. In addition, it enables users to navigate through pages and menus on a screen by reading out the name of each object under their finger on the screen (Nuñal, 2012; Google, 2013). To enable TalkBack application, end users can go to *Settings* > *Accessibility* and enable the TalkBack service (see Figure 11). Google's Android Accessibility developers improve and update the TalkBack application regularly (Nuñal, 2012).



**Figure 11. TalkBack service**

Blind users can use TalkBack service to open up the proposed tools. Users can move their fingers over the screen. Each time they touch a button the TalkBack service will read out the name of the button. If the touched button is what the users were looking for, users can double tap on the button in order to activate the function of this button. If the button is not what users were looking for, they can move their fingers to the next button until they find the target button.

#### **4.2.2.2 TYPING AND EDITING USING THE SINGLETAPBRAILLE KEYBOARD**

We follow the “iOS Human Interface Guidelines”, while creating the application interface. The interface guidelines outline design principles for mobile devices for blind and visually impaired people. Based on these guidelines, the Text-to-Speech function is used to read out the content of the screen to end users (Russell-Minda, Jutai, & Strong, 2006; iOS Human Interface Guidelines, 2014; Designing for Screen Reader Compatibility, 2014).

#### **4.2.3 Experimental model**

In this section, we describe the implementation of the SingleTapBraille keyboard to enable blind users to type on a touchscreen using Braille as well as to overcome the navigation problems of the common used keyboard. We explain the SingleTapBraille interface, typing and editing functions. This section presents the SingleTapBraille architecture of the user interaction module, which includes:

1. Gesture pattern recognition module
2. Braille character pattern recognition module
3. Grade 2 abbreviation module
4. Keyboard modes or layout module
5. Editing module
6. Text to speech module

To enter a character on a touchscreen using the SingleTapBraille, one to six gestures are required to recognize a character. The SingleTapBraille keyboard algorithm contains five modules: the first detects the gesture pattern; the second recognizes Braille characters; the third is the Grade 2 abbreviation module; the fourth determines the keyboard mode; and

the fifth is an editing module, which includes gestures for special keys such as adding a space or backspace. The sixth element is the Text-to-Speech module used to speak out each input entered or removed from the screen.

#### **4.2.3.1 Gesture Recognition Algorithm**

The gestures of tapping, pressing, and swiping vertically/horizontally occur when a user places one finger on the smartphone screen until the time the user lifts the finger from the screen. The entire pattern of finger movements is interpreted as a particular gesture by the Android application. First, all the relevant data of touch events are gathered and then they are interpreted to see which gestures they match and whether the pertaining actions are executed as specified in the code.

We used *onTouchEvent*, which is with *MotionEvent* class to detect common gestures such as tap, press and swipe on a screen.

##### ***a. Using MotionEvent to Detect Gestures***

*MotionEvent* class is used to define an action movements and its axis values. To identify a gesture on a touchscreen, the *MotionEvent* class has been used to send the entered gestures to *Views* via the *onTouchEvent()* method. The *MotionEvent* class has all the information that are related to a touch or a gesture on the screen, for example, the number of pointers (ID), the (X, Y) coordinates and size and pressure of each touch. It identifies the action status such as a pointer going down or up or using the constants *Action\_up* and *Action\_down*. The main goal of these constants is to determine the action performed on a screen. Once the pointer touches the screen, each pointer has a unique ID that is assigned

when the finger goes down that identified by *Action\_down*. The ID of each pointer is valid until the pointer lifted off from the screen that identified by *Action\_up*.

#### **b. Using *onTouchEvent()* to Detect Gestures**

We used the *onTouchEvent()* method to capture the touch events on a screen. This method gets called each time the size, direction and position of the touch event changes or when the user places a new finger on the touch screen.

#### **c. *Overriding the Activity onTouchEvent()***

To detect the touch events, we override the *onTouchEvent()* method of the activity. One of the touch events class is *MotionEvent* class, which is used to capture and identify the action that being performed on a screen. This class has a method called *getAction()* method that used to determine the type of action and return an integer that represents the action code, such as *ACTION\_UP* or *ACTION\_DOWN*. Under this class, there are the *getX()* and *getY()* methods that are used to return the current X/Y coordinates of any interaction on the screen.

To detect the common gestures like tap, long tap, and swipe vertically and horizontally, we used the *GestureDectector* class. This class performs all the complicated procedures to handle the common gestures by itself. The input variables Xdown, Ydown, Xup, and Yup coordinates of each touch events happens on the smartphone screen. The Minimum swipe distance SWIPETHRESHOLD is set as 100 pixels. The variable Xdown is value of X coordinates when the finger touches the screen. Ydown is value of Y coordinates when the finger touches. Similarly, the variable Xup is the value of X coordinate when the finger goes up, Yup is the value of Y coordinate when the finger goes up.

*d. The logic of gesture pattern identification is that*

*Step 1* reads the input values of Xdown, Ydown coordinates values in an **ActionDown** event

*Step 2* reads the input values of Xup, Yup coordinates values in an **ActionUp** event.

*Step 3* checks if the subtracted value of Xup from Xdown is greater than SWIPETHRESHOLD; Xdown is the x coordinate when finger touches the screen and Xup is the x coordinate when the finger goes up of the screen. The outcome is swipe vertically to change the keyboard modes between letters, numbers and punctuations.

*Step 4* checks if the subtracted value of Yup from Ydown is greater than SWIPETHRESHOLD; Ydown is the y coordinate when finger touches the screen and Yup is the Y coordinate when the finger goes up of the screen. The action is swipe horizontally to do editing features in the keyboard.

*Step 5* check if the value of Xup is greater than Xdown; perform right swiping to add a space.

*Step 6* check if the value of Xup is smaller than Xdown; perform left swiping to do a backspace.

*Step-5 or Step-6* will be executed when Xup is greater than Xdown (Swiping Horizontally).

#### **4.2.3.1.1 Explanation of Gestures**

The gestures that have been used to implement the SingleTapBraille keyboard are:

1. Short Tap
2. Long Tap
3. Swiping horizontally

#### 4. Swiping vertically

The algorithm is implemented in Java language in an Android studio platform. The implemented algorithm is designed mainly based on the interaction techniques that users perform on the mobile phone screen.

Tapping gestures are used to form the Braille patterns of each character. Swiping gestures have been used to perform editing. The input values are one to six combination of tapping gestures that are taken from the smartphone screen. The algorithm produces expected outputs (letters, number or punctuations) based on the relationship between the position of the entered taps and their orders.

---

#### **/\* Gesture Recognition Algorithm \*/**

---

**Comment:** gestures used are swipe left, swipe right, swipe up and swipe down  
**Comment:** SWIPETHRESHOLD 100px. // Required minimum distance traveled to be considered swipe

**Step1. Read** input details from the touch screen including events where the touch position changed using MotionEvent

**Step2. Detect** gestures' type

On **ActionUp** event: read Xup, Yup coordinates values

On **ActionDown** event: read Xdown, Ydown coordinates values

**Step 3. If** ( $|Xup - Xdown| > SWIPETHRESHOLD$ ) **then**  
    **return** "Swipe Horizontally"

**Step 4. If** ( $Xup > Xdown$ ) **then**  
    **return** "Swipe right"  
    **do** "add space"  
    **end if**

**Step 5. If** ( $Xup < Xdown$ ) **then**  
    **return** "Swipe left"  
    **do** "Backspace"  
    **end if**  
**end if**

**Step 4. If** ( $|Yup - Ydown| > SWIPETHRESHOLD$ ) **then**  
    **return** "Swipe Vertically"

**Step 5. If** ( $Ydown < Yup$ ) **then**  
    **return** "Swipe down"  
    **do** "Change the keyboard mode"  
    **end if**

**Step 6. If** ( $Xdown > Yup$ ) **then**

---



---

```

    return "Swipe up"
    do "Change the keyboard mode"
    end if
end if
End Gesture Recognition Algorithm

```

---

**Figure 12. The description of gestures recognition algorithm**

```

ux = event.getX();
uy = event.getY();

deltaX = dx - ux;
deltaY = dy - uy;

//Horizontal Swipe Detection
if (Math.abs(deltaX) > min_distance) {
    swipe = true;
    if (deltaX < 0) { //Left to right
        addSpace(); //Add a space
        return false;
    }
    else if (deltaX > 0){ //Right to left
        backSpace(); //Delete a character
        return false;
    }
}
//Vertical Swipe Detection
else if (Math.abs(deltaY) > min_distance){
    swipe = true;
    if (deltaY < 0){ //Top to bottom
        if (mode == 0){
            mode = 4; //Wrap Around
        }
        else {
            mode--;
        }
    }
}
}

```

**Figure 13. Snapshot of swiping gesture recognition**

#### 4.2.3.2 Braille character pattern recognition module

The input variables are the touching events that users perform on the screen. This algorithm requires one to six inputs (taps) to produce one character. It processes these touches or gestures (input variables) by analyzing the number of touches being performed on the screen; the position of these touches (the shape or the relationship between taps by determining the coordinates of each tap); and the gestures' type, and orders. The active dots in each Braille character are represented by short tap gestures and the inactive dots are

ignored in this algorithm.

---

**/\* Braille Character Recognition Algorithm \*/**

---

**Comment:** Tapping gesture is used to present Braille pattern

**Comment: variables:** minimum distance 100px, error 50, gap 150

**Step1. Tap** on the screen representing Braille character

**Step2. Start** the time count once a user touches the screen

//Timer == 0 and the size of array list is zero (Taps' list [])

**Step3. Detect** the gesture type (*Gesture recognition algorithm*)

**Step4. Add** tap gestures on a screen, its coordinates, order into the array list (Taps' list []) and the number of tap gestures that have been performed on the screen in the specified time.

**Step5. DO step 1, 3 and 4 as many as the number of active dots in the Braille code of a character** (representing Braille dots in each character)

**Step6. Stop** the timer == 3 seconds

**Step7. If** (array list has more than six gestures)

**return** "more than six dots"

**speak** "couldn't recognize letter"

**end if**

**Step8. If** (array list has six gestures or less)

// 4 cases: one tap case, two taps case, three taps case, four taps case and five taps case

// 4 modes. mode 0: small letter, mode 1: capital letter, mode 2: numbers, mode 3: symbols

**Switch** (the number of taps stored in the array list)

**case 1:** // (one tap)

**if** (the keyboard is in uppercase mode)

**Enter** "Letter A"

**end if**

**else if** (the keyboard in lowercase mode)

**Enter** "Letter a"

**end if**

**else if** (the keyboard in number mode)

**Enter** "number 1"

**end if**

**break;**

**case 2:** (Two Taps)

**If** ((|X1-X2| < error) && (|Y1-Y2| <= gap))

**if** (the keyboard in uppercase mode)

**Enter** "Letter B"

**end if**

**else if** (the keyboard in lowercase mode)

**Enter** "Letter b"

**end if**

**else if** (the keyboard in number mode)

**Enter** "number 2"

**else if** (|Y1-Y2|<error)

**if** (the keyboard in uppercase mode)

---

---

```

    Enter "Letter C"
  else if (the keyboard in lowercase mode)
    Enter "Letter c"
  else if (the keyboard in number mode)
    Enter "number 3"
  else if ((X1-X2<0) &&(Y1-Y2<0))
    if (the keyboard in uppercase mode)
      Enter "Letter E"
    else if (the keyboard in lowercase mode)
      Enter "Letter e"
    else if (the keyboard in number mode)
      Enter "number 5"
  else if ((X1-X2) <0) && ((Y1-Y2)>0)
    if (the keyboard in uppercase mode)
      Enter "Letter I"
    else if (the keyboard in lowercase mode)
      Enter "Letter i"
    else if (the keyboard in number mode)
      Enter "number 9"
  else
    speak "Could not recognize letter"
  end if
  break;
case 3: (Three Taps)
  If ((|Y1-Y2| < error) && (|X2-X3| < error))
    if (the keyboard in uppercase mode)
      Enter "Letter D"
    else if (the keyboard in lowercase mode)
      Enter "Letter d"
    else if (the keyboard in number mode)
      Enter "number 4"
  else If ((|X1-X2| < error) && (|Y1-X3| < error)&&(|Y1-Y2|)< gap)
    if (the keyboard in uppercase mode)
      Enter "Letter F"
    else if (the keyboard in lowercase mode)
      Enter "Letter f"
    else if (the keyboard in number mode)
      Enter "number 6"
  else If ((|X1-X2| < error) && (|Y1-X3| < error)&&(|Y1-Y2|))
    if (the keyboard in uppercase mode)
      Enter "Letter H"
    else if (the keyboard in lowercase mode)
      Enter "Letter h"
    else if (the keyboard in number mode)
      Enter "number 8"

```

---

---

```

else If ((X1-X2)<0 && (|X2-X3| < error) && (|Y1-Y3| < error))
  if (the keyboard in uppercase mode)
    Enter "Letter J"
  else if (the keyboard in lowercase mode)
    Enter "Letter j"
  else if (the keyboard in number mode)
    Enter "number 0"
  end if
  break;
case 4: (Four Taps)
if ((|X1-X2|<error)&& (X3-X4<error)&&(Y1-Y3<error)&&(X1-X2|
<gap)&& (|Y1- Y2|<gap)&&(X2<X3) &&(Y3<Y2) && (|X3 X4|<gap)
&&(Y3-Y4<gap))
  if (the keyboard in uppercase mode)
    Enter "Letter G"
  else if (the keyboard in lowercase mode)
    Enter "Letter g"
  else if (the keyboard in number mode)
    Enter "number 7"
  end if
  break;
case 5: (Five Taps)
if ((|X1-X2|<error)&& (X2-X3<error)&&(X4-X5<error)&& (Y1-Y4< error)
&&(Y2-Y5<error))
  if (the keyboard is in uppercase mode)
    Enter "Letter Q"
  else if (the keyboard is in lowercase mode)
    Enter "Letter q"
  else if ((|X1-X2|<error)&& (|Y1-Y2|<gap)&&(X3-X4<error)&&(X4-
X5<error)
&&(Y1-Y3<error)&&(Y2-Y5<error))
  if (the keyboard is in uppercase mode)
    Enter "Letter Y"
  else if (the keyboard is in lowercase mode)
    Enter "Letter y"
  end if
  break;
default:
  speak("Couldn't recognize letter");
  break;
end
Step9. Clear the array list
Step10. Repeat step 1 to 9
Step 11. End Character Recognition Algorithm

```

---

Figure 14. The description of Braille characters' recognition algorithm

```

//check the size of list
switch(list.size()){
  case 1:
    if (mode == 2){
      writeText('1');
    }
    else{
      writeText('a');
    }
    break;
  case 2:
    raw_dx = list.get(0).x - list.get(1).x;
    raw_dy = list.get(0).y - list.get(1).y;

    xdif = Math.abs(list.get(0).x - list.get(1).x);
    ydif = Math.abs(list.get(0).y - list.get(1).y);
    //check b
    if ((xdif < error)&&(ydif <= gap)){
      if (mode == 2){
        writeText('2');
      }
      else {
        writeText('b');
      }
    }
    //check k
    else if((xdif < error)&&(ydif > gap)){
      writeText('k');
    }
    //check c
    else if (ydif < error){
      if (mode == 2){
        writeText('3');
      }
      else {
        writeText('c');
      }
    }
}

```

**Figure 15. Snapshot of Braille characters' recognition algorithm**

#### 4.2.3.3 Grade 2 abbreviation module

The replace () method is used to change the abbreviation of a word with the entire words letters. For example, if the user enters “ab”, then the replace method changes it to “about”.

```

d = x.substring(x.lastIndexOf(' ')+1);
if (d.length() == 2) {
  ch = x.substring(Math.max(x.lastIndexOf(' '), x.length() - 2));
  if (ch.equals("ab") || ch.equals("Ab")) {
    txtText.setText(x.replace(ch, "about") + chx);
    Speak("about");
  } else if (ch.equals("af") || ch.equals("Af")) {
    txtText.setText(x.replace(ch, "after") + chx);
    Speak("after");
  } else if (ch.equals("ac") || ch.equals("Ac")) {
    txtText.setText(x.replace(ch, "according") + chx);
    Speak("according");
  } else if (ch.equals("ag") || ch.equals("Ag")) {
    txtText.setText(x.replace(ch, "again") + chx);
    Speak("again");
  } else if (ch.equals("al") || ch.equals("Al")) {
    txtText.setText(x.replace(ch, "also") + chx);
    Speak("also");
  } else if (ch.equals("bl") || ch.equals("Bl")) {
    txtText.setText(x.replace(ch, "blind") + chx);
    Speak("blind");
  } else if (ch.equals("cd") || ch.equals("Cd")) {
    txtText.setText(x.replace(ch, "could") + chx);
    Speak("could");
  } else if (ch.equals("ei") || ch.equals("Ei")) {
    txtText.setText(x.replace(ch, "either") + chx);
    Speak("either");
  }
}
else if (d.length() == 3) {
  ch = x.substring(Math.max(x.lastIndexOf(' '), x.length() - 3));
  {
    if (ch.equals("acr") || ch.equals("Acr")) {
      txtText.setText(x.replace(ch, "across") + chx);
      Speak("across");
    }
  }
}

```

Figure 16. Snapshot code of Grade 2 abbreviation method

#### 4.2.3.4 Switching between the keyboard layout

---

##### Change the keyboard mode (Lowercase, uppercase, numbers and punctuations)

---

**Step 1. Check if the user performs swiped Vertically on the screen**

**Step 2. If (YDown < Yup) is valid then**

If the keyboard mode is typing symbols mode (m==3)

Change it to lower case mode (m == 0)

else

Increase the keyboard mode value to wrap around the layouts (m++)

Switch (keyboard mode)

Case 0: lower case mode (m==0)

Break;

Case 1: Uppercase mode (m==1)

Break;

Case 2: Numbers (m==2)

Break;

Case 3: Symbols (m==3)

Break;

---

Figure 17. Switching between the keyboard layout algorithm

```

//Vertical Swipe Detection
else if (Math.abs(deltaY) > min_distance){
    swipe = true;
    if (deltaY < 0){//Top to bottom
        if (mode == 0){
            mode = 4; //Wrap Around
        }
        else {
            mode--;
        }
    }
    else if (deltaY > 0){//Bottom to top
        if (mode == 4){
            mode = 0; //Wrap Around
        }
        else {
            mode++;
        }
    }
    switch (mode){
        case 0:
            Speak("Small Letters");
            break;
        case 1:
            Speak("Capital Letters");
            break;
        case 2:
            Speak("Numbers");
            break;
        case 3:
            Speak("Symbols");
            break;
        case 4:
            Speak("Grade 2 braille");
            break;
    }
}

```

Figure 18. Snapshot of switching between the keyboard layout code

#### 4.2.3.4 Editing module

This module has typing, add space and backspace functions.

##### 1. Add space

---

###### Add space method

---

Void addspace ()

X is the entered text on the screen

Return X + ' '; // Add an empty space ' ' to the entered text X

Speak "space";

End addspace method

---

Figure 19. Add space method

```

private void addSpace() {
    Speak("Space");
    final char chx = ' ';
    final String x = txtText.getText().toString();
    txtText.setText(x + chx);
}

```

Figure 20. Snapshot of add space code

## 2. Backspace method

---

### Backspace method

---

Void **Backspace ()**

**X** is the entered text on the screen

**If** (the length of the entered text **X** is 0)

**return X**

**end if**

**else if** (the length of the written text is longer than 0)

**remove** the last character (the entered text **X** -1)

**Speak** the removed character

**return** the written text **X** without the last character

**end if**

**end Backspace method**

---

**Figure 21. Backspace method**

```
private void backSpace(){
    Speak("Backspace");
    final String x = txtText.getText().toString();
    if (x.length() == 0) return;
    txtText.setText(x.substring(0, x.length()-1));

    txtText.post(new Runnable() {
        @Override
        public void run() {
            txtText.setSelection((txtText.getText()).length());
        }
    });
    if (txtText.getText().toString().length() == 0){
        mode = 1; //Set to capital letters
        empty = true;
    }
}
```

**Figure 22. Snapshot of Backspace method code**

## 3. Write function

---

### Typing method

---

Void **WriteText (Character ch)**

**Add** **x + ch**; // **ch** is the new entered character

**Speak** the new entered character

**Return** the new string text **X**

**End WriteText**

---

**Figure 23. Typing method**



```

private void writeText(final char ch){
    Speak(String.valueOf(ch));
    final char chx;
    if (mode == 1){
        chx = Character.toUpperCase(ch);
    }
    else{
        chx = ch;
    }
    final String x= txtText.getText().toString();

    txtText.setText(x + chx);

    txtText.post(run() -> { txtText.setSelection((x + chx).length()); });
    if (empty) {
        empty = false;
        mode = 0;
    }
}

```

**Figure 24. Snapshot of write Text method code**

#### 4.2.3.6 Text to speech module

The Text-to-Speech function is implemented in this keyboard to provide an audio feedback for the final user.

---

#### **Speak method ()**

---

##### **Void Speak (the entered text)**

```

    if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.LOLLIPOP)
        t2s.speak(text, TextToSpeech.QUEUE_FLUSH, null, null);
    end if
else
    t2s.speak(text, TextToSpeech.QUEUE_FLUSH, null);
end else
end Speak

```

---

**Figure 25. Speak method**

```

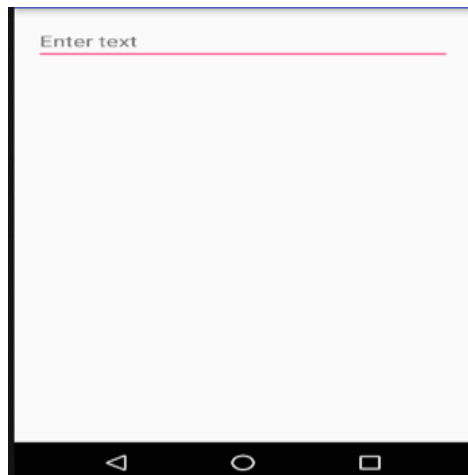
private void Speak(String text){
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        t2s.speak(text, TextToSpeech.QUEUE_FLUSH, null, null);
    }
    else {
        t2s.speak(text, TextToSpeech.QUEUE_FLUSH, null);
    }
}

```

**Figure 26. Snapshot of speak method code**

#### 4.2.4 SingleTapBraille interface

The application will be installed in the users' Android devices. As shown in Figure 25, the SingleTapBraille keyboard has no visual content. Except for the edit text box at the top of the screen, the remainder of the screen is empty to allow users to tap and move their fingers over the screen without concern for hitting any object on the screen.



**Figure 27. SingleTapBraille interface**

### 4.3 EVALUATION METHODOLOGY

This section describes the evaluation of the SingleTapBraille, which relied on having blind and visually impaired participants use our proposed keyboard and the QWERTY keyboard to perform the same tasks. Afterwards, we were able to compare the performance of the keyboards based on quantitative measures and the feedback of the participants. We also examined the accessibility, learnability and usability of the SingleTapBraille keyboard.

### 4.3.1 RECRUITMENT OF PARTICIPANTS

Evaluation was conducted with blind users. We had seven blind users (2 males and 5 females) ranging in age from 28-52 years old (mean age: 40). Recruitment of participants was undertaken with the assistance of the director of the Canadian National Institute for Blind. An approval from the Dalhousie University Social Sciences and Humanities Research Ethics Board was acquired before commencing the study (see Appendix D).

All participants had prior experience with touchscreens, the QWERTY keyboard, and TalkBack service. All of them were regular Braille users and were familiar with the six dots Braille system. The participants' demographic data is presented in Table 5.

**Table 5: Participants' demographic data**

<b>P</b>	<b>Age</b>	<b>G</b>	<b>Sight</b>	<b>PIB</b>	<b>BDU</b>	<b>ESP</b>	<b>MTT</b>
<b>P1</b>	48	M	blind	expert	Perkins Braille, slate and stylus	6 years	A QWERTY keyboard
<b>P2</b>	34	F	blind	expert		7 years	AQWERTY keyboard, MBraille
<b>P3</b>	40	F	blind	expert		5 years	A QWERTY keyboard
<b>P4</b>	50	M	blind	expert		5 years	A QWERTY keyboard
<b>P5</b>	46	F	blind	expert		3 years	A QWERTY keyboard
<b>P6</b>	36	F	blind	expert		3 years	A QWERTY keyboard
<b>P7</b>	29	F	blind	expert		5 years	A QWERTY keyboard

\*P = Participant number, G = Gender, PIB = Proficiency in Braille, BDU = Braille device they use, ESP = Experience with smartphones and touchscreens, MTT= method used to enter text on a touchscreen

### **4.3.2 METHODOLOGY**

The following sections describe the evaluation process for the SingleTapBraille application with respect to interface usability, learnability, accuracy and speed. The main task for the participants was to listen to phrases and enter those phrases as messages using each of the keyboards, QWERTY and SingleTapBraille. The independent variable here is input method (QWERTY and SingleTapBraille) and phrases; the dependent variable is the time required to type the text as well as error rate.

#### **4.3.2.1 Study Procedure**

The study was conducted at the Canadian National Institute for Blind (CNIB). For each participant, the study was consisted of a training session and test sessions. Prior to the training session, we conducted a pre-questionnaire to collect basic demographic data and other details related to the input methods typically used for text entry on a touchscreen (see Appendix A). In the training session, following an overview of the objective of the study, we explained SingleTapBraille's operation and functions, described how to do the finger tap on a touchscreen, and demonstrated how the finger taps relate to the Braille code and to the letters of the alphabet. To familiarize the participants with the input method, we provided a 20-minute training period during which they could practice using the application to type letters and numbers and to send a set of phrases as SMS messages using the SingleTapBraille application. In order to avoid potentially confounding effects related to participants memorizing particular phrases and learning how to type them, the set of phrases provided for training differed from those used for the test session. Data associated with the training session were not analyzed as part of this study.

In the first part of the test session, which lasted 20-30 minutes, participants listened to a set of phrases, typing each of them using the SingleTapBraille keyboard before moving on to the next phrase. We obtained a set of short message phrases from the English for Students website (English for Students, 2015). A moderate number of phrases was chosen in order to maintain a reasonable session length that would not discourage people from participating; the number of phrases used here was similar to other studies (Oliveira et al., 2011; Mattheiss et al., 2011). Participants were encouraged to type as quickly and accurately as possible. These phrases were sent as texts to the researcher's phone number. The text typed and the time required to type it were recorded in order to assess typing accuracy and typing speed, respectively. We used a video recorder in order to help us understand exactly how the participants interacted with a touchscreen. After the participants had texted all the phrases, they answered a post-study questionnaire regarding the usability and learnability of the SingleTapBraille approach and the SMS application (see Appendix B). Following that, we asked participants to type the same phrases using the QWERTY keyboard, providing another 20-30 minutes' session. After that, to the participants were further interviewed with respect to the strengths and weaknesses of the two keyboards assessed in the study; participants were also invited to suggest improvements with respect to the SingleTapBraille method (see Appendix C).

The entire session took each participant about 2 hours to complete. There was a short introduction, followed by a training session that took 15-20 minutes. Participants then took about 20-30 minutes to write the sentences using each input method (45-60 minutes total). The post-study interview took about 5-10 minutes.

### 4.3.3 RESULTS AND DISCUSSION

#### 4.3.3.1 Results

During the study, each participant typed 14 short messages for each keyboard, totaling 1596 characters, including space and symbols, on the SingleTapBraille and QWERTY keyboards. The results, detailed below, include quantitative measures of typing speed and accuracy, as well as qualitative findings related to the participants' experiences using the keyboards.

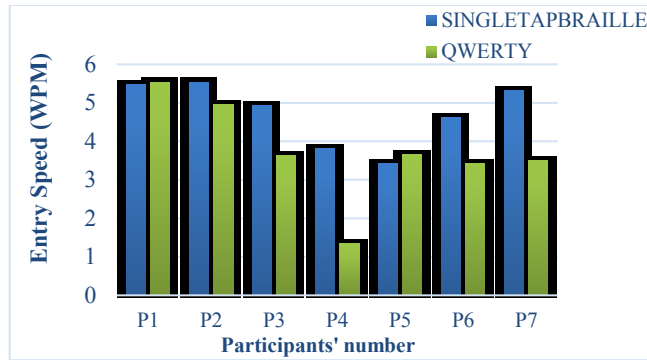
##### 4.3.3.1.1 Quantitative results

###### *1. Speed*

The measure used to assess text entry speed was Words-Per-Minute (WPM), calculated as (transcribed text length/ entry time in seconds \* 60 seconds in a minute) / (5 characters per word) (Soukoreff & MacKenzie, 2001).

$$\text{Typing Speed} = \frac{\text{Transcribed text length}/5 \text{ characters per word}}{\text{Entry time in in a minute}} \quad (1)$$

The designation of five characters, including letters, numbers, spaces and symbols, as the accepted word length was based on previous studies (e.g., Yamada 1980). There was a significant difference in the SingleTapBraille speed (Mean = 4.71 WPM) and the QWERTY keyboard speed (Mean = 3.72 WPM), paired t-test;  $p = 0.03$ ) (Figure 28).



**Figure 28: Text entry speeds (WPM) for each participant for SingleTapBraille and QWERTY keyboard**

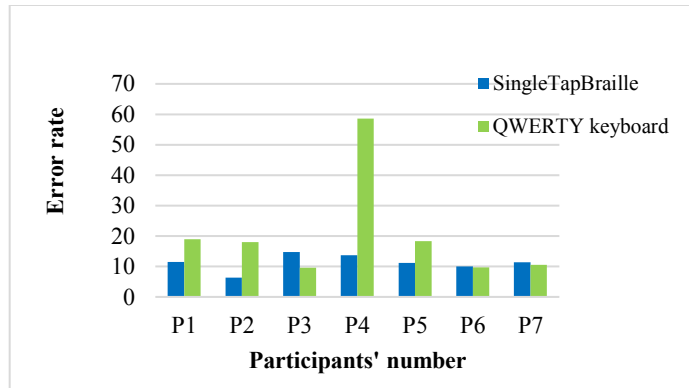
## 2. Accuracy

Participants were permitted to insert text without restrictions during the study, such that they may or may not correct any typing errors. Therefore, to measure accuracy we chose to use Minimum String Distance (MSD) (Soukoreff, MacKenzie, 2001). The MSD method analyzes three types of primitives, insertion, deletion, and substitution. Keystrokes are classified into four categories within the input stream: correct keystrokes (C), the number of erroneous keystrokes that are not corrected in the transcribed text (INF), the number of erroneous keystrokes and they are corrected in the transcribed text (IF), and the number of keystrokes that used to correct errors, such as delete or backspace (F).

This classification enables us to measure several statistics, including the total error rate:

$$\text{Total Error Rate} = \frac{(INF+IF)}{(C+INF+IF)} * 100\% \quad (2)$$

Total error rate (TER) represents the number of errors made while typing the phrases. The total error rate was lower for the SingleTapBraille keyboard than it was for the QWERTY keyboard indicating higher accuracy using the SingleTapBraille method. There was not a significant difference in the total error rate for SingleTapBraille (Mean=11.23%, SD=2.14) and QWERTY keyboard (Mean=20.54, SD=3.40), (paired t-test;  $p = 0.193$ ) (Figure 29).



**Figure 29: Average total error rate for each participant for SingleTapBraille and QWERTY keyboard**

### 4.3.3.2 Qualitative Feedback

#### 4.3.3.2.1 Pre-questionnaire

The participants were asked for their opinions regarding the possibility of a new input method, as well as the strengths and limitations of the keyboards they had used. Most of participants use the QWERTY keyboard. All participants agreed that the audio feedback is the best solution to confirm that they have located the intended letter.

The users were asked about the main difficulties associated with using touchscreen keyboards. The most common complaint related to the slowness of the QWERTY keyboard; for example, Participant 5 stated that “because I have to swipe my finger and listen to all letter I have passed. Finding a specific location of symbol takes time. Similarly, QWERTY keyboard in your phone it is much slower than a keyboard that you can have your finger tips to type with on a Bluetooth keyboard”.

Another obstacle reported was interference by alternative keyboards that pop up when a user pressed for a long on a letter. For example, Participant 3 stated that “I found something frustrating on a touch screen, which is if you swipe your finger across the screen



to locate a certain letter, another keyboard or something else pops up and it says “Alternative keyboards”, and then it types something else not the one I am looking for”.

Five participants made reference to the difficulty of locating the keys for specific letters, numbers and symbols. Participant 7 said, “I found difficulty to find the symbols and numbers more than letters. Because sometimes it layout differently and also because again it is a touchscreen. I always guessing where I put my finger”. Reflecting this difficulty, Participant 4, the only participant with limited experience using the QWERTY keyboard and sending short messages, only reached 1.10 wpm with audio feedback, indicating considerable difficulty in locating the requires characters.

One participant that uses MBraille keyboard, which is similar to BrailleTouch, said “I use six fingers to type based on Braille patterns and my thumbs hold the device from the back, which is awkward”. She also didn’t like other features of MBraille, including the need to copy text from MBraille to other applications (e.g., Facebook) and its tendency to shut down unexpectedly.

We also asked participants about how they hold their smartphones and interact with touchscreens device while typing. Most participants use one hand to stabilize the smartphone device while using the index finger of the other hand to tap on the screen. Unlike six of the participants, participant 3 reported a preference for using her thumb to type on a screen because she lost her sight when she was 20 years old; this method contrasts with the tendencies of most people who are born blind, who typically use their index finger to read Braille and type on a touch screen. We also asked the participants about sending messages while walking; only two participants do this, with the others reporting it was too difficult because they do not have two free hands while walking.

#### *4.3.3.2.2 Usability questionnaire*

The usability questionnaire contained 19 statements regarding the keyboard, with participants required to indicate their level of agreement with each statement on a 5-point Likert scale (1 = disagree strongly, 5 = agree strongly). The participants rated the keyboard highly with respect to usefulness, ease of use, and learnability. Nonetheless, they were unsatisfied with some aspects of speed and usability. Overall, however, the results of the questionnaire parallel the quantitative improvements in speed and reflect improvements related to usability, learnability and accessibility.

#### *4.3.3.2.3 Interview*

We developed approximately 15 qualitative codes to summarize the most relevant findings we learned from each participant, which we clustered into sets of high level themes, including difficulty typing the text, difficulty switching between keyboard layouts, audio feedback, annoying, convenient, easy to use, physical comfort and navigation issues.

All participants successfully used the newly developed approach to enter text with the help of the Text-to-Speech feature built into the keyboard itself. All participants found the audio to be clear and easy to follow. After the participants completed their test session, we asked them about various elements of their experiences using the keyboard.

All participants liked that interaction with the touchscreen did not involve a button or looking for a specific location. For example, Participant 3 said “I like it because I did not have to look all over the screen looking for something, it just a matter of tapping in or swiping, it is simple to use.” Participants also liked the use of swiping gestures, reporting that swiping from bottom to top and vice versa are easy gestures that blind users can perform quickly. Interviews also revealed that that swiping gesture to switch between

layers and to capitalize letters is easier for blind users than finding the switch or shift button on the QWERTY keyboard. As Participant 1 said, “It is simple, it is straightforward, it is intuitive, that particular function, I like better than buttons in the normal keyboard because it is quicker to flick up anywhere and switch between number, letters, and symbols than finding the little button.” Likewise, Participant 7 said “switching up and down is quite efficient and easier than trying to navigate with a QWERTY keyboard.” This finding is supported by previous research; for example, Wobbrock and Ladner (2011) found that the easiest gestures for blind users involve swiping rather than typing (Shaun et.al, 2011). Participants also reported that using a long press gesture anywhere on a touchscreen to perform a specific task is easier than finding a specific button. Participant 5 stated, “it was great: on my Ipad, If I want to send message, I have to swipe over to locate send button. I mean, it is much easier, pressing for long anywhere on a screen.” Participant 2 also said “that’s cool because you do not need to look for send button.”

Four of the participants were not satisfied with the keyboard speed, and they provided different possible reasons for its inadequate speed. Participant 7 stated that “I was not fast, because I have to think about Braille dots because I have never obviously typed with Braille on a touch screens. Whenever I typed Braille, I used six keys right in front of me, so it kind of the same I was typing on computers versus being on a regular keyboard with a voiceOver, I can type faster and it gets easier as you go.” In contrast, Participant 5 found the keyboard speed is normal, stating, “normal, a little of delay because I think that’s good because it gives you time to correct mistakes if you made one.” Most participants also suggested the implementation of Braille grade 2 in order to minimize the number of letters required to type a word.

The users were also asked about the features that they liked the most, as well as what they saw as the primary limitations of the SingleTapBraille keyboard. All seven participants liked the way of using Braille to enter text on a screen. Participant 2 said “the advantage is using something I grow up with it, I use Braille for 30 years.” Participant 3 agreed with that and added, “I am very keener for Braille, you will not forget Braille, if you keep using it, you will not forget the Braille combination of dots to type a letter. It is so straight forward and it is very simple to use.”

The use of the swiping gesture and the long press gesture to perform different tasks were both strengths according to the participants. Participant 2 also appreciated the ability to hold the smartphone using one hand, saying “I am walking it is difficult to feel the screen where is my d, a, s etc.; Here user can use one hand to enter text based on the shape of Braille letter, which is for somebody who knows Braille and use it well is more natural than slid your finger on the keyboard to find a specific letter.” Nonetheless, participants agreed on some limitations for the keyboard. One criticism, brought up by three of the participants, was that the keyboard does not repeat the word that has been typed out, an action that helps blind users confirm that they entered the correct word. Another limitation is the delay associated with users having to enter the Braille dots individually. Three participants had difficulty typing some letters, like n, p and r, due to the similarity of Braille code for these letters.

#### **4.3.3.2.4 Evaluation of Grade 2 encoding in SingleTapBraille**

To address the main limitation of the first prototype, we implemented the second type of Braille (Grade 2) to develop a complete prototype and we evaluated the second version of

the keyboard to obtain the participants' feedback about the feature that has been added, and also to guide the final evaluation study design.

After implementing Grade 2 Braille, the improved keyboard was evaluated with a blind participant to determine whether the implementation of Grade 2 Braille improved the accessibility and overall performance of the keyboard. During the evaluation, we explained the improved SingleTapBraille and how can a user open the Grade 2 Braille and insert words. Then, we asked the blind participant to type several words for 5 minutes as a training session. The participant was then required to type specific words that have abbreviations or signs in Grade 2 Braille. Some of these words can be represented by only the first letter of each word, for example, "can", "you", "do", "like", "people", and "rather". Other words can be represented using the first two letters in each word like "about", "after", "also", "again", "blind" and "according". After typing these words, we conducted an interview with the participant about the strengths and limitations of the improved keyboard. He stated that this keyboard is a very useful text entry tool. The most valuable features in this keyboard after integrating Grade 2 Braille is enabling a user to type words in a short time using Grade 2 abbreviations and without a need to insert each letter in a word. He said "I like it because it is much quicker than spelling the word right out because it is a shortcut for us". He also liked that the keyboard provides both Grade 1 and 2 levels. He stated "I like the way it allows us to have both options Grade 1 and 2 by swiping. It is much easier and we are not guessing where we are swiping and typing Braille dots." He also stated that "I could use Grade 1 to type but I prefer Grade 2 Braille because it is much faster for me than the other one."

He also likes the way that the keyboard speaks and types out the whole word. In Grade 2 Braille, only abbreviations of words can be typed, which might cause confusion. In addition, the participant pointed to a limitation with the keyboard after adding Grade 2. He found that in case the user types a wrong abbreviation, the keyboard will insert the whole word and the user has to delete the entire word letter by letter. Thus, he suggested we allow the users in Grade 2 Braille to delete entire words and not each letter.

Implementing this suggestion will definitely increase the speed of the keyboard. In general, the results of the initial evaluation of the improved keyboard confirm that the single tap approach is a promising direction for further exploration and development of an accessible touchscreen keyboard for the blind users.

This SingleTapBraille concept can be implemented in any mobile platform, including iOS, Android, and Windows phones. The components of designing the user interface of this approach are available across all platform versions. In the implemented approach, we used only essential gestures to create a simple interface including tap and swipe gestures that are easy to implement in other platforms. This approach does not require any element that is restricted to any specific platform release, and can be easily implemented across platform versions.

#### **4.3.3.3 DISCUSSION**

In this study, we compared the newly developed SingleTapBraille keyboard with the standard QWERTY keyboard, with respect to speed, and accuracy. In this chapter, we recap those results and discuss the suggestions from the participants regarding how to improve this keyboard. We also discuss the key factors that should be considered in developing an input method or application for the visually impaired.

#### **4.3.3.3.1 Speed and accuracy**

The SingleTapBraille keyboard demonstrated superior speed and accuracy compared to the QWERTY keyboard. The main advantage of the SingleTapBraille, likely leading to the observed differences in performance, is that it allows blind users to enter letters based on Braille coding. The reliance on Braille coding reflects one of SingleTapBraille's limitations, as the need to enter letters in several dot-based steps potentially slows the text entry process. Along with this, this keyboard requires care and accuracy because the tapping of the first dot directly influences how the following dots are interpreted. Nonetheless, it is still faster than the standard keyboard and other options available for this population. The average error rate was significantly less in the SingleTapBraille keyboard compared to the QWERTY keyboard. The results indicated that the SingleTapBraille keyboard is less prone to errors than the QWERTY keyboard, perhaps because users operating the QWERTY keyboard can get confused while hearing all letters that have passed as well as the letter that they have typed. In our keyboard, users only get audio feedback once they have typed a letter. Just as sighted people have to spend a lot of effort to understand a lot of presented visual information at once, blind users need to obtain audio feedback when necessary; thus, eliminating unnecessary sounds can increase accessibility and minimize cognitive load.

A limitation of the study is that we did not counter balance the order of using each keyboard.

#### **4.3.3.3.2 Keyboard Features**

Using a single tap to enter text based on Braille patterns significantly improves the accessibility for blind and visually impaired people while using touchscreens. The blind participants expressed that the SingleTapBraille keyboard makes typing text easier than it is with the QWERTY keyboard. It is much easier for blind users to tap anywhere on a screen rather than trying to locate a specific key on a screen and worry about hitting a wrong key. Swiping on a touch screen to perform a specific task was also considered an easy and fast way to accomplish tasks. Most participants found flicking up and down to switch the keyboard as well as flicking left to right to add or backspace to be much easier than finding a shift key on a QWERTY keyboard or finding a switch sign to switch to other layers or finding a space button to add space or delete button to perform backspace. It is also much faster to flick than it is find a specific button on a touch screen.

Some user interfaces on smartphones force blind users to use the QWERTY keyboard and do not support the alternative keyboards that have been designed to serve this population. For example, in the PIN (passcode) interface for a smartphone device, blind users must enter a password using the QWERTY keyboard. According to the participants, blind users find it easier to locate buttons near the corner and screen edges; therefore, they often create simple passwords that rely on these landmarks. As a result, blind users are at a disadvantage with respect to security, underlining the need to provide them with the ability to easily create and formulate any password combination.

Most participants usually use two hands to interact with touchscreens and have great difficulty using new methods, including using only one hand. Participant 1 said “it is different way of typing text. It takes some learning time because I have to train my muscles



to use to it”. Based on my observation, none of the participants were able to use predictive words on the QWERTY keyboards, indicating that this feature is essentially inaccessible for blind users. This feature therefore needs improvement in order to increase accessibility. All participants liked the keyboard because it does not require them to deviate from the language, Braille, that they use in daily life.

#### **4.4 BRAILLEENTER INPUT METHOD**

The BrailleEnter is an improvement version of the SingleTapBraille based on the participants’ suggestions, who evaluated the SingleTapBraille keyboard.

##### **4.4.1 The Framework of BrailleEnter Text Entry Method**

BrailleEnter is a new touchscreen text input method for blind people. The main goal of the research is to overcome the navigation problems that blind people face while interacting with touchscreens. We also aim to allow users to use one finger to interact with the screen. Based on previous studies the most accessible interaction technique with touchscreens for blind people is using one finger because it is the most accessible way for them (Paisios, 2012, Alnfai, Sampalli, 2016). The BrailleEnter keyboard description and functions are described in the following section. BrailleEnter has several functions which are: Typing small and capital letters, typing numbers, typing punctuation, editing functions (add space and backspace), and audio feedback. These five functions will be explained in detail in the following subsections.

#### **4.4.1.1 Typing function**

In the BrailleEnter keyboard, users can type based on Braille codes using one hand. In Braille, each character is represented by six dots, which are organized in two columns and are ordered from one to six. These Braille dots are inserted from left to right based on the technique in which blind users read Braille.

Each character has a combination of activated dots and inactivated dots in different orders. Based on the number of activated dots and inactivated ones along with their orders, users can distinguish between characters. For example, when typing the letter “a”, the first Braille dot is activated and the left five dots are inactivated. For letter “b”, the first two dots are activated and the other four dots are inactivated.

In our input method, users can use two types of gestures to represent active and inactive Braille dots. BrailleEnter uses a press gesture to represent active dots and a tap gesture to represent inactive dots. The press gesture is defined as a long tap and the tap gesture is defined as a brief tap.

In BrailleEnter, users are required to tap or press on a touchscreen six times to represent a letter. However, they can tap or press anywhere on the screen without any concern about the location of interactions, they can tap inactivated Braille dots and press the activated dots anywhere.

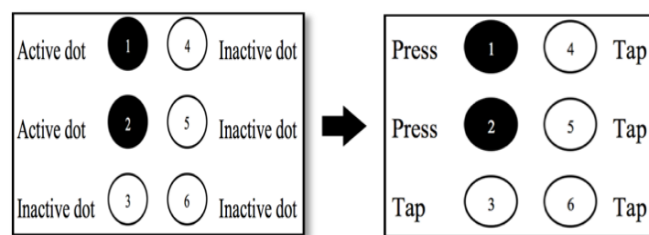
#### **4.4.1.2 BrailleEnter algorithm**

From a previous study, we found that blind people do not know what user interfaces look like and sometimes where they have touched on a screen (Mattheiss et al., 2015). Once they move their fingers from screens, they are not always able to indicate where they have

been. Most of their interactions on touchscreens – and especially button based interfaces – are completely based on guessing. Therefore, our algorithm does not require users to find an object location nor use two hands to interact with a screen. The BrailleEnter algorithm is designed based on two main concepts, which are:

1. Defining the active dots and inactivated dots in each character
2. Determining the order of these active and inactive Braille dots in each character
3. Determining the number of the active and inactive dots in each character

Blind users already know the active and the inactive dots in each character and their orders. When a blind user taps and presses on touchscreen six times, the BrailleEnter algorithm analyses the inserted gestures and interprets them based on gesture types as well as their orders. The algorithm analyses each dot and distinguishes between a brief tap or a long press on the screen, as well as determines the order. This algorithm eliminates the need to check the location of an interaction event on a touchscreen because this is invaluable factor for blind users. Figure 30 demonstrates how blind users can type letter b using the BrailleEnter keyboard.



**Figure 30: Typing the letter “b” in BrailleEnter**

The swiping gesture is the best and most accessible interaction technique on touchscreen devices for blind people (Mattheiss et al., 2015). In order to change the keyboard mode, users can swipe vertically in a loop to move between keyboard modes.

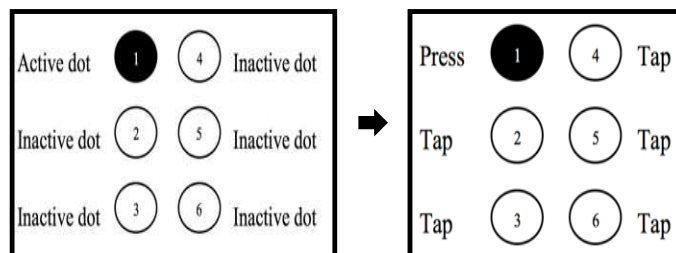
The first swipe will allow users to type capital letters and audio feedback will speak the name of the keyboard mode. Then, if the users swipe vertically twice, they will initiate the second mode, which is for typing numbers (see Table 6). For instance, when the user performs the first vertical swipe from bottom to top, the app will open up “capital letter” mode. In case the user does not want the “capital letter” mode, he/she can swipe again from bottom to top to type numbers instead of capital letters. Performing a vertical swipe from top to bottom will open up “lower case” mode. A second swipe from top to bottom will insert “symbols” mode instead of “small letter” mode.

**Table 6. BrailleEnter keyboard interaction techniques**

Interaction techniques	purposes
Swipe from bottom to top	Capital letters, numbers
Swipe from top to bottom	Small letters, symbols

#### 4.4.1.3 Typing numbers

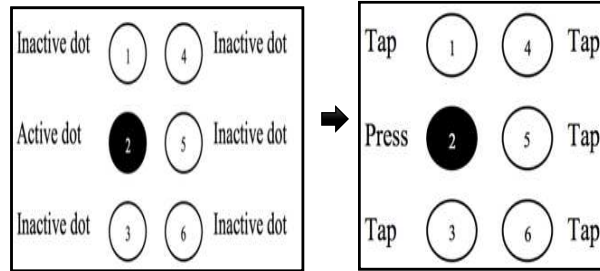
In BrailleEnter, blind users can type numbers as they do in Braille. First, they must change the keyboard mode by swiping to number mode. Second, they can insert Braille dots based on gesture type. For example, in order to type the number 1, users press, tap, tap, tap, tap (as shown in Figure 31).



**Figure 31. An example of typing a number in BrailleEnter**

#### 4.4.1.4 Typing punctuation

The BrailleEnter keyboard also enables users to type punctuation based on Braille coding. Users can tap the inactivated dots and press the activated dots to represent the punctuation. For example, in order to insert a comma, users tap, press, tap, tap, tap, tap) (as shown in Figure 32).



**Figure 32.** Type symbol “,” in BrailleEnter

#### 4.4.1.5 Editing function

Space and backspace operations have specific keys in the physical Braille machine and the other six keys are used to represent Braille dots.

Our keyboard also uses horizontal swiping gestures to enable the BrailleEnter keyboard to perform editing functions, as detailed in Table 7. BrailleEnter uses audio feedback to speak out the defined gesture’s function name to facilitate editing.

**Table 7. Editing interaction methods and their functions in BrailleEnter**

Interaction methods	function
Swipe from left to right	Add space
Swipe from right to left	Backspace

#### **4.4.1.6 Using voice feedback**

BrailleEnter applies the TalkBack service to read out the entered text on a touchscreen as well as read out editing functions while users swipe over the screen. This service enables blind users to make sure they entered the intended letter.

#### **4.4.1.7 Keyboard advantages**

The main advantage of our keyboard is that it is eyes-free; users can interact anywhere on the screen because the screen layout does not require users to locate a key on a screen. Allowing the users to interact anywhere on a screen will significantly reduce the error rate. The keyboard layout provides audio feedback that enables blind users to have the same experience as sighted users. It is quite easy to use because it is mainly based on their mother language, Braille, and it follows the approach of slate and stylus technique, which is the old-fashioned Braille machine (WritingBraille, 1999). It is also cost effective because nowadays most blind people own a smartphone device, thus it is much cheaper to download a new keyboard than to buy a physical keyboard. The keyboard layout uses accessible gestures, which are tapping, pressing and swiping. All these gestures are easy for blind people to use (Alnfiai, Sampalli, 2016). In addition, the keyboard allows blind users to use a thumb or one finger to insert a character, which is the most accessible way.

### **4.5 BRAILLEENTER KEYBOARD IMPLEMENTATION**

In this section, we describe the implementation of the BrailleEnter keyboard to allow blind users type on touchscreen easily. We will describe the process of entering character using the BrailleEnter, its functions and interfaces.

## 4.5.1 Experimental model

In this section, we describe the implementation of the BrailleEnter keyboard. We will describe the BrailleEnter interface as well as typing and editing functions.

### 4.5.1.1 User interaction module

This section presents the BrailleEnter architecture of the user interaction module, which includes:

1. **Gesture pattern recognition module** (see section 4.2.3.1)
2. **Braille character pattern recognition module**
3. **Keyboard modes** (see section 4.2.3.2)
4. **Editing module** (see section 4.2.3.3)
5. **Text to speech module** (see section 4.2.3.4)

To enter a character on a touchscreen using the BrailleEnter keyboard, six touches are required to recognize a character. The BrailleEnter architecture contains six modules: the first detects the gesture pattern; the second recognizes Braille characters; the third module determines the keyboard mode; the fourth, edit module, includes gestures for special keys such as adding a space or backspace; and the fifth, Text-to-Speech module speaks out each input entered or removed from the screen.

#### 1. Gesture Recognition Algorithm

We followed the same process in the implementation to detect gestures enacted on the smartphone screen. We also used *onTouchEvent()* method with *MotionEvent class* to

detect common gestures such as Tap, Press and Swipes on a screen ( as explained in section 4.2.3.1).

The input values include six combinations of tapping and pressing gestures that taken from the smartphone screen. The algorithm produces expected outputs (letters, number or punctuations) based on the order of the entered gestures and their types.

## 2. Braille character pattern recognition module

The input variables are the touching events that users perform on the screen. This algorithm requires six inputs to produce one character. It processes these six touches or gestures (input variables) by analyzing the gestures' type and their orders. The active dots are represented by long tap gestures and the inactive dots are represented by short tap gestures. Character Recognition Algorithm follows four main steps to recognize and enter a character, which are:

1. Detect and store the gestures
2. Check the order and type of the six gestures that are stored in the array list (is the gesture tapping or pressing and their order from 0 to 5)
3. Check the keyboard modes (numbers, letters and punctuations)
4. Enter the character

---

**/\* Character Recognition Algorithm \*/**

**Comment:** gesture/Tapping and Pressing by user recognition (Gesture recognition)

**Comment:** Enter the type of gesture from the list (TA)

**Step1.** Time == 0 and the size of array list is zero (c[]). // Time handler

**Step2.** Add touch gestures on a screen to the array list c[]

**Step3.** Start the time count once a user touches the screen

**Step4.** Detect the gesture type (*Gesture recognition algorithm*)

**Step5.** Store the gesture type and order in an array list, c[]

---



---

**Step6. DO** step 2, 3 and 5 six times (representing the six dots of Braille code)

**Step7. If** (array list has more than six gestures) **then**

**return** “more than six dots”

**else if** (array list has less than six gestures) **then**

**return** “less than six dots”

**end if**

**else If** (The first digit == Long tap gesture && The second digit == Tap gesture && The third digit == Tap gesture && the fourth digit == Tap gesture && the fifth digit == Tap gesture && the sixth digit == Tap gesture)

**then**

**if** (the keyboard in uppercase mode)

**Enter** “Letter A”

**else if** (the keyboard in lowercase mode)

**Enter** “Letter a”

**else if** (the keyboard in number mode)

**Enter** “number 1”

**end if**

**else If** (The first digit == Long Tap gesture && The second digit == Long Tap gesture && The third digit == Tap gesture && the fourth digit == Tap gesture && the fifth digit == Tap gesture && the sixth digit == Tap gesture) **then**

**Enter** “Letter b”

**if** (the keyboard in uppercase mode)

**Enter** “Letter B”

**else if** (the keyboard in lowercase mode)

**Enter** “Letter b”

**else if** (the keyboard in number mode)

**Enter** “number 2”

**end if**

**else If** (The first digit == Long Tap gesture && The second digit == Tap gesture && The third digit == Tap gesture && the fourth digit == Long Tap gesture && the fifth digit == Tap gesture && the sixth digit == Tap gesture)

**then**

**if** (the keyboard in uppercase mode)

**Enter** “Letter C”

**else if** (the keyboard in lowercase mode)

**Enter** “Letter c”

**else if** (the keyboard in number mode)

**Enter** “number 3”

**end if**

**else If** (The first digit == Long Tap gesture && The second digit == long Tap gesture && The third digit == Tap gesture && the fourth digit == Long Tap gesture && the fifth digit == Tap gesture && the sixth digit == Tap gesture)

**then**

**if** (the keyboard in uppercase mode)

**Enter** “Letter D”

**else if** (the keyboard in lowercase mode)

---

---

```

    Enter "Letter d"
  else if (the keyboard in number mode)
    Enter "number 4"
  end if
else If (The first digit == Long Tap gesture && The second digit == Tap
gesture && The third digit == Tap gesture && the fourth digit == Tap gesture
&& the fifth digit == Long Tap gesture && the sixth digit == Tap gesture)
then
  if (the keyboard in uppercase mode)
    Enter "Letter E"
  else if (the keyboard in lowercase mode)
    Enter "Letter e"
  else if (the keyboard in number mode)
    Enter "number 5"
  end if
else If (The first digit == Long Tap gesture && The second digit == long Tap
gesture && The third digit == Tap gesture && the fourth digit == Long Tap
gesture && the fifth digit == Tap gesture && the sixth digit == Tap gesture)
then
  if (the keyboard in uppercase mode)
    Enter "Letter F"
  else if (the keyboard in lowercase mode)
    Enter "Letter f"
  else if (the keyboard in number mode)
    Enter "number 6"
  else if (the keyboard in symbol mode)
    Enter "symbol !"
  end if
else If (The first digit == Long Tap gesture && The second digit == long Tap
gesture && The third digit == Tap gesture && the fourth digit == Long Tap
gesture && the fifth digit == Long Tap gesture && the sixth digit == Tap
gesture) then
  if (the keyboard in uppercase mode)
    Enter "Letter G"
  end if
  else if (the keyboard in lowercase mode)
    Enter "Letter g"
  end if
  else if (the keyboard in number mode)
    Enter "number 7"
  end if
else If (The first digit == Long Tap gesture && The second digit == long Tap
gesture && The third digit == Tap gesture && the fourth digit == Tap gesture
&& the fifth digit == Long Tap gesture && the sixth digit == Tap gesture)
then
  if (the keyboard in uppercase mode)

```

---

---

```

    Enter "Letter H"
end if
else if (the keyboard in lowercase mode)
    Enter "Letter h"
end if
else if (the keyboard in number mode)
    Enter "number 8"
end if
else If (The first digit == Tap gesture && The second digit == long Tap
gesture && The third digit == Tap gesture && the fourth digit == Long Tap
gesture && the fifth digit == Tap gesture && the sixth digit == Tap gesture)
then
    if (the keyboard in uppercase mode)
        Enter "Letter I"
    end if
    else if (the keyboard in lowercase mode)
        Enter "Letter i"
    end if
    else if (the keyboard in number mode)
        Enter "number 9"
    end if
else If (The first digit == Tap gesture && The second digit == long Tap gesture
&& The third digit == Tap gesture && the fourth digit == Long Tap gesture
&& the fifth digit == Long Tap gesture && the sixth digit == Tap gesture)
then
    if (the keyboard in uppercase mode)
        Enter "Letter J"
    end if
    else if (the keyboard in lowercase mode)
        Enter "Letter j"
    else if (the keyboard in number mode)
        Enter "number 0"
    end if
else If (The first digit == Tap gesture && The second digit == long Tap gesture
&& The third digit == Tap gesture && the fourth digit == Tap gesture && the
fifth digit == Long Tap gesture && the sixth digit == Long Tap gesture) then
    Enter "symbol ."
end if
else If (The first digit == Tap gesture && The second digit == long Tap gesture
&& The third digit == Tap gesture && the fourth digit == Tap gesture && the
fifth digit == Tap gesture && the sixth digit == Tap gesture) then
    Enter "symbol ,"
end if
else If (The first digit == Long Tap gesture && The second digit == long Tap
gesture && The third digit == Tap gesture && the fourth digit == Tap gesture
&& the fifth digit == Long Tap gesture && the sixth digit == Tap gesture)then

```

---

---

```

    Enter "symbol ?"
  end if
else If (The first digit == Tap gesture && The second digit == Tap gesture &&
The third digit == Long Tap gesture && the fourth digit == Tap gesture &&
the fifth digit == Tap gesture && the sixth digit == Tap gesture) then
  Enter "symbol "
  end if
else If (The first digit == Tap gesture && The second digit == Long Tap
gesture && The third digit == Tap gesture && the fourth digit == Tap gesture
&& the fifth digit == Long Tap gesture && the sixth digit == Tap gesture)
then
  Enter "symbol -"
  end if
else If (The first digit == Tap gesture && The second digit == Tap gesture &&
The third digit == Long Tap gesture && the fourth digit == Long Tap gesture
&& the fifth digit == Long Tap gesture && the sixth digit == Long Tap
gesture) then
  Enter "symbol #"
  End if

```

**Step8.** Stop the time after 3 seconds

**Step9.** Clear the array list

**Step10.** Repeat step 1 to 9 until user types the whole text

**Step 11. End Character Recognition Algorithm**

---

**Figure 33. Braille characters recognition algorithm**

```

if (mTouchHelper.size() > 6) {
    if (!mTouchHelper.get(0).isPressed() && !mTouchHelper.get(1).isPressed() && !mTouchHelper.get(2).isPressed() && !mTouchHelper.get(3).isPressed() &&
        !mTouchHelper.get(4).isPressed() && !mTouchHelper.get(5).isPressed() && !mTouchHelper.get(6).isPressed()) {
        Speak("more than 6 dots");
    }
} else {
    // type a
    if (mTouchHelper.get(0).isPressed() && !mTouchHelper.get(1).isPressed() && !mTouchHelper.get(2).isPressed() && !mTouchHelper.get(3).isPressed() &&
        !mTouchHelper.get(4).isPressed() && !mTouchHelper.get(5).isPressed()) {
        if (mode == 0) {
            writeText('a');
        } else if (mode == 1) {
            writeText('A');
        } else if (mode == 2) {
            writeText('1');
        }
    }
} else
    //type b
    if (mTouchHelper.get(0).isPressed() && mTouchHelper.get(1).isPressed() && !mTouchHelper.get(2).isPressed() && !mTouchHelper.get(3).isPressed() &&
        !mTouchHelper.get(4).isPressed() && !mTouchHelper.get(5).isPressed()) {
        if (mode == 0) {
            writeText('b');
        } else if (mode == 1) {
            writeText('B');
        } else if (mode == 2) {
            writeText('2');
        }
    }
}

```

**Figure 34. Snapshot of Braille characters recognition code**

```

// type (
else if (!mTouchHelper.get(0).ispressed() && mTouchHelper.get(1).ispressed() && mTouchHelper.get(2).ispressed() &&
mTouchHelper.get(3).ispressed() && mTouchHelper.get(4).ispressed() && mTouchHelper.get(5).ispressed()) {
    if (mode == 3) {
        writeText('(');
    }
}

// type *
else if (mTouchHelper.get(0).ispressed() && !mTouchHelper.get(1).ispressed() && !mTouchHelper.get(2).ispressed() &&
!mTouchHelper.get(3).ispressed() && !mTouchHelper.get(4).ispressed() && mTouchHelper.get(5).ispressed()) {
    if (mode == 3) {
        writeText('*');
    }
}

// type <
else if (mTouchHelper.get(0).ispressed() && mTouchHelper.get(1).ispressed() && !mTouchHelper.get(2).ispressed() &&
!mTouchHelper.get(3).ispressed() && !mTouchHelper.get(4).ispressed() && mTouchHelper.get(5).ispressed()) {
    if (mode == 3) {
        writeText('<');
    }
}

// type %
else if (mTouchHelper.get(0).ispressed() && !mTouchHelper.get(1).ispressed() && !mTouchHelper.get(2).ispressed() &&
mTouchHelper.get(3).ispressed() && !mTouchHelper.get(4).ispressed() && mTouchHelper.get(5).ispressed()) {
    if (mode == 3) {
        writeText('%');
    }
}
}

```

**Figure 35.** Snapshot of Braille punctuations recognition

```

private void process() {

    Log.e("onpresssize", "" + mTouchHelper.size());

    timer = new Timer();
    final Handler handler = new Handler();
    TimerTask task = run() -> {
        handler.post(run() -> {

            if (mTouchHelper.size() == 0) return;

            Speak("Couldn't recognise letter");
            mTouchHelper.clear();

        });
    };
    timer.schedule(task, 1000 * 3, 1000 * 3);
}

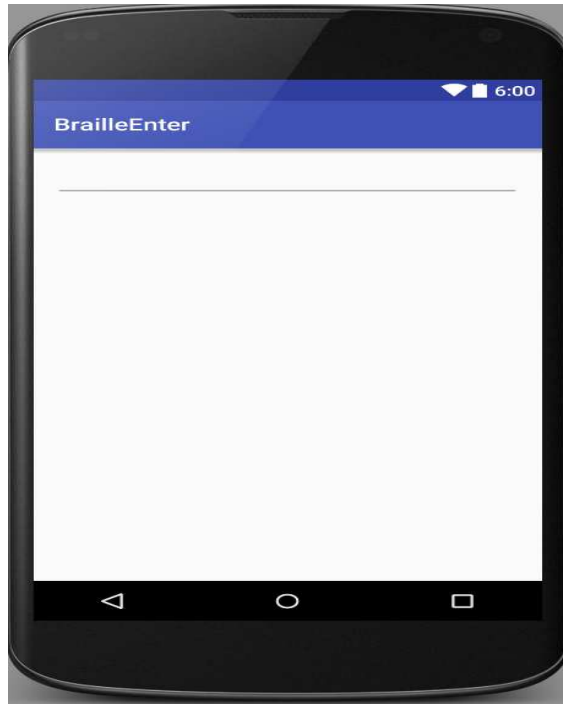
```

**Figure 36.** Time handler

#### 4.5.1.2 BrailleEnter Keyboard interface

The BrailleEnter keyboard is designed for blind people who know Braille. The main objective of the keyboard is to make touchscreen applications more accessible for blind

users and to decrease the typing error rate on a touchscreen. The BrailleEnter keyboard feedback interface is completely based on speech and has contains no visual content (see Figure 37).



**Figure 37.** BrailleEnter interface

#### **4.6 Preliminary evaluation**

We conducted an empirical evaluation to compare BrailleEnter, which is button free keyboard to Swift Braille, which is button-based. Both of them were designed based on Braille code. We conducted the evaluation study with eight blind participants who know Braille very well to evaluate the performance of both keyboards. Next, we describe the participants, the study design, apparatus, text phrases and the evaluation metrics that used.

#### 4.6.1 Participants

The study was conducted at the Canadian National Institute for Blind (CNIB) in Halifax and Taif accessibility center in Saudi Arabia. An approval from the Dalhousie University Social Sciences and Humanities Research Ethics Board was acquired before commencing the study (see Appendix F). We recruited eight participants, (five females, three males), with an average age of 40. Six participants are from CNIB and 2 participants are from Taif university accessibility center. The same CNIB participants (P1 to P6) who participate in the SingleTapBraille study joined again in the BrailleEnter study. All participants read and write Braille at a professional level. One of the requirements to participate in the study is knowing Braille very well because knowing the Braille code of each character is essential for each participant to be able to enter text using Braille keyboards. All participants have had experience with touchscreens for at least 3 years and all of them have been using the QWERTY keyboard with VoiceOver as the main input method on smartphone devices (see Table 8). Five of the participants are iPhone users and one is iPad user. All of them hold a smartphone device with one hand and interact with a screen using their index finger.

**Table 8: Participants' demographic data**

<b>P</b>	<b>Age</b>	<b>Gender</b>	<b>Sight</b>	<b>Braille device used</b>	<b>Experience with smartphones</b>	<b>Touchscreen keyboard</b>
<b>P1</b>	49	Male	Legally blind	Perkins Braille, Slate and stylus	7 years	Qwerty keyboard
<b>P2</b>	51	Male			10 years	
<b>P3</b>	36	Female			5 years	
<b>P4</b>	30	Female			6 years	
<b>P5</b>	46	Female			3 years	
<b>P6</b>	62	Male			8 years	+ MBraille
<b>P7</b>	20	Female	Visually impaired		5 years	Qwerty keyboard
<b>P8</b>	27	Female	Legally blind		6 years	

#### 4.6.2 Apparatus

We chose to compare our developed keyboard with the Swift Braille keyboard for several reasons: both are developed based on Braille and both have the same aims of our keyboard, which is allowing users to use one finger while interacting with a screen. Also, some of the Braille based keyboards that presented in the related work were not full developed and did not enable users to type all Braille characters and some Braille keyboards were not available at the time we conducted the evaluation study. The Swift Braille keyboard was selected based on its high number of downloads and customer reviews (as shown in Table 9).

**Table 9. Touchscreen keyboards designed for blind and visually impaired**

Keyboards	Approach	Android		iPhone	
		downloads	Fee	downloads	Fee
MBraille	Multi-touch + location based approach	60 (Free)		50 (\$54)	
BrailleTouch		✕		✕	
BrailleKey	Location based approach	✕		✕	
<b>Swift Braille</b>		71 (Free)		✕	
EdgeBraille		✕		✕	
BrailleSkitch		✕		✕	
BrailleType		✕		✕	
Keeble (QWERTY)		✕		5 (\$34)	
TypeinBraille		✕		19 (\$5.06)	
BrailleEasy	Multi-touch approach	✕		0 (Free)	
iOS Braille		✕		✓	
Super Braille Keyboard		7 (\$9)		✓	
Perkinput		✕		0 (Free)	
Soft Braille		54 (Free)		✕	
GBraille		2 (Free)		✕	
Eyedroid		✕		✕	
BrailleÉcran	Tactile layout	✕		✕	
MoonTouch		✕		✕	
BrailleSketch		✕		✕	



Table 9 shows all the input methods that developed on either Android or iPhone smartphone devices. Some of these keyboards are available on both platforms and some of them are available only on one platform. Not all mentioned keyboards are available for public they are still under development or evaluation.

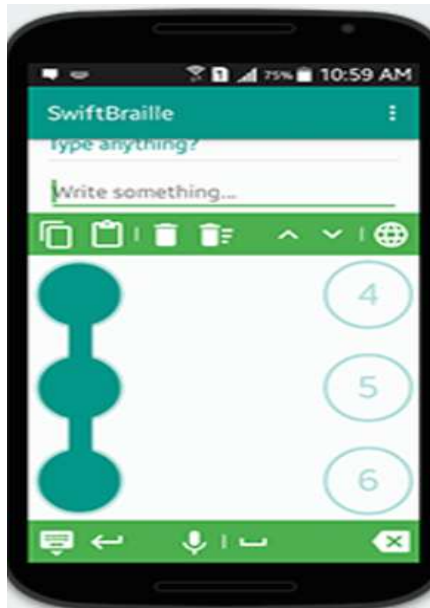
For the study tasks, we used an Android mobile phone that runs on Google's operating system. The ASUS smartphone (Android version 5.0.2) was used in each experiment to compare both the BrailleEnter and Swift Braille keyboards. We developed the BrailleEnter prototype on an ASUS smartphone, which runs an Android operating system (Android version 5.0.2). We installed the Swift Braille keyboard from Google play on the same Android device to control other variables that associated with smartphone devices, including audio feedback clarity, screen size as well as screen sensitivity.

The BrailleEnter prototype only utilizes Grade 1 Braille to perform a fair comparison with the Swift Braille keyboard, which only implemented Grade 1 Braille. The same phrases had to be inserted on both keyboards. Phrases and keyboards were presented randomly to participants to avoid learning and fatigue effects.

#### **4.6.2.1 Swift Braille keyboard**

Swift Braille is a soft keyboard developed for blind and visually impaired people (AlBanna, 2016; Mattheiss et al., 2015). The keyboard interface has six large buttons, which represent Braille dots and are located on the edges and corners of a smartphone screen (as shown in Figure 38). The main concept of this keyboard is that it uses one finger to connect the wanted Braille dots of each character. It enables users to type letters, numbers, and

symbols. It uses Text-to-Speech service to read out the inserted letter when the user's finger is lifted. Users can add a space by flicking from right to left and a back space by flicking from left to right. The main advantage of this keyboard is it allows users to use only one finger to type on a screen.



**Figure 38. Swift Braille keyboard**

#### **4.6.2.2 Text Phrases**

The phrases chosen were commonly used, simple and easy to remember. There are three test phrases for each text type (a total of 12 phrases). The order of the phrases was randomized. This procedure will help researchers measure WPM and error rate and allow us to compare between two keyboards while people type the same sentences.

These phrase types are:

1. **Sentences:** The goal of this task is to evaluate the usability while typing capital and small letters
2. **Number:** The goal of this task is to evaluate the usability while typing contact number

3. **Address:** This goal of this task is to evaluate the usability while typing both numbers and letters
4. **Email:** This goal of this task is to evaluate the usability while typing symbols and letters.

**Table 10: The twelve test phrases used**

No.	Phrases' types	Phrases	Chars.
1	Sentences	Don't worry about it.	21
2		How are you?	12
3		I will be arriving at 10 o'clock.	33
4	Addresses	36 Robie Street, Halifax	24
5		56 Queen Street, Dartmouth	26
6		90 Oxford Street, Sackville	27
7	Email	<u>Adam@hotmail.com</u>	16
8		<u>daniel@gmail.com</u>	16
9		<u>Chris@yahoo.com</u>	15
10	Numbers	My number is 1902345777	23
11		Her contact is 14893456787	26
12		Call him at 18745690345	23

Based on our previous experiment with blind participants, we observed that 12 sentences were adequate. We could not increase the number of sentences without avoiding making them bored and tired. We also observed that participants have trouble remembering long sentences, and to avoid cognitive load, we provided short, commonly used sentences that could be easily remembered by users (as shown in Table 10). All of the chosen phrases are 27 characters or shorter, which was a recommendation derived from other keyboard evaluations with blind people (Azenkot et al., 2014). We asked them to insert different types of text to simulate realistic text entry. Usually users need to add a contact number or email, send an email, or send an address. Researchers read out a phrase and participants

were asked to repeat the phrase orally before typing it to make sure they heard the phrase correctly.

#### **4.6.3 Experimental Design**

The experiment used 2\*4 within-subject design structure. The two main independent elements are:

1. Keyboard type: BrailleEnter keyboard, Swift Braille keyboard
2. Text type: sentences, numbers, addresses and emails.

All participants examined both keyboards, and the order of testing was balanced using a 2x2 Latin square. All participants typed all 12 phrases, and the order of typing was balanced using 12x12 Latin square. We counterbalanced the two keyboard trials across participants and text types to control for ordering type. The dependent variables are entry speed (words per minute or WPM), error rates, and keystrokes per character (KSPC). For the subjective evaluation, we used two questionnaires: First, the Lewis questionnaire to rate ease of inserting letters, ease of fast typing, ease of accurate typing, ease of learning keyboard approach, ease of typing, and acceptability of keyboard design (Lewis, 1995).

In the experiment, users were seated in the office, holding the mobile device with one hand and interacting with the screen with the other. We used a stand with a camera to record the typing tasks and to record the smartphone screen and the way participants' hands interacted with the screen to help us determine types of errors and causes. Figure 39 illustrates the experiment setup.



**Figure 39: Experimental setup**

#### **4.6.3.1 Procedure**

At the beginning of the study, the study objectives and tasks were explained and the participant completed the informed consent process. Then, a pre-questionnaire was conducted to collect the participants' demographic data (e.g. age, gender, vision status, Braille devices used, touchscreen experience, type of smartphone, touchscreen keyboard used, difficulties that participants face while using the keyboard, number of hands used to type on a screen, tasks that perform while using smartphone devices, etc.) and determine the participant experiences with available touchscreen keyboards that are designed mainly for blind users (see Appendix E).

The evaluation study was divided into two sessions per participant for each keyboard: training session and test session. All participants were asked to try and test the two keyboards. Both training and testing were performed on an ASUS Android phone with Android 5.0.2.

At the beginning of the training phase, participants underwent a training session for

one of the keyboards. The researcher showed the participant how the keyboard works, explaining how participants can type and edit their writing. To make them familiar with the input method, we instructed them to type two sentences using the keyboard. These two sentences are different from the phrases that were used for the test session to avoid learning effect contamination. The first sentence was: “*The quick brown fox jumps over the lazy dog.*” which has all the English letters and spacing. The second phrase is: “56 Tobin Street”, which has numbers and letters. We also asked them to use the keyboard to learn other gestures and functions, allowing users to interact with the keyboards for 15 minutes. After completing the training with the first keyboard, participants had a short break.

In the study session, participants were asked to use the keyboard and insert the 12 chosen phrases as quickly and as accurately as possible. In case participants made errors, they could correct their typing mistakes using the backspace feature of each keyboard. After typing the phrases, participants were asked to rate the tested keyboard using questions from the Lewis’ rating questionnaire that pertain to the design of Braille keyboards (Lewis, 1995). Since some of the Lewis’ questionnaire focuses only on physical keyboard features, we made slight changes to make this questionnaire valuable for touchscreen keyboards (see Appendix G). After completing the study tasks with the first keyboard, participants had an interview about the keyboard’s strengths and limitations and to discuss the features they liked and disliked (see Appendix H). During the interview, an audio recorder was used to help us record the information faster than writing.

After that, participants had a short break and then the researcher showed the participants how the second keyboard works, explaining the keyboard functions and operations. Then participants followed the same procedures with the other keyboard.

Participants practiced using the keyboard and were asked to write the same phrases using the other keyboard. They again completed the Lewis' rating questionnaire, then they had an interview about the keyboard's strength and limitations.

After completing training and study sessions for both keyboards, participants were asked to rate the keyboard interaction techniques in order of preference. This step helped us figure out which keyboard participants prefer and the reasons why they prefer one keyboard over the other. At the end of the study, the participants were compensated \$20 for their time.

The entire session took each participant about 2 hours to complete. There was a short introduction, followed by a training session that took 15-20 minutes. Participants then took about 30 minutes to write the sentences using each input method (45-60 minutes total). The post-study interview took about 5-10 minutes.

#### **4.6.4 Evaluation Metrics**

To evaluate user experiences for both Braille keyboards, we measured efficiency (speed), effectiveness (accuracy), and user satisfactions (questionnaire).

##### **a. Speed**

Typing speed is measured by word per minute (WPM), which is the number of words inserted per minute, where a word contained five characters. Timing was measured as the time from the first interaction with the keyboard layout until the last interaction.

## **b. Accuracy**

In the study, we permitted users to type naturally so if they made a spelling error, they may or may not correct it. Thus, Minimum String Distance (MSD) is used to measure accuracy (Soukoreff and MacKenzie, 2003), and keystrokes per character (KSPC). In the MSD method, there are four types of keystrokes in the input stream, which are: the number of correct keystrokes (C), the number of incorrect keystrokes in the transcribed text (INF), the number of incorrect keystrokes fixed in the transcribed text (IF), and the number of keystrokes used to correct typing errors, such as backspace (F).

Based on the observations, the time spent to type the chosen phrases varies significantly among both keyboards and participants. That means the time spent to type each sentence might be affected by time that participants spent editing their typing mistakes. To address this, we chose to use the Corrected Error Rate (CER), and Not Corrected Error Rate (NCER).

### **1. Error rate:**

$$\text{Total error rate} = \frac{(INF+IF)}{(C+INF+IF)} * 100 \quad (3)$$

Total Error Rate shows the percentage of typing mistakes made by users while typing the sentences using a particular input method.

### **2. Corrected error rate**

$$\text{Corrected Error Rate} = \frac{(IF)}{(C+INF+IF)} * 100\% \quad (4)$$

Corrected Error Rate shows the percentage of typing mistakes made by users while typing the sentences using a particular input method and then corrected.



### 3. Uncorrected error rate

$$\text{Not Corrected Error Rate} = \frac{(INF)}{(c+INF+IF)} * 100\% \quad (5)$$

Total Not Corrected Error Rate shows the percentage of typing mistakes made and not corrected.

### 4. KSPC

$$\text{KSPC} = \frac{|\text{Input stream}|}{|\text{Transcribed text}|} \quad (6)$$

We also used KSPC (keystrokes per character) in order to measure the cost of making typing errors and correcting them, as well as the effort that participants chose to invest in error correction (Soukoreff and MacKenzie, 2001).

#### c. Lewis' rating questionnaire

After typing the phrases, participants were asked to rate the tested keyboard using questions from the Lewis' rating questionnaire pertaining to the design of Braille keyboards (Lewis, 1995). Since some of the Lewis' questionnaire focuses only on physical keyboard features, we made slight changes to make this questionnaire valuable for touchscreen keyboards. The questionnaire has nine item scales, where the scale starts from 1 very likely to 7 very unlikely. That means lower scores are better than higher scores.

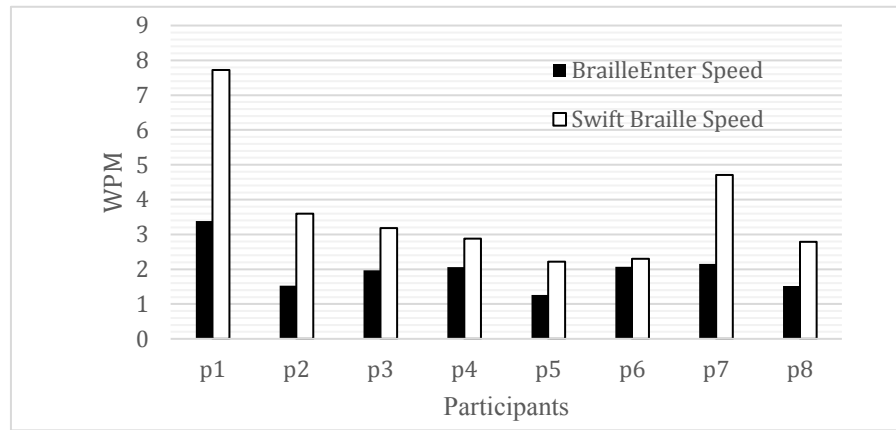
#### 4.6.5 Results and Discussion

Statistical analysis of data collected was performed and processed in Microsoft Excel program.

#### 4.6.5.1 Quantitative results

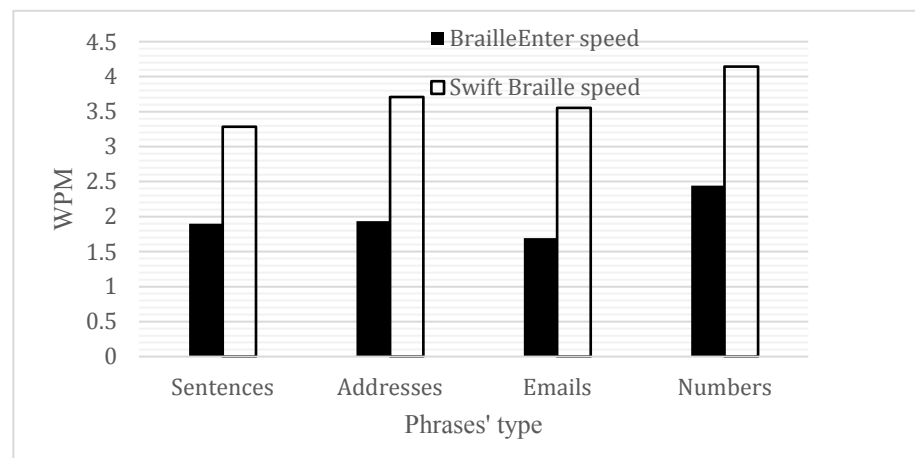
##### 4.6.5.1.1 Speed

Apparently, participants typed faster using the Swift Braille keyboard (mean= 3.67 WPM) than using the BrailleEnter keyboard (mean= 1.99 WPM). The result indicates a significant difference between the keyboards' speed (paired t-test;  $p = 0.007$ ). Figure 40 shows the mean typing speed of each participant for both keyboards.



**Figure 40:** Typing speed for both keyboards

##### 4.6.5.1.1.1 Keyboards' speed across text types



**Figure 41:** Average typing speed of both keyboards across text types

BrailleEnter speed was fastest when typing numbers (average = 2.44 WPM), over typing addresses (average= 1.94 WPM), Sentences (average= 1.89 WPM), and emails (average= 1.69 WPM) (see Figure 41). Similarly, Swift Braille was fastest when typing numbers (average= 4.15 WPM) over typing addresses (average= 3.7 WPM), sentences (average= 3.28 WPM), and emails (average= 3.55 WPM). Overall, typing all text types on Swift Braille keyboard was faster than the BrailleEnter keyboard (as shown in Table 11).

**Table 11: Average typing speed of both keyboards**

Keyboards' speed based on text types		
	<b>BrailleEnter</b>	<b>Swift Braille</b>
<b>Sentences</b>	1.898973602	3.285500812
<b>Addresses</b>	1.936343126	3.709764145
<b>Emails</b>	1.694503227	3.551991737
<b>Numbers</b>	2.441188684	4.145991337

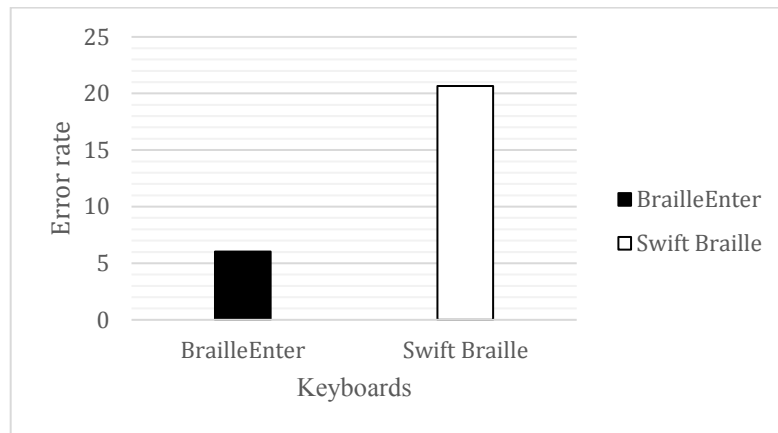
#### 4.6.5.2 Accuracy

##### 4.6.5.2.1 Error rate

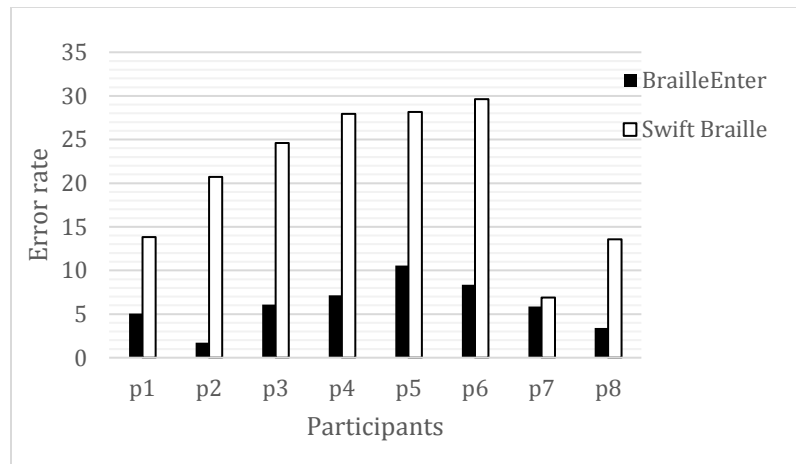
The Swift Braille Keyboard was the most prone to errors at 20.66% while the BrailleEnter keyboard had the lowest error rate at 6.03%. Figure 50 provides a summary of the error rates of both keyboards. The error rate was higher using the Swift Braille keyboard than the BrailleEnter keyboard. This indicates a significant difference in the error rate for the Swift Braille keyboard (Mean=20.66%, SD=8.37) and the BrailleEnter keyboard (Mean=6.03%, SD=2.63), paired t-test;  $p=0.00070$ ) (see Figure 42 and 43).

The high error rate for the Swift Braille keyboard was due to the inconsistency of the editing service; when users meant to delete (flicking from right to left) or to make space

(flicking from left to right) most often the keyboard inserted a new line. For example, when they wanted to make a space they by mistake touch dot 1 and 4 and inserted letter “c” as well as when they tried to do backspace they hit dot 2 accidentally and that inserts character “;”. Another reason is that when blind users intended to change the keyboard mode and they swipe vertically from top to bottom or vice versa, then they accidentally hit a dot, which caused an error.



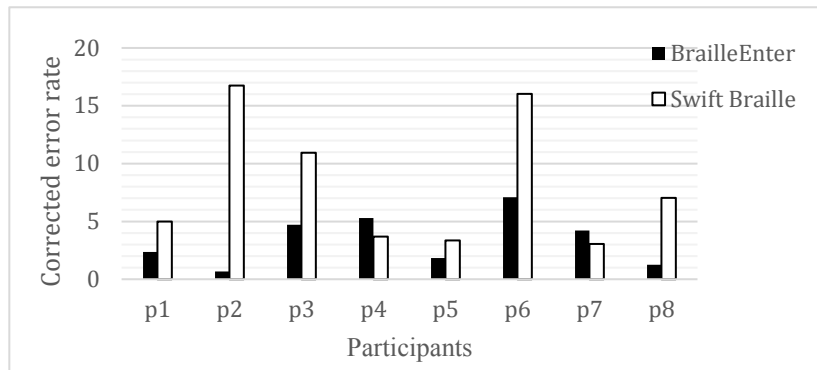
**Figure 42.** Average error rate of both keyboards



**Figure 43.** Average total error rate for each participant for BrailleEnter and Swift Braille keyboards

Overall, the error rate is higher for Swift Braille keyboard for each participant than BrailleEnter error rate.

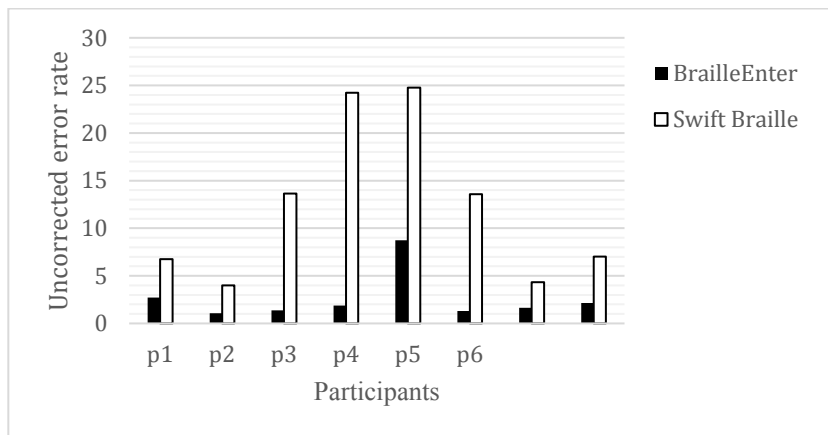
#### 4.6.5.2.2 Corrected error rate



**Figure 44.** Average corrected error rate for each participant for Braille Enter and Swift Braille keyboard

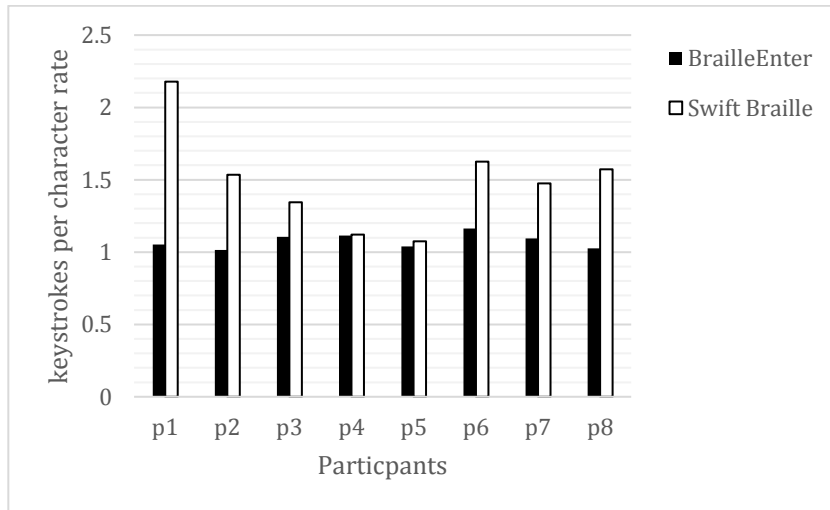
The overall corrected error rate for BrailleEnter was lower than the corrected error rate for Swift Braille keyboard. There was a significant difference in the corrected error rate for BrailleEnter (Mean=3.43%, SD=5.65) and Swift Braille keyboard (Mean=8.23%, SD=2.23), paired t-test;  $p=0.02$ ) (Figure 44).

#### 4.6.5.2.3 Uncorrected error rate



**Figure 45.** Average uncorrected error rate for each participant for Braille Enter and Swift Braille keyboard

This figure 45 shows the differences in NCER was also significant. There was also a significant difference in the NCER for the Swift Braille (Mean=17.39%, SD=8.39) and the BrailleEnter (Mean=3.4%, SD=2.53),  $p = 0.005$ .

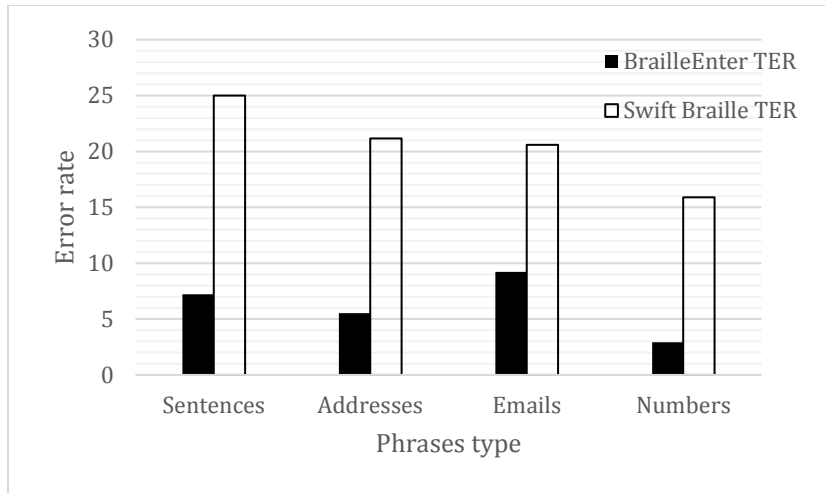


**Figure 46:** Average keystrokes per character rate for each participant for BrailleEnter and Swift Braille keyboard

Keystrokes per character (KSPC) also shows a significant difference in the KSPC for the Swift Braille keyboard (Mean =1.8, SD=0.34) and the BrailleEnter keyboard (Mean =1.3, SD=0.05), (paired t-test;  $p = 0.03$ ), indicating that BrailleEnter demands significantly fewer keystrokes per character than Swift Braille when typing text (see Figure 46). Swift Braille does not always respond correctly when users try to insert a space or delete thus it demands more keystrokes.

#### 4.6.5.2.4 Error rate across text types

In BrailleEnter, the error rate was lowest when typing numbers (average = 2.93%, over typing addresses (average= 5.53%), sentences (average= 7.23%) and emails (average= 9.23%).



**Figure 47.** Average of total error rate based on phrases type, and keyboards

Similarly, the error rate was lowest when typing numbers using the Swift Braille keyboard (average= 15.89%), over typing sentences (average= 25.01%), addresses (average= 21.15%), emails (average= 20.60%), (see Figure 47). Overall, typing numbers in both keyboards caused fewer errors than other types of text.

#### **4.6.5.2.5 Lewis' rating questionnaire results**

We calculated the mean of the nine items of the Lewis' rating questionnaire. In this questionnaire, the lower score indicates good status. This questionnaire provides more information about the reason why participants preferred one keyboard over another (see Table 12).

**Table 12: Results of Lewis' rating questionnaire**

<b>Statements</b>	<b>BrailleEnter</b>	<b>Swift Braille</b>
Easy to insert letters	2.67	2.67
Keyboard interaction method acceptable	2.17	3.17
Easy to type fast	5.17	4
Easy to type accurately	2.83	3.17
Easy to learn how to type letters	2.33	1.83
Using one finger is accessible	1.17	1.33
Audio feedback is accessible	1.17	1.5
Editing is accessible	1.8	5

The Lewis' rating for BrailleEnter slightly exceeded the rating for the Swift Braille keyboard. The mean rating was 2.41 for BrailleEnter, and 2.83 for Swift Braille keyboard. Table 12 shows the mean ratings of each statement in the questionnaire for both keyboards. Paired-tests result ( $p=0.19$ ) indicate that there was no significant difference in the Lewis questionnaire rating between the BrailleEnter and Swift Braille keyboards. Both keyboards had positive ratings for using one finger and audio feedback to interact with the touchscreen. However, The BrailleEnter keyboard had poorer ratings on the easy to type fast statement than the Swift Braille keyboard ( $p=0.22$ ). On the other hand, the Swift Braille keyboard had significantly poorer ratings on the editing is accessible statement than the BrailleEnter keyboard ( $p= 0.005$ ).



## **4.6.5.2 Qualitative results**

### **4.6.5.2.1 Pre-questionnaire**

All of the participants use the QWERTY keyboard with VoiceOver as the main input method on touchscreen devices. They mentioned some of the difficulties they have faced while using QWERTY with VoiceOver, including that it is hard for them to find the location of some letters and symbols and the QWERTY is not efficient because it causes errors. These difficulties are explained in detail in (Alnfai and Sampalli, 2016).

Most of the participants have also used voice detection in private places to insert messages, passwords, type notes and research online. However, in public places they used the QWERTY keyboard with VoiceOver to type text to overcome the voice dictation limitations, including that it does not detect their sound when they are in a noisy environment, and people around them are able to hear what is said. One of the participants reported that “when they pause Siri will not wait for them to complete the message”.

### **4.6.5.2.2 Interview**

#### **4.6.5.2.2.1 BrailleEnter Strengths and Limitations**

We developed approximately 10 qualitative codes to summarize the most relevant findings we learned from each participant, which we clustered into sets of high level themes, including difficulty typing the text, difficulty switching between keyboard layouts, audio feedback, annoying, convenient, easy to use, fast, error prone, physical comfort and navigation issues. The most important advantage of the BrailleEnter keyboard is allowing users to interact anywhere on a screen without looking for a specific location. Participant 1 stated that “Yes, I like it because you do not have to look around for a specific area of

the screen to type.” Participant 3 also agreed with the accessibility of typing anywhere on a screen by saying “I like it because I do not have to move all around to find a letter. I can type whenever was nice to be able to do that. If the Braille keyboard works better, it would be faster.” Participant 6 said that: “You do not have to be specific in what area of the screen you type in”.

All participants were able to perform tap and long tap gestures easily, even though they tended to hold too long to perform the long tap gesture. Participant 5 stated that “Tap and long tap gestures are easy to differentiate between. I do not know whether I hold too long or not I tried to speed up and it seems to work”. Participant 1 reported that “In the beginning I noticed I held too long on a screen more than what I need to, once you get used to the hold you can speed it a little bit.” Participant 3 stated that she has had difficulty with how long she should hold her finger over the screen to perform a long tap.

All participants liked using the swiping gesture to switch between keyboard modes. Participant 3 stated that “Swiping, I like that you can just swipe to get to number, symbols, letters, not to have to move around the screen too much and find things is nice.”

The main disadvantages of the BrailleEnter keyboard is that it increases the cognitive load on a user. Participant 2 reported that “it makes me think” (cognitive load). This limitation leads to slowness. Thus, other drawbacks with this keyboard is that it is not fast. The BrailleEnter keyboard had the slowest typing speed, due to participants having to think about which dots require long taps and which dots require short taps for each character. Some of the participants had not used Braille for some time, thus they spent a significant amount of time trying to remember the Braille code of a character. Two participants commented that BrailleEnter input method required too many interactions to

type on the screen.

#### **4.6.5.2.2.2 BrailleEnter suggestions**

Most of the participants recommended that we implement Grade 2 to improve the BrailleEnter speed. Grade 2 Braille is a shorten form of words that are represented by one Braille sign or two signs and to implement it would definitely speed up the keyboard. Some of the participants suggested we replace the long tap gesture with other gestures that require less time to perform.

Participant 1 suggested we implement this input method on Apple Watch to type Braille characters on a small screen to insert information on calendar, note, and calculators. P1 stated that “I could see it more practical on an Apple watch.”

#### **4.6.5.2.2.3 Swift Braille strengths and limitations**

The main strength of the Swift Braille keyboard is that it allows users to use one finger to interact with a screen. Participant 1 reported that “I think it would a good way for beginners to learn, “it’s kind of like drawing Braille letters.”

Another advantage is that the Braille dots are spread out on the screen and are located on the edge of the screen, which allow blind users to find the dot quickly. Another strength in this keyboard is that it uses vibration which informs users that they have touched a dot. Based on how many times they feel vibration; they can identify how many dots have been touched. Participant 3 stated that “Vibration was nice; you will know you hit the right button or the wrong button.” The main limitation of Swift Braille keyboard is that users have difficulty skipping a dot while trying to draw a letter that does not include dot 2 or 5.

For example, typing the letter ‘u’ requires users to connect dots 1, 3 and 6; they often hit dot 2 by mistake because they cannot be precise about the exact area or space that they can move their fingers without hitting unwanted dots. Participant 2 stated that “you have to make sure when you are going in between for letters (u, m, x) because you are tracing those letters. Someone with trouble with coordination might hit other buttons.”

Editing in the Swift Braille keyboard was really difficult because when they performed delete, which is flicking from left to right, they hit other buttons on the screen and accidentally inserted a character. Making space was also a drawback on this keyboard, usually when participants made a space by flicking from right to left, they hit the wrong dot or sometimes created a new line. P2 stated that “I have a little difficulty because of the space, new line and delete.” Participant 5 reported skipping dots and making spacing are difficult to do in this keyboard. You have to be precise when you move from dot to the another”. Another drawback is that accidently touching other buttons on the Swift Braille interface might open up another screen and they have hard time to return back to the keyboard layout. P6 stated that “Blind people have to run their fingers over the screen, they cannot just touch”. This means blind people need to be able to move their fingers around the screen first to identify the location of keys and then when they find the wanted key they can perform a specific action to selected it. Thus, they made a lot of errors on the Swift Braille keyboard because each time they touch the screen they insert a dot or enter a command.

#### **4.6.5.2.2.4 Swift Braille suggestions**

Some of the participants suggested we add two buttons, one to add space and the second

one to delete. The main objective of adding two buttons on the Swift Braille interface is to avoid hitting other buttons on a screen and making it do something else. Participant 5 also suggested that providing instructions in the beginning, adding correction features, spell check and implementation of Grade 2 Braille.

#### **4.6.5.2.2.5 Observation**

We tried to understand how blind users interact with both input methods. All participants stated that interacting with touchscreen using one finger is the most accessible way for blind people. Participant 1 stated that “I can hold the phone with one hand and just use one finger to type.” and Participant 2 stated that “it is much easier to navigate around a screen using one finger”. When users use the BrailleEnter keyboard, some of the users tapped on one spot of a screen and did not change their finger position. Some of them tap as the Braille shape which is 3\*2 matrix. P1 and P4 typed at a speed that was slightly faster than other participants. This might be because they tapped on one area of the screen, reducing the movement of the finger while interacting with the screen to insert text. Other participants took more time when they moved their finger from one position to another while tapping the six dots.

Regarding the Swift Braille keyboard, we noticed that users were trying to move their finger slowly when they were trying to skip the middle dot to avoid hitting a dot by mistake. We also noticed that they connected the Braille dots in order. For example, letter x is the connection of the dots 1, 3, 4 and 6, so some of the participants connected these dots in the same order, but some of them when they connected the dots they pick the shortest way. For example, they connected 1, 4, 5 and 6, this path is easier because they

only came from the middle once despite that following the order requires the users to come from the middle between dots two times, which might cause more typing errors.

#### **4.6.5.2.2.6 Preference**

At the end of the evaluation study, participants were asked which keyboard they preferred and the reason they preferred one keyboard over the other. Six out of eight participants preferred the BrailleEnter keyboard over the Swift Braille keyboard. The other 2 participants preferred the Swift Braille (P2, P5).

Some of the participants stated the reason why they preferred a BrailleEnter keyboard over the Swift Braille keyboard. Participant 3, who preferred BrailleEnter, addressed the reason for disliking Swift Braille by saying that the keyboard consistently behaved “differently than what I expect.” Participant 3 stated that it is really easy to delete using BrailleEnter”. Participant 4 stated that “I did not like [Swift Braille as much as the BrailleEnter, it was not quite as easy to delete space. I found flicking up and down was not as easy as the BrailleEnter.”

The main reason Swift Braille was not preferable is that it required users to find the specific location of keys on a screen. Participant 1 said that with Swift Braille he sometimes “found the gestures were inconsistent when he changes the keyboard mode”. Participant 6 also reported that “This one here you had to be a little more precise as to what area of the screen you are on when connecting the dots. Sometimes, another screen comes up, which is really confusing.”

#### **4.6.6 Discussion**

We compared the BrailleEnter keyboard, a new input approach with the Swift Braille keyboard in terms of speed, accuracy, usability and preference. In this study, we also discussed the keyboards' strengths, drawbacks and suggestions for improvements. Based on the analyzed data, we highlighted the critical points for designing an accessible keyboard for blind people.

##### **4.6.6.1 Speed and accuracy**

This experiment found the WPM of Swift Braille is much higher than the BrailleEnter. Swift Braille was expected to have the same speed as BrailleEnter. Swift Braille requires one swipe to connect between Braille dots of each character. However, based on our observation, any time participants touched the Swift Braille interface, they inserted a character, space or a new line even if they did not intend to do that. Sometimes, they did not even finish connecting Braille dots of a character or they touched the screen unintentionally and they inserted a wrong character.

BrailleEnter keyboard was slower than the Swift Braille keyboard because it requires users to interact six times with a screen to type a character. In addition to that, we observed participants hold their fingers on the screen to perform the long tap gesture longer than they should. That might be considered a reason why they took longer to type text on this keyboard. The study results also show that there was a significant difference in both keyboards' speed across four text types. It shows that both keyboards were fast when users typed numbers. This is because numbers have fewer active dots in Braille coding. Most of the numbers have fewer than three active dots in each Braille number code.

The error rate was significantly higher in the Swift Braille than the BrailleEnter keyboard. Analyzing the error rate of each keyboard revealed three main reasons that make Swift Braille more error prone. The first reason is that which some of the participants mentioned; they made more typing mistakes on Swift Braille when they tried to make spaces, delete or switch between keyboard modes. They found using swiping gesture inconsistent and confusing because when they made a space they typed letter “c” or number “3” and when they deleted (backspace) they typed “;”. The second reason is that typing some characters on Swift Braille is difficult, such as ‘u’, ‘m’, and ‘x’. This was exposed in the interviews with participants and from the video analysis. Some of the participants mentioned that “it is hard to skip dots while connecting between dots”. The third reason is that searching for a dot while using Swift Braille requires a continuous touching of the screen to explore the interface. This touching gesture negatively affected the Swift Braille keyboard accuracy because it makes the keyboards insert unwanted characters, which then often either required the user to correct the typing error or start searching again for the target dots. Even when participants were trying to correct their mistakes, they inserted another error by hitting other dots while flicking from right to left.

The result indicated that BrailleEnter causes less typing errors, perhaps because the BrailleEnter keyboard does not require users to be precise while navigating on the screen. It allows them to interact in any accessible part of the screen. Other reason is that it only uses gestures to perform all the keyboard functions without presenting any button on the screen. One participant mentioned that the feeling of typing on BrailleEnter is slightly different from typing on Perkins Braille, in Perkins users have to insert only the active dots, but in BrailleEnter they have to insert both active and inactive dots as a filler. That creates



cognitive load while inserting dots. This might be a reason why BrailleEnter's speed is slow. Furthermore, the study results show that there was a significant difference in both keyboards' accuracy across the four text types.

Overall, the Swift Braille keyboard was the fastest, but it was the most error prone. The study also showed that while BrailleEnter is the most accurate text entry, it is the slowest input method. The study also examined the speed and accuracy of both input methods across four text types including sentences, addresses, emails and numbers. Overall, the result indicates that typing numbers on Braille requires less effort because they have fewer numbers of active Braille dots. However, the Swift Braille input method exceeded BrailleEnter in text entry speed in all four text types. On the other hand, the error rate was lowest in BrailleEnter. The error rate was lowest while typing numbers over other types of text.

#### **4.6.6.2 Usability and preference**

When participants were instructed about which keyboard they prefer most, the result showed that BrailleEnter was the most preferred. Overall, BrailleEnter was more usable and preferable than the Swift Braille keyboard. The Lewis questionnaire revealed the main points that participants prefer a keyboard over the other. Subjective ratings of the two Braille keyboards show reliable differences between the keyboards' features and highlights the main strengths and limitations in both keyboards. The essential advantages on both keyboards were allowing blind people to use index finger to interact with a screen to type as well as providing an audio feedback feature. Though, features that are not accessible in the Swift Braille keyboard are its editing service including entering a space, backspace and

switching between keyboards. Most participants were frustrated when they did something and got unexpected outcome. However, vibration and locating the Braille dots' keys on the screen edges improve the interaction with the keyboard interface. Regarding BrailleEnter, interacting six times with a screen to insert a letter was not preferred by participants as it significantly reduces the keyboard speed. However, typing anywhere on a screen without boundaries was the most preferable feature of the BrailleEnter keyboard. The other features participants liked was swiping horizontally to switch between keyboards and vertically to edit were accurate and easy to perform. Overall, Table 13 shows the general result about both keyboards.

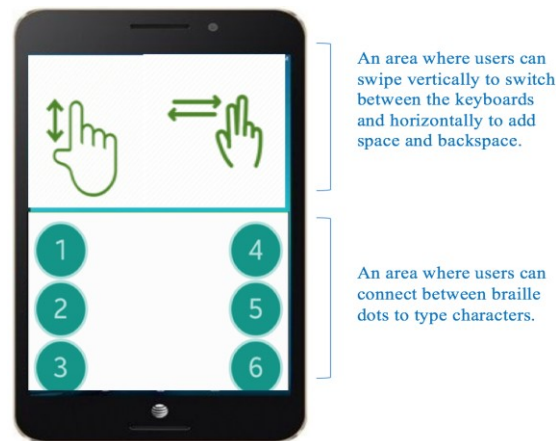
**Table 13. Result summary**

<b>Keyboard</b>	<b>Evaluation matrix</b>			
	<b>Efficiency (speed)</b>	<b>Effectiveness (accuracy)</b>	<b>Usability (Lewis ratings)</b>	<b>Preference (Number of participants)</b>
<b>BrailleEnter</b>	<b>1.99 WPM</b>	<b>6.03%</b>	<b>2.41</b>	<b>6</b>
<b>Swift Braille</b>	<b>3.67 WPM</b>	<b>20.66%</b>	<b>2.83</b>	<b>2</b>

Based on BrailleEnter's speed, we can say that BrailleEnter is a valuable keyboard for short text like passwords, and social media posts. It also can be a well-designed keyboard for inserting passwords to login website applications, or enter bank information. Moreover, we believe that the BrailleEnter keyboard might be a valid keyboard for small smart devices like smart watches.

The study also indicated that the Swift Braille keyboard could be improved by eliminating the limitations that have been mentioned above. To overcome the Swift Braille limitations, we suggest that making the six dots near to each other and locating them on

the bottom of the screen to make an unfilled space and the empty space of the top of the screen can be used to enable users to perform swiping vertically to delete or add space and horizontally to switch between keyboard modes (As shown in Figure 48).



**Figure 48.** Suggestion for improvements of Switch Braille keyboard

Mattheiss et al. also suggested that using unused Braille combination of dots to implement editing features including delete and space (Mattheiss et al., 2015). However, this suggestion will decrease learnability because users have to learn the combination of dots that represent delete or space functions.

The most important design principles of developing an accessible input method for blind people have been extracted from our evaluation study of two Braille keyboards.

1. Avoid using both buttons and gestures on one layout to prevent users from making typing mistakes. Use both buttons and gestures on one layout that causes a lot of errors. When a blind user performs a gesture for any function, they may be unable to avoid hitting other buttons on a screen or they might not know exactly where buttons are located on a screen, thus they perform a gesture anywhere which causes errors.

2. Eliminate using buttons on the application interface that are used to perform specific functions., Instead try to use gestures that are accessible by blind people. For example, in the Swift Braille keyboard, there are some buttons on the interface that are used to allow users to copy, add a new line or remove the whole word, but users hit these buttons accidentally and make an error.
3. Use vibrations to help users get feedback while interacting with a screen.
4. Shrink the distance between the Braille dots to minimize the time searching for other dots on a screen.
5. Provide a clear, quick and short feedback mechanism for users to inform them what is going on a screen.
6. Implement Grade 2, word prediction, auto corrections, and sentences library to enhance blind users' performance on touchscreen keyboards.

#### **4.6.6.3 Comparative analysis with other input methods for visually impaired**

The quantitative and qualitative data collected from previously published papers to display an overview about the performance and accessibility of most existing input methods (see Table 14 and 15) (Lee, 2004, Azenkot et al., 2014; Azenkot et al., 2012; BrailleEasy, 2018; Frey et al., 2011, Guerreiro et al., 2008; Heni et al., 2016; Jalaliniya et al., 2015; Keeble, 2018; MBraille, 2018; Mattheiss et al., 2015; Mascetti at al., 2011; Oliveira et al., 2011; Orf, 2015; Siqueira *et al.*, 2016; Sepic et al., 2015; Shabnam, Govindarajan, 2014; Southern et al., 2012; Subash et al., 2012; Soft Braille, 2018; Bonner et al., 2010; Alnfiai, and Sampalli, 2016a; Alnfiai, and Sampalli, 2017b; Alnfiai, and Sampalli, 2017c).

**Table 14. An overview of input method performance**

Keyboards	Types	Interaction technique	Participants' Number	Speed (WPM)	Error rate
Keeble	SW	Location based approach	-	-	-
Qwerty keyboard	SW		7	4.71	20.54
BrailleKey	SW		5	1.80	5.60
BrailleSkitch	SW		10	14.35	10.6
BrailleType	SW		15	1.45	9.7
Swift Braille	SW		8	3.67	20.66
EdgeBraille	SW		14	3.97	8.43
No-Look-Notes	SW		10	1.67	11
Eyedroid	SW		12	-	-
BrailleTouch	SW		Multi-touch approach	11	18.62
MBraille	SW	-		-	-
BrailleEasy	SW	6		7.03	11.75
iOS Braille	SW	-		-	-
Super Braille	SW	-		-	-
Perkinput	SW	8		6.1	15.75
Soft Braille	SW	-		-	-
GBraille	SW	-		-	-
TypeinBraille	SW	10		3.6	6.5
SingleTapBraille	SW	Single Touch approach	7	3.72	11.23
BrailleEnter	SW		8	1.99	6.03
MoonTouch	SW+HW	Tactile layout	6	10.14	5
BrailleÉcran	SW+HW		10	2.52	2.86
H-Slate	SW+HW		6	-	-
Korean Perkins Brailier	HW	Physical keyboard	-	27.9	2.8%
Portuguese Perkins	HW		-	16.7	-

**Table 15. An overview of input methods' Qualitative features**

	Interaction approach	Is easy to learn	Is easy to use	Requires one hand	Requires one finger to interact with a screen	Uses VoiceOver service	Uses tactile feedback	Eliminates switching between layers	Eliminates looking for a specific location	Eliminates interacting based on particular coordination	Supports number, alphabets, punctuation	Allows editing	Integrates the keyboard with an application	
Keeble	Location based approach	√	√	√	√	√	X	X	X	X	√	√	√	
Qwerty		√	√	√	√	√	X	X	X	X	√	√	√	
BrailleKey		√	X	X	X	√	X	√	X	X	√	√	X	
BrailleSkitch		√	√	√	√	√	X	√	X	X	X	√	X	
BrailleType		√	√	√	√	√	X	√	X	X	√	X	X	
Swift Braille		√	√	√	√	√	X	X	X	X	√	√	√	
EdgeBraille		√	√	√	√	√	X	√	X	X	X	X	X	
No-LookNotes		√	√	√	√	√	X	X	X	X	X	X	X	
Eyedroid		X	√	√	√	√	X	√	√	√	X	X	X	
BrailleTouch		√	X	X	X	√	X	√	X	X	X	X	X	
MBraille	Multi-touch approach	√	X	X	X	√	X	√	X	X	√	√	X	
BrailleEasy		√	√	X	X	√	X	√	X	X	√	X	X	
iOS Braille		√	√	X	X	√	X	√	√	√	√	√	√	
Super Braille		√	√	X	X	√	X	√	X	X	√	X	X	
Perkinput		√	√	X	X	√	X	√	X	X	√	X	X	
Soft Braille		√	X	X	X	√	X	√	X	X	√	√	√	
GBraille		X	√	X	X	√	X	√	X	X	√	√	X	
TypeinBraille		X	√	X	X	√	X	√	X	X	√	√	X	
SingleTap Braille		Single touch	√	√	√	√	√	X	√	√	X	√	√	√
BrailleEnter			√	√	√	√	√	X	√	√	√	√	√	√
MoonTouch	Tactile	X	√	√	√	√	√	X	X	√	√	X	X	
BrailleÉcran		√	√	√	√	√	√	X	X	√	√	X	X	
H-Slate		√	√	√	√	√	√	X	X	√	√	X	X	

Tables 14 and 15 show user performance and qualitative features of existing text input methods that have been developed for visually impaired people. These keyboards used different approaches including location based, multi-touch, single-touch and tactile interface. Each approach has its unique strengths and weaknesses. The multi-touch approach allows blind users to type fast. However, there is a high rate of error. Due to the small size of the smartphone device, the blind users' hands tend to drift across the touchscreen over time. Another requirement of the multi-touch entry method that makes this approach inaccessible for blind users is that it is not practical to use two hands to type when one hand is occupied while holding the device.

Adding a tactile overlay on a touchscreen improves the accessibility for blind users as it provides tangible interaction. Tactile feedback allows blind people to type fast with a lower error rate. However, tactile overlay limits the applications' functionality or requires extensive customization of the hardware where it is very expensive and complex to prepare as it requires intensive labour (He et al., 2017).

Keyboards that are designed based on location are time consuming and cause too many errors. This type of keyboards represents the keyboard layout visually and requires users to find the exact location of a key. This requires the users to spend a long time to find the desired key. Beginners need to spend more time and effort to map the keyboard layout and memorize the location of each key. It is also error prone because the users might accidentally hit the wrong key or link on the screen, in which case another screen pops up or deletes the inserted data.

Input methods that use a single touch approach are still slow and produce errors, but this type of keyboards improve the accessibility where the users can type using only

one finger and they can type anywhere on the screen without any concern about hitting unwanted buttons. They also do not require the users to spend time to learn how to operate the keyboards as they are familiar with Braille, their mother language. Other multi-touch keyboards require users to learn the keyboards' interaction techniques and map and memorize the keyboard layout. Our method enables users to switch between keyboard layouts without the need to locate a particular button on the screen; users tap the Braille sign for the desired layout. These keyboards also can be easily integrated with other applications like SMS applications or any social media application. This is novel because other approaches like tactile or multi-touch require developers to take into account how to manage and organize the application buttons so users do not accidentally press them.

#### **4.7 CONCLUSIONS**

Because keyboards based on Braille patterns provide an accessible input method on touch screen devices, we tried to develop a text entry method that makes smartphones more accessible for all populations, including blind people. We have described novel non-visual text entry methods, SingleTapBraille and BrailleEnter, that do not require users to locate a particular object on a touch screen; instead, text can be inserted by touching on a touch screen based on Braille shapes. The SingleTapBraille and BrailleEnter keyboards were evaluated in the CNIB, enabling us not just to quantify how blind users interact with touch screens but also to understand specific aspects of phone use, like how they hold devices to use them while walking. Our evaluation also enabled us to identify the advantages and disadvantages of both input methods and to obtain feedback for improvements.



The main contribution of our research is that we have used Android mobile devices to build accessible keyboards to help blind people overcome some of the difficulties they face while typing on a touch screen using the standard keyboard (QWERTY keyboard) or a button based keyboard (Swift Braille). The evaluation provided promising feedback and also provided some suggested improvement for further development. It also showed that using a thumb to interact with a touchscreen and holding the smartphone using only one hand is not a common technique for blind users. Indeed, the ability to use this system with only one hand is a critical advantage over other keyboards, especially for a population that often has to use one hand for a cane or guide dog. Our results indicate that entering characters based on Braille significantly enhances accessibility and usability, and has the potential to be a valuable text entry tool for blind users.

In the future, there are several opportunities for improvement to the proposed keyboards. Firstly, because seven to eight is a relatively small sample size, future work will test the keyboards with larger groups of people in order to better understand their strengths and weaknesses. We are also planning to compare our keyboards with a multi-touch keyboard. Future work will also increase the keyboards' usefulness by integrating them with existing communicating apps and implementing it with wearable technology such as the Apple Watch to make it even more convenient for blind people to text short messages while walking. Further, we will try to find a way in which blind users can use the predictive text feature with Braille keyboards. Lastly, we will try to increase security and privacy through a keyboard based entirely on Braille. Other possible future directions include implementation of efficient error correction interfaces that minimize time needed to edit typing errors.

## **CHAPTER 5: NUMBER INPUT METHOD**

### **5.1 The Design of the BrailleTap Calculator**

Our main goal is to design an accessible calculator on a touchscreen device that allows blind users to tap Braille numbers anywhere on a touchscreen using their thumb or one finger without the need to find a particular number or operation button on a calculator layout. In our proposed calculator, users are not required to navigate the calculator keys.

The calculator is designed based on Braille codes. As mentioned above, the code has two columns and three rows. Blind users generally enter Braille dots from left to right and from one to six. Our calculator follows the same principle, allowing users to insert Braille dots from left to right and from one to six. Users can activate Braille dots individually using their thumb or a finger to tap on the screen. In order to activate the first Braille dot, users can tap once anywhere on the screen and this will represent the number “1”. In case the user wants to activate more than one dot, he/she can tap the screen quickly to create Braille shapes. Once the user stops tapping on the screen for a short time, our algorithm will interpret the Braille pattern and the corresponding letter. At the same time, a Text-to-Speech function will read out the resulting number. Our approach allows users to hold the smartphone device using one hand while they interact with the touchscreen using their thumb to tap Braille dots anywhere on the screen. This follows Paisios (2012), who found that the most accessible way for blind users to interact with a touchscreen is by using one finger.

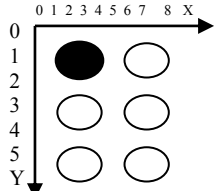
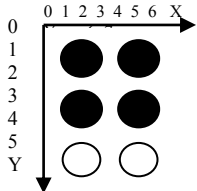
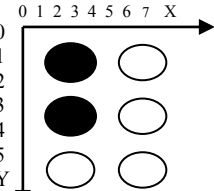
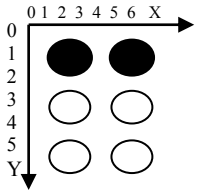
In order to determine the corresponding number when users tap Braille dots on a screen, our algorithm analyzes the relationship between all activated dots in each Braille

number based on the active dots' coordinates. For instance, the number two will be represented when a user activates the first and second dots on the touchscreen. Our algorithm is mainly based on the clustering of active dots for each number and identifying the relationship between those active dots. The algorithm clusters all the inserted taps on a touchscreen and analyzes all these taps based on the number of taps, the shape of these taps, as well as each tap's coordinates. When the user stops tapping on the screen for a short interval, the taps are analyzed and the number matched. Table 16 and Figure 49 represent the algorithms that have been used to examine the relationship between dots.

The factors that indicate the relationship between Braille dots are:

- a. The value of coordinates (X, Y) for each tap on the screen
- b. The number of dots in each number in Braille code

**Table 16. Categorization of Braille numbers based on number of dots**

a. Character that has one dot	e. Characters that have four dots
<p><b>Number 1</b></p>  <p><b>One tap anywhere in a mobile screen</b></p>	<p><b>Number 7</b></p>  <p><math> X1-X2 &lt;e;  X3-X4 &lt;e; X2&lt;X3;  Y1-Y3 &lt;e;  Y2-Y4 &lt;e; Y3&lt;Y2</math></p>
b. Characters that have two dots	
<p><b>Number 2</b></p>  <p><math> X1-X2 &lt;e</math></p>	<p><b>Number 3</b></p>  <p><math> Y1-Y2 &lt;e</math></p>

<p><b>Number 5</b></p> <p><math>X1-X2&lt;0; Y1-Y2&lt;0</math></p>	<p><b>Number 9</b></p> <p><math>X1-X2&gt;0; Y1-Y2&lt;0</math></p>
<p><b>c. Characters that have three dots</b></p>	
<p><b>Number 4</b></p> <p><math>Y1-Y2=0; X2-X3=0</math></p>	<p><b>Number 6</b></p> <p><math>X1-X2=0; Y1-Y3=0</math></p>
<p><b>d. Characters that have three dots</b></p>	
<p><b>Number 8</b></p> <p><math>X1-X2=0; Y2-Y3=0</math></p>	<p><b>Number 0</b></p> <p><math>X1-X2&lt;0; X2-X3=0; Y1-Y3=0</math></p>

As seen in table above, each Braille dot's position is reflected by both an x and y coordinates. These coordinates are used by the algorithm to create an equation that reflects the relationship between the position of the dots in each Braille code. These equations are represented below each Braille code in the table above. Once the pattern of Braille code is recognized the algorithm will present the corresponding number.

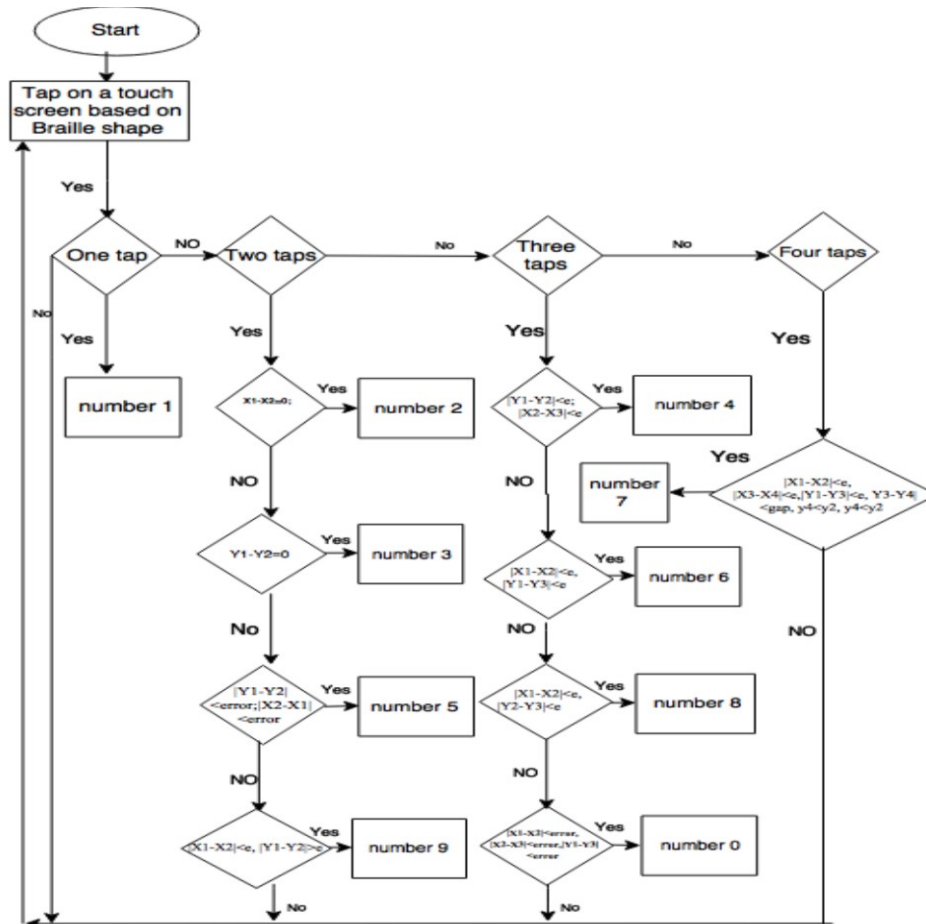


Figure 49. Algorithm flowchart

The error variable “e” is a defined value that is used as a threshold for whether two floating numbers are in a vertical or horizontal line. If the difference between the x or y values of two taps is smaller than the error value, the algorithm will accept it because it is difficult for a user to insert two taps in an exact vertical or horizontal line.

### 5.1.1 Representing Braille numbers using BrailleTap

BrailleTap allows operations to be entered in a way that is accessible and effective using swiping gestures with a single thumb. This is in accordance with Paisios and Richard et al., who found that using swiping and tapping to interact with a touchscreen is the easiest

gesture and most accessible methods for blind users (2012). In order to insert an operation, users can swipe vertically in a loop to activate a particular operation. The first swipe will activate the first operation and TalkBack service will speak the operation name. Then, if the users swipe vertically twice, they will obtain the second operation. For example, when the user performs the first vertical swipe from bottom to top, the app will say “addition”. In case the user does not want the addition operation, he/she can swipe again from bottom to top to select subtraction instead of addition. Performing a vertical swipe from top to bottom will insert division operation. A second swipe from top to bottom will insert multiplication instead of division. They can also swipe horizontally to edit their typing (see Table 17). After the users enter the mathematics equation, they can perform a long press gesture to obtain the result.

**Table 17. Braille keyboard interaction techniques and their purposes.**

<b>Interaction techniques</b>	<b>purposes</b>
Swipe from bottom to top	Addition, subtraction
Swipe from top to bottom	Division, multiplication
Swipe from left to right	Clear operation
Swipe from right to left	Backspace
Long press	equal

## **5.2 BrailleTap calculator Implementation**

In this section, we describe the implementation of the BrailleTap calculator to allow blind users perform calculation easily. We will describe the function and interfaces of BrailleTap calculator.

### **5.2.1 User interaction module**

This section presents the BrailleTap architecture of the user interaction module, which includes:

- 1) Gesture pattern recognition module
- 2) Braille number pattern recognition module
- 3) Switching between operations
- 4) Editing module
- 5) Text to speech module

To enter a number on a touchscreen using the BrailleTap, one to four taps are required to recognize a number. The BrailleTap calculator algorithm contains five modules detecting the gesture pattern, recognizing Braille numbers, switching between operations, editing module includes gestures, and Text-to-Speech module.

### **5.2.2 Gesture Recognition Algorithm**

This section explains the gestures used to implement the BrailleTap calculator.

#### **5.2.2.1 Explanation of Gestures**

The accessible gestures that have been used to implement the BrailleTap calculator are:

1. Short Taps: to form the Braille patterns of each number.
2. Long Tap: to get the result of the entered equation “=”.
3. Swiping horizontally: to edit
  - Swipe right: delete on digit
  - Swipe left: clear all
4. Swiping vertically: to switch between operations including (+, -, \*, /)

The algorithm is implemented in Java language in an Android studio platform. The corresponding output of the entered inputs is explained in tabulated from Table 1. The implemented algorithm is designed mainly based on the interaction techniques that users made on the mobile phone screen.

The input values are one to four of tapping gestures that are taken from the smartphone screen. The algorithm produces expected outputs (numbers) based on the relationship between the entered taps and their orders. The gesture swipe vertically and horizontally is used to trigger one action (see Figure 50 and 51).

## 1. Gesture recognition module

---

**/\* Gesture Recognition Algorithm \*/**

---

**Comment:** The gesture used is swipping and performed by end user

**Comment:** Enter the type of gesture from the list (Swipe right, Swipe Left, Swipe up, Swipe down)

SWIPETHRESHOLD 100px. // required minimum distance traveled to be considered swipe

**Step1. Read** input details from the touch screen including events where the touch position changed using MotionEvent

**Step2. Detect** gestures' type

On **ActionUp** event: read Xup, Yup coordinates values

On **ActionDown** event: read Xdown, Ydown coordinates values

**Step 3. If**  $|Xup - Xdown| > SWIPETHRESHOLD$  **then**

**return** "Swipe Horizontly"

**Step 4. If**  $(Xup > Xdown)$  **then**

**return** "Swipe right"

**do** "Clear"

**end if**

**Step 5. If**  $(Xup < Xdown)$  **then**

**return** "Swipe left"

**do** "Delete one digit"

**end if**

**end if**

**Step 6. If**  $|Yup - Ydown| > SWIPETHRESHOLD$  **then**

**return** "Swipe Vertically"

**Step 7. If**  $(Ydown < Yup)$  **then**

**return** "Swipe down"

---



---

```

do "Switch between operations"
end if
Step 8. If (Xdown > Yup) then
return "Swipe up"
do "Switch between operations"
end if
end if

```

---

Figure 50. Gesture recognition algorithm

```

//Horizontal Swipe Detection
if (Math.abs(deltaX) > min_distance) {
swipe = true;
if (deltaX < 0) { //Left to right
clear(); //Clear all
return false;
}
else if (deltaX > 0){ //Right to left
backSpace(); //Delete a character
return false;
}
}
//Vertical Swipe Detection
else if (Math.abs(deltaY) > min_distance){
swipe = true;
if (deltaY < 0){ //Top to bottom
if (mode == 0){
mode = 3; //Wrap Around
}
else {
mode--;
}
}
else if (deltaY > 0){ //Bottom to top
if (mode == 3){
mode = 0; //Wrap Around
}
else {
mode++;
}
}
}

```

Figure 51. Code of gesture recognition algorithm

## 2. Braille number pattern recognition module

The input variables are the touching events that users perform on the screen. This algorithm requires one to four inputs to produce a number. It processes these touches or gestures (input variables) by analyzing the number of touches that are being performed on the

screen, the position of these touches (the shape or the relationship between taps by determining the coordinates of each tap), the gestures' type, and order. The active Braille dots are represented by short tap gestures and the inactive dots are ignored in this algorithm (see Figure 52 and 53).

---

**/\* Number Recognition Algorithm \*/**

---

Void **NumberRecognition** ()

**Comment:** The gesture used is tapping and performed by end user

**Comment:** variable is error50 //e=50

**Step1. Interact** with the screen by **tapping on** the screen

**Step2: Start** the time count once a user touches the screen

//Timer == 0 and the size of array list is zero (Taps' list[]). // Time handler

**Step3. Detect** the gesture type (*Gesture recognition algorithm*)

**Step4. Add** touch gestures on a screen, its coordinates, order into the array list to the array list Taps' list [] and the number of taps that have been performed on the screen in the specified time.

**Step5. DO step 1, 2 and 4 as many as the number of active dots in the Braille code of the entered number** (representing the active dots of Braille code) in 3 seconds.

**Step6. Stop** the timer == 3 seconds

**Step7. If** (array list has more than 4 gestures)

**return** "more than 4 dots"

**speak** "could not recognize number"

**End if**

**Step8. Else if** (array list has 4 taps or less)

**Switch** (the number of taps stored in the array list)

**case 1:** // (one tap)

**Enter** "number 1"

**break;**

**case 2:** (Two Taps)

**If** ((|X1-X2| < error))

**Enter** "Number 2"

**End if**

**else if** (|Y1-Y2|<error)

**Enter** "Number 3"

**End if**

**Else if** ((X1-X2<0) &&(Y1-Y2<0))

**Enter** "number 5"

**End if**

**else if** ((X1-X2) <0) && ((Y1-Y2)>0)

**Enter** "number 9"

**else**

**Speak** "Could not recognize number"

---

---

```

    break;
    case 3: (Three Taps)
    If ((|Y1-Y2| < error) && (|X2-X3| < error))
        Enter "number 4"
    else If ((|X1-X2| < error) && (|Y1-X3| < error))
        Enter "number 6"
    else If ((|X1-X2| < error) && (|Y1-X3| < error) &&(|Y1-Y2|))
        Enter "number 8"
    else If ((X1-X2) < 0 && (|X2-X3| < error) && (|Y1-Y3| < error))
        Enter "number 0"
    case 4: (Four Taps)
    if ((|X1-X2|<error)&& (X3-X4<error)&&(Y1-Y3<error)&&
        (X2<X3)&&(Y3<Y2))
        Enter "number 7"
    Break;
    default:
        Speak("Couldn't recognize number");
        break;
    end

```

**Step9.** Clear the array list  
**Step10.** Repeat step 1 to 9 until users enter the wanted numbers  
**Step 11.** End Number Recognition Algorithm

---

**Figure 52.** Number recognition algorithm

```

//check the size of list
switch(list.size()){
    case 1:
        writeText('1');
        break;
    case 2:
        raw_dx = list.get(0).x - list.get(1).x;
        raw_dy = list.get(0).y - list.get(1).y;

        xdif = Math.abs(list.get(0).x - list.get(1).x);
        ydif = Math.abs(list.get(0).y - list.get(1).y);

        //check 2
        if ((xdif < error)){
            writeText('2');
        }
        //check 3
        else if (ydif < error){
            writeText('3');
        }
        //check 5
        else if ((raw_dx < 0)&&(raw_dy < 0)){
            writeText('5');
        }
        //check 9
        else if ((raw_dx < 0)&&(raw_dy > 0)){
            writeText('9');
        }
        else{
            Speak("Couldn't recognise letter");
        }
        break;
}

```

**Figure 53:** Snapshot of number recognition algorithm code

### 3. Switching between operations module

---

#### Change the calculator operations (+,-,/,\*)

---

**Step 1. Check if the user performs swiped Vertically on the screen**

**Step 2. If (YDown < Yup) is valid then**

**If** the calculator operation valid on multiplication mode // (m==3)

**Change** it to summation operation (m == 0)

**else**

**Increase** the calculator mode value to wrap around the calculator layouts (m++)

**Switch** (keyboard mode)

**Case 0:** addition operation (m==0)

**Break;**

**Case 1:** subtraction operation (m==1)

**Break;**

**Case 2:** division (m==2)

**Break;**

**Case 3:** multiplication (m==3)

**Break;**

---

**Figure 54.** Change the calculator operations

```
switch (mode){
    case 0:
        writeText('+');
        Speak("addition");

        break;
    case 1:
        writeText('-');
        Speak("subtraction");
        break;
    case 2:
        writeText('/');
        Speak("Division");
        break;
    case 3:
        writeText('*');
        Speak("Multiplication");
        break;
}
```

**Figure 55.** Snapshot of switching between calculator operations code

## 4. Editing module

This module has two methods, which are backspace and clear.

### 1. Delete method

Delete method (Backspace) has been explained in Chapter 4.

### 2. Clear method

Clear function removes the contents of the edit text (see Figure 56 and 57).

---

#### Clear function

---

Void clear ()

X is the entered equation on the screen

**Return** ' '; // **remove all** the entered equation X

**Speak** "clear";

**End clear method**

---

**Figure 56.** Clear method

```
private void clear(){
    Speak("clear");
    final char chx = ' ';
    final String x = txtText.getText().toString();
    txtText.setText(chx);
    txtText.post(new Runnable() {
        @Override
        public void run() { txtText.setSelection((x + chx).length()); }
    });
}
```

**Figure 57.** Snapshot of clear method code

## 5. Text to speech module

The Text-to-speech function has been implemented in this calculator to enable blind users to hear the entered numbers, operations as well as results. This function has been explained in detail in Chapter 4.

### 5.2.1 BrailleTap interface

The BrailleTap interface is based mainly on audio feedback and does not use any buttons

on the screen. Users can tap and perform gestures anywhere on the screen (as shown in Figure 58). The TalkBack accessibility service in Android platforms reads out the written text on the screen.



**Figure 58.** BrailleTap user interface

As shown in Figure 58, the BrailleTap calculator has no visual content except the edit text box in the top of the screen that is used to enter the equation and to allow people with low vision to see the entered numbers. The remaining of the screen is empty in order to allow users to tap and move their fingers over the screen without concerning about hitting any object on the screen.

### **5.3 Pilot Study**

We evaluated the BrailleTap calculator to examine the performance of the calculator as well as to compare it with the TalkBack calculator. Other touch-based methods that aim to increase the accessibility of calculators to visually impaired individuals include Tapulator and DigiTaps. However, in contrast to TalkBack these methods are not available on smartphone devices. In addition, DigiTaps does not enable numerical computation. As a

result, the researchers only compared BrailleTap to the TalkBack calculator. This evaluation was conducted with two participants who are completely blind and who are fluent in Braille. The participants were involved from the initial phase of the calculator's design and provided some suggestions about choosing the appropriate touchscreen gestures for operations. After we designed the application, they were asked to perform eight mathematics equations using both the BrailleTap calculator and the TalkBack calculator.

During a brief training period of approximately five minutes, participants learned how to use the BrailleTap calculator. This pilot study utilized the counterbalance methodology meaning that participant 1 used the BrailleTap application prior to the button calculator and vice versa. Participants stated that they liked the application because they can use Braille code, the language they are most familiar with, to input their commands.

We asked the participants to perform several mathematical equations using both the TalkBack calculator as well as the BrailleTap calculator. Table 18 indicates the amount of time that each participant spent to compute each mathematical equation on both calculators.

**Table 18: The time spent to use the button calculator and the BrailleTap calculator**

<b>Mathematical equation</b>	<b>Average time spent using button calculator</b>	<b>Average time spent using BrailleTap</b>
<b>51+36</b>	28 seconds	23 seconds
<b>54+34-15</b>	71 seconds	27 seconds
<b>50*200</b>	96 seconds	48 seconds
<b>450/5</b>	108 seconds	15 seconds
<b>50+25/4</b>	87 seconds	35 seconds
<b>600/5-50</b>	119 seconds	39 seconds
<b>200+40*2</b>	109 seconds	64 seconds

Table 18 shows that BrailleTap was faster than the button keyboard because it overcomes the navigation problem and the user can tap anywhere on the screen. BrailleTap

is also faster because it eliminates noisy audio feedback that is used to help the user identify where he is on the calculator interface or identify the button name. In the interview, she stated that the BrailleTap calculator is a promising tool that allows users to enter numbers based on Braille and perform operations using accessible gestures. She also likes being able to perform several tasks on the touchscreen without having to identify an object location. She also found it helpful to delete everything at once using a simple gesture.

Overall, the initial result indicates that the proposed calculator is easy to learn, more accessible than the TalkBack calculator, and it overcomes most of the limitations that are associated with the previously discussed calculators. In the BrailleTap calculator, users are not required to learn specific techniques to enter a number. Unlike Tapulator, where users have to memorize the prefix codes to be able to interact with the screen, BrailleTap allows them to simply use their experience with Braille. Based on our observation of the participants' interaction with both calculators, we noticed that there are noticeable improvements in both accessibility and speed of the BrailleTap calculator over the button calculator. Although it is common for these types of studies to use one or a few participants, the main limitation of the current study was being unable to recruit a large number of blind participants.

## **5.4 Conclusion**

We have presented BrailleTap, a non-visual number entry application with audio feedback based on Braille coding. The proposed calculator allows visually impaired people to perform basic operations on touchscreen devices more easily than presently available calculators. It does so by overcoming their main limitation, which is requiring visually



impaired individuals to locate an object on a touchscreen.

Based on the initial study, there are certainly opportunities for improvement to the BrailleTap calculator. Firstly, we need to test the calculator with larger groups of people via both qualitative and quantitative methods in order to better understand its strengths and weaknesses. We will also measure total typing time, digits per minute, as well as identifying types of errors. We will improve the calculator based on the participant's suggestions, for example, by adding other mathematics operations using accessible touchscreen gestures for blind users.

## **CHAPTER 6: BRAILLEPASSWORD METHOD**

This chapter presents a new authentication method on smartphone devices for blind people.

### **6.1 Research Challenges**

User authentication methods on touchscreen web applications have failed to tackle the challenges that blind people face when entering passwords. We claim that due to lack of touchscreen accessibility, the common authentication methods such as alphanumeric passwords are highly vulnerable to various attacks. An attacker uses different ways to discover the inserted password including hidden cameras, peeping or shoulder-surfing. The observation attacks harm blind people more than sighted people since it is very difficult for blind users to recognize and notice a shoulder surfer nearby or the existence of a hidden camera recording their private information (Saxena, Watt, 2009; Kuber, Shiva, Sharma, 2012). Saxena and Watt (2009) reported that it is very difficult to discover spyware monitoring password entry while using screen readers to navigate through an interface. When blind users do not trust an online website security, they usually rely on trusted sighted persons to aid them with accessing authentication mechanisms by giving them all the private information (Sauer et al., 2010).

Most of the challenges faced by blind users are caused by touchscreen devices, input methods and the restrictions of assistive technology. Touchscreen devices are used mostly to present a visual content on an interface. Touchscreen input methods also require users to visually locate a button on the screen. This poses difficulty for blind people to navigate through a visual layout on a screen. Thus, assistive technology like screen readers is used to convert the visual content to oral information. The main goal of integrating the screen reader on touchscreen devices is to help blind people obtain an accessible content

format like auditory. For example, in the QWERTY keyboard, they move their fingers over the screen until they find the desired character with the support of the TalkBack service. The TalkBack service reads aloud a character that is located under the users' fingers to help users navigate around the keyboard keys. When the user finds the desired character, they perform a double tap to insert it. Even though the QWERTY keyboard with the support of TalkBack service enables blind people to type on a touchscreen, it is time consuming and error prone (Alnfiai, Sampalli, 2016). Another problem is that listening to the TalkBack service reading out everything located under the user's finger while sequentially navigating from one object to another is annoying. The QWERTY keyboard with TalkBack service's drawbacks are not only distracting blind users when typing on a touchscreen, but it also poses security and privacy threats when accessing the authentication interface.

We are planning to improve web application security on smartphone devices by designing a new accessible and secure authentication technique called BraillePassword. BraillePassword uses gestures to input Braille characters on touchscreen devices. The BraillePassword uses two types of gestures to represent Braille dots of each character, and allows users to insert the password anywhere on the screen. BraillePassword does not provide visual or oral feedback thus it is resistant to aural and visual threats.

## **6.2 BraillePassword authentication technique**

Unauthorized access to an online account is the major security risk for blind users due to inaccessibility of the available authentication methods and the limitations that are associated with the current input methods on touchscreen devices. To overcome this issue, we have developed a novel touchscreen web authentication method that is entirely button-

free, and resilient to visual and aural eavesdropping, unlike the traditional authentication method.

### **6.2.1 Design Principles**

The design of our method is based on several design principles established by our previous study findings (Alnfai, Sampalli, 2016), findings from interviews with blind people (Azentok, 2012), and from an evaluation study on how blind people interact with smartphones (Paisios, 2012) and standard authentication guidelines (Schneider, 2009). The extracted design principles to overcome security and accessibility challenges are:

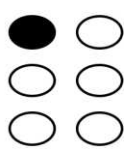
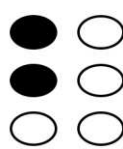
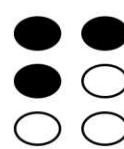
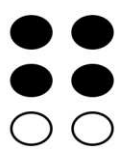
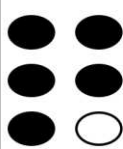
1. Allow users to use one finger to interact with a touchscreen.
2. Allow users to interact anywhere on a screen by eliminating visual interaction
3. Allow users to type passwords fast (speed)
4. Allow users to type passwords without audio feedback that permits unauthorized persons to listen to their private information (Robust to aural eavesdropping).
5. Allow users to type passwords without visual feedback that permits others to see their inputs (robust to visual eavesdropping)
6. Allow users to easily type combinations of capital and lower-case letters, numbers and punctuation (high password strength).

### **6.2.2 The BraillePassword Design**

BraillePassword is an observation-resilient authentication method that uses a gestural user interface and requires no modification on the server side. BraillePassword is button free, allowing the user to use Braille patterns to login anywhere on the screen. The

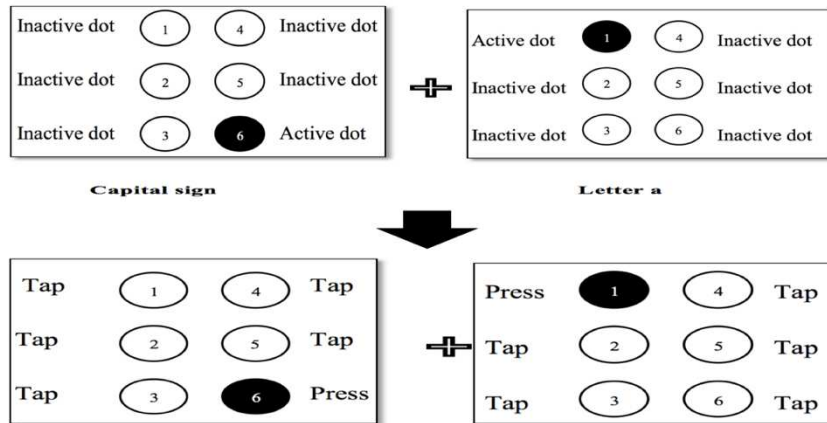
gestures that are used are a brief tapping gesture to insert unraised dots and a long tap gesture to insert raised dots in a Braille pattern. Braille code has a combination of six dots; some are raised and some are unraised, facilitating differentiation between characters. In BraillePassword, users can insert a combination of six long taps or short taps gestures to represent raised and unraised dots based on the order of these dots in the Braille code of a character. The BraillePassword was developed based on a similar technique, the BrailleEnter keyboard (Alnfiai, Sampalli, 2017). The BrailleEnter keyboard is used as an input method in the BraillePassword authentication method, allowing users to enter credentials and access online account securely. We also improved the BrailleEnter algorithm performance to insert characters on touchscreen devices. For example, we allow users to differentiate between lower and upper-case characters using Braille signs instead of performing swiping gestures

Users can type anywhere on a screen without taking into account the positions of any interaction made on the interface. They can type letters, numbers, punctuation, lower and upper cases (see Figure 59 and 60).

Braille characters based on both activated and inactivated dots				
<p>Letter a</p>  <p>(Press, Tap, Tap, Tap, Tap, Tap)</p>	<p>Letter b</p>  <p>(Press, Press, Tap, Tap, Tap, Tap)</p>	<p>Letter f</p>  <p>(Press, Press, Tap, Press, Tap, Tap)</p>	<p>Letter g</p>  <p>(Press, Press, Tap, Press, Press, Tap)</p>	<p>Letter q</p>  <p>(Press, Press, Press, Press, Press, Tap)</p>

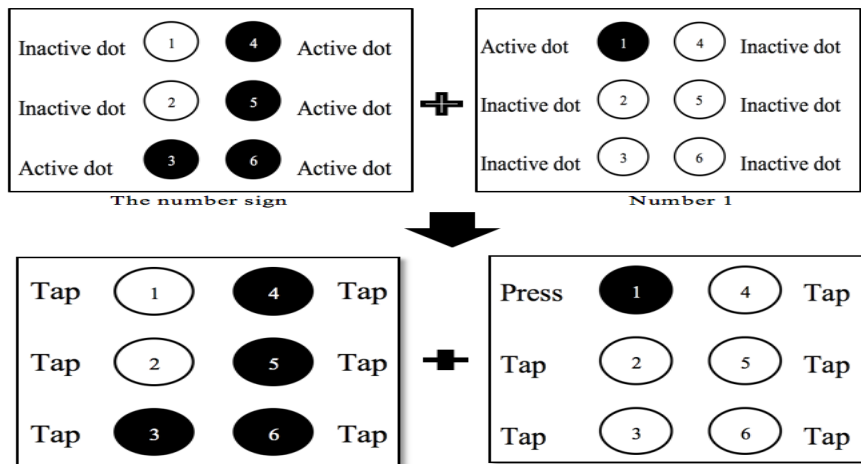
**Figure 59.** Sample of Braille lowercase letters

Figure 59 shows a sample of how can blind users type lower case letters based on Braille patterns using long tap and short tap gestures.



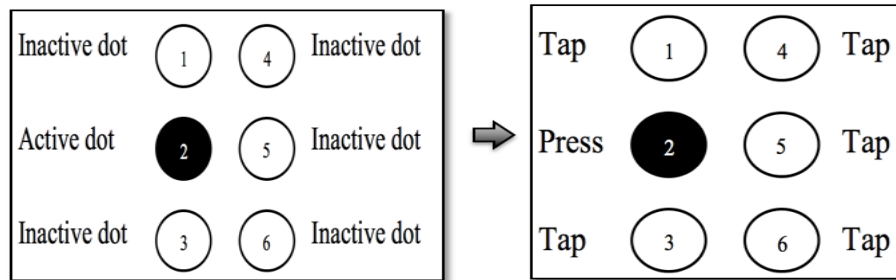
**Figure 60.** Example of inserting capital A in BraillePassword

Figure 60 shows a sample of how blind users can type upper case letters. First, a user can insert a Braille capital sign followed by the Braille pattern of letters.



**Figure 61.** Example of typing number 1 in the BraillePassword

In BraillePassword, blind users can type numbers as they do in Braille. First, they can type the prefix number sign, which requires the activation of dots 3, 4, 5, and 6; then they can insert the actual number. The number sign is placed before the characters from a to j in order to represent 0 to 9. For example, to type number 1, users can first type the Braille number sign and then number one based on Braille patterns (as shown in Figure 61).



**Figure 62:** Example of typing punctuation in BraillePassword

BraillePassword also enables users to type punctuation based on Braille. Users can type Braille punctuation sign then they can type the desired punctuation. First, user can insert the Braille symbol sign, which is tap, press, and 4 taps, followed by the Braille pattern of the punctuation they want to insert (see Figure 62).

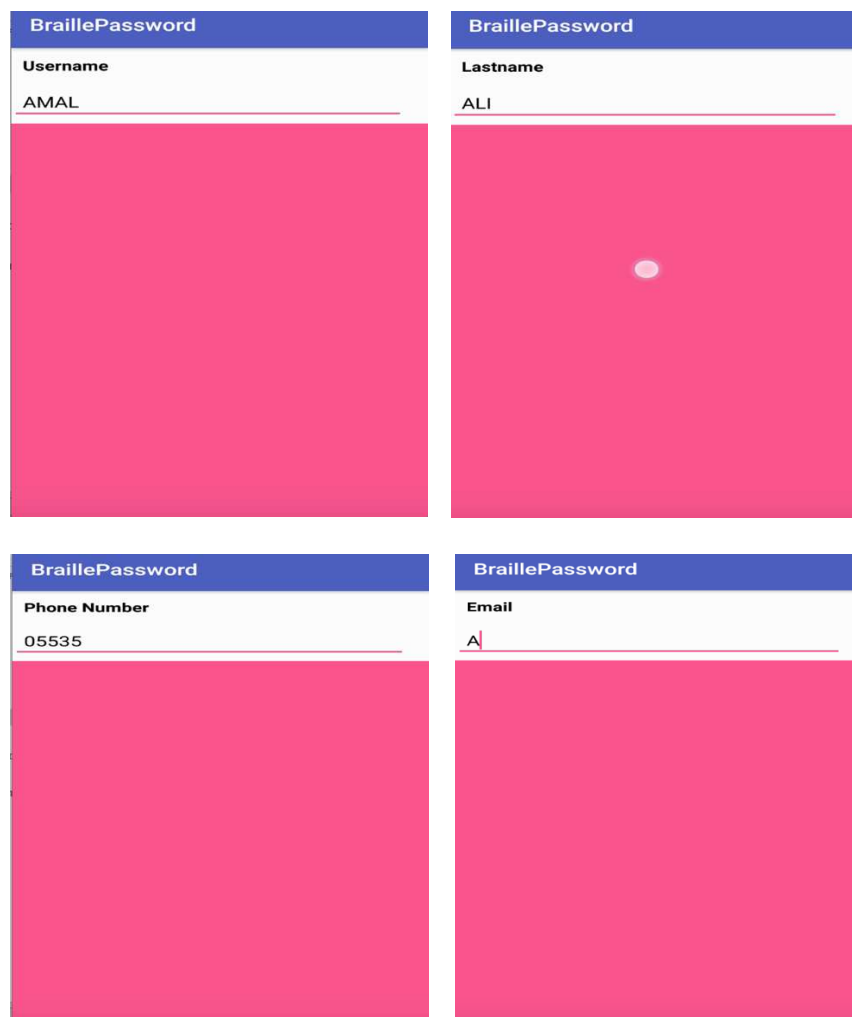
### 6.2.3 BraillePassword interfaces

BraillePassword interfaces are designed mainly to make touchscreen applications more accessible for blind people and to reduce the security challenges that they face when logging in to an online account.

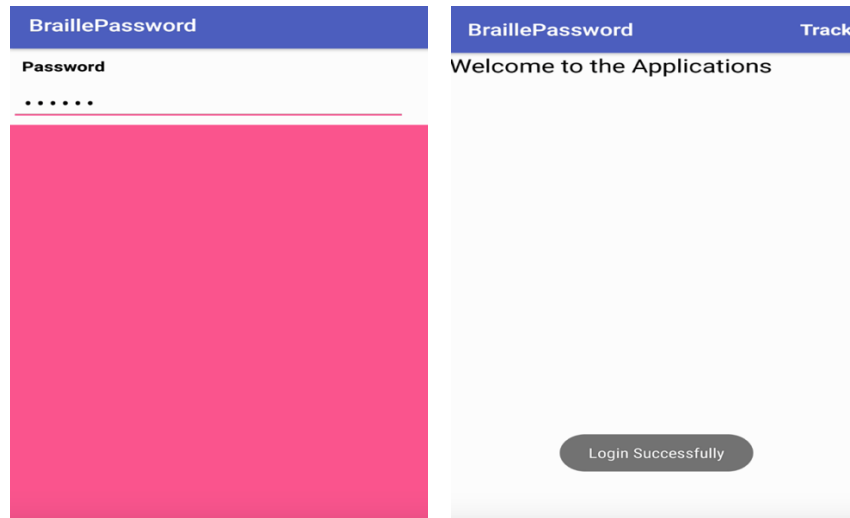
The application has two main interfaces, the signup interface and the login interface. Each interface has several pages that contain limited visual content.

### 6.2.3.1 Registration interface

The registration interface has five pages, as shown in Figure 63: First name page, last name page, email page, phone number page and password page. Each page contains only one field or edit text box, eliminating the need to navigate between several objects in one page. The first four pages, excluding the password page, are completely based on oral feedback, while the password page uses only haptic (vibration) feedback.







**Figure 63:** Registration pages

Users can switch between these pages by swiping from top to bottom. They enter text on all pages using the BrailleEnter keyboard. Users can enter letters, numbers and symbols in any page by entering the Braille signs for capital letter, number, or punctuation followed by the intended character code.

The data entered in the password page is represented by dots. Users receive a vibration each time they enter a digit in the password page. The main objective of this vibration is to inform users that they have entered a digit. In this page there is no visual or oral feedback, and the app reads only the keyboard layout (number, letter, or punctuation) when the users enter the Braille sign of a character.

### **6.2.3.2 Login interface**

The login interface has two pages, which are username and password page.

### **1. Username page**

The user name interface is completely based on gesture and provides oral feedback. This page has an edit text box that shows in the top of the screen (see Figure 63).

### **2. Password page**

The password interface is completely based on gestures and has an edit text function that shows the inserted characters as dots (see Figure 63).

BraillePassword uses audio feedback to inform blind users which page they are on. In the password page, we eliminated the audio and visual feedback in order to completely prevent observer attacks (shoulder surfing). Instead, we used a vibration feedback to inform users when they enter a digit.

#### **6.2.3.3 Password recovery**

In case the user forgets the password, he/she can tap 7 times anywhere on the screen, then the app will ask the user to enter the registered phone number in order to send a new password (as shown in Figure 64). Users can enter the received password and then they change their password.



**Figure 64.** Password recovery pages

#### **6.2.4 Interaction techniques**

Swiping is the easiest and most accessible gesture for blind users. Blind users can also swipe horizontally to edit their typing. The swiping gesture is used to allow users to add or delete spaces (As shown in Table 19).

**Table 19. Editing interaction techniques in BraillePassword**

<b>Interaction</b>	<b>purpose</b>
<b>Swipe from right to left</b>	Add space
<b>Swipe from left to right</b>	Back space

In order to switch between fields in the registration page, users can swipe vertically from the top to the bottom in a loop to open up a particular field. The first swipe will open up the first name field, and TalkBack service will read out the field name. After the users enter the first name and swipe vertically again, it will open up the last name page; and so on for the remaining separate pages. Performing a vertical swipe from top to bottom will open the

first name page. A second swipe from top to bottom will open the last name page. A third swipe from top to bottom will open the email page. Swiping vertically from the bottom to the top in a loop is also used to allow users to switch between login and registration interfaces. The BraillePassword has only one text edit opportunity in each interface, eliminating the need to navigate between objects on a screen. Table 2 shows the interaction types and their functions.

**Table 20: Switching between authentication interfaces in BraillePassword**

<b>Interaction technique</b>	<b>Purpose</b>	<b>User interface</b>
<b>First swipe from up to down</b>	Switching from user name to password field	The login interface
<b>Second swipe from up to down</b>	Switching from first name to last name field	The registration interface
<b>Third swipe from up to down</b>	Switching from last name to email field	
<b>Fourth swipe from up to down</b>	Switching from email to phone number field	
<b>Fifth swipe from up to down</b>	Switching from phone number to password field	
<b>Swipe from down to up</b>	Switching from registration interface to login interface or vice versa	

## 6.3 BraillePassword Implementation

### 6.3.1 BraillePassword authentication module

This section presents the BraillePassword architecture, which includes:

1. Designing a registration interface that includes first name, last name, email, phone number and password fields as well as login interface that includes first name and password fields.
2. Designing welcome interface that includes logout method

3. Setting up the database
4. Text-to-Speech function
5. Validation
6. Improved BrailleEnter keyboard

The flowchart below shows register and sign in processes (see Figure 65).

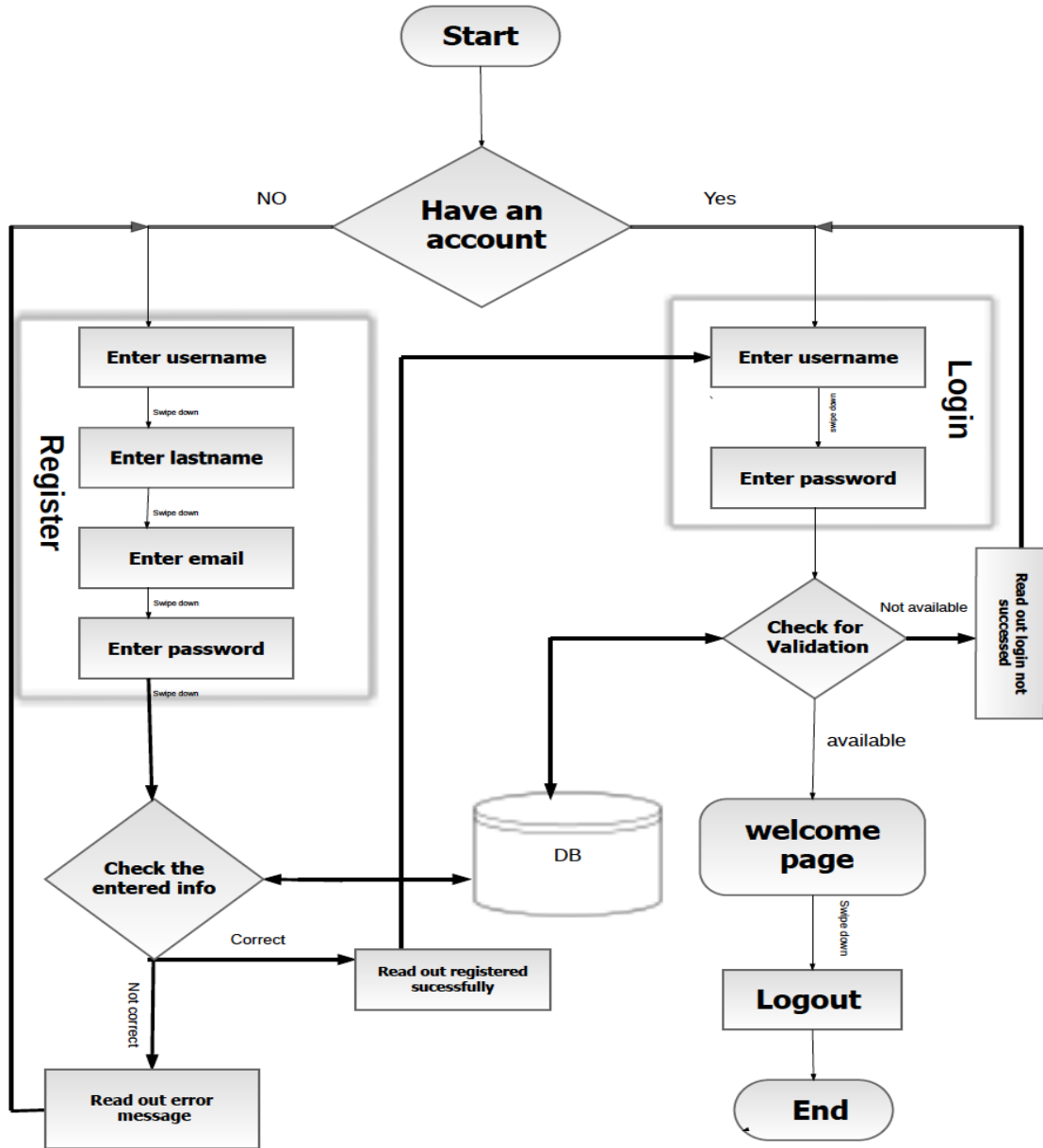


Figure 65. Register and sign in steps in the BraillePassword method

## 1. Designing registration and login pages

### 1.1 Register and login Algorithm

---

**/\* Registration and login Algorithm \*/**

---

**Comment:** Registration interface has 5 fragments and login interface has 2 fragments

**Comment:** Swipe right and Swipe Left are used for editing. Swipe up and Swipe down are used to move between pages.

**Step1. If** (the login interface is opened)

**Then** Swipe up

        Open the register interface

**End if**

**Step 2. Enter** first name **in the first fragment**

**Then** Swipe down

**Enter** last name **in the second fragment**

**Then** Swipe down

**Enter** email **in the third fragment**

**Then** Swipe down

**Enter** phone number **in the third fragment**

**Then** Swipe down

**Enter** password **in the fourth fragment**

**Then** Swipe down

**Step 3. If** (the entered data in all fields valid)

    Register a new user in the database

**Read out** “registered successfully”

**Open up** the login page

**Else**

**Read out** “registered failed”

**Return back** to the first field in the registration page

**End if**

**Step 4. Enter** the first name in the first field in the login page

**Then** Swipe down

**Enter** password **in the fourth fragment**

**Then** Swipe down

**Step 5. If** (the entered data in all login fields valid)

**Open up** the welcoming page

**Else**

**Display** an error message “login failed”

**Return** back to the page of the first name in the login interface

**End if**

**Step6. If** (the welcome page is open)

**Do** Swipe down

**Logout**

**End if**

**Step 7. If** (the user is already registered)

---

Swipe up to open login page  
Repeat step 4, 5 and 6 to login again

**Figure 66. Registration and login Algorithm**

Figure 66 shows the steps of filling the registration and login input fields.

## 1.2 Register and login interfaces

This section shows that XML files that represent the user interface elements of each field.

<pre>&lt;TextView   android:layout_width="wrap_content"   android:layout_height="wrap_content"   android:text="@string/lastname"   android:textColor="@color/blackColor"   android:textSize="16sp"   android:padding="10dp"   android:textStyle="bold"/&gt;  &lt;EditText   android:padding="10dp"   android:layout_marginRight="20dp"   android:id="@+id/lastnameEditText"   android:layout_width="match_parent"   android:layout_height="wrap_content"   android:textColor="#000000"/&gt;</pre>	<pre>&lt;TextView   android:layout_width="wrap_content"   android:layout_height="wrap_content"   android:text="@string/firstname"   android:textColor="@color/blackColor"   android:textSize="16sp"   android:padding="10dp"   android:textStyle="bold"/&gt;  &lt;EditText   android:padding="10dp"   android:layout_marginRight="20dp"   android:id="@+id/firstnameEditText"   android:layout_width="match_parent"   android:layout_height="wrap_content"   android:textColor="#000000"/&gt;</pre>	
<b>1. First name XML layout</b>	<b>2. Last name XML layout</b>	
<pre>&lt;TextView   android:layout_width="wrap_content"   android:layout_height="wrap_content"   android:text="@string/email"   android:textColor="@color/blackColor"   android:textSize="16sp"   android:padding="10dp"   android:textStyle="bold"/&gt;  &lt;EditText   android:padding="10dp"   android:layout_marginRight="20dp"   android:id="@+id/emailEditText"   android:layout_width="match_parent"   android:layout_height="wrap_content"   android:textColor="#000000"/&gt;</pre>	<pre>&lt;TextView   android:layout_width="wrap_content"   android:layout_height="wrap_content"   android:text="@string/phoneNumber"   android:textColor="@color/blackColor"   android:textSize="16sp"   android:padding="10dp"   android:textStyle="bold"/&gt;  &lt;EditText   android:padding="10dp"   android:layout_marginRight="20dp"   android:id="@+id/phoneNumberEditText"   android:layout_width="match_parent"   android:layout_height="wrap_content"   android:inputType="textPhonetic"   android:textColor="#000000"/&gt;</pre>	<pre>&lt;TextView   android:layout_width="wrap_content"   android:layout_height="wrap_content"   android:text="@string/password"   android:textColor="@color/blackColor"   android:textSize="16sp"   android:padding="10dp"   android:textStyle="bold"/&gt;  &lt;EditText   android:padding="10dp"   android:layout_marginRight="20dp"   android:id="@+id/passwordEditText"   android:layout_width="match_parent"   android:layout_height="wrap_content"   android:inputType="textPassword"   android:textColor="#000000"/&gt;</pre>
<b>3. Email layout</b>	<b>4. Phone number layout</b>	<b>5. Password layout</b>

**Figure 67. XML codes of Register and login fields**

## 2. Connecting XML files to java files

This section shows which XML files are working with Java files.

### 2.1 Register process

Users swipe up and the application directs users to “register.php” and show up a first field in the register interface, which is first name field.

After filling the first name field using the BrailleEnter keyboard, users swipe down to move to the next field, which is last name and so on for the remaining fields. After filling all the registering fields users swipe down to send all the registered data to signup() function. The signup function checks validation of the entered data in each field in the registration stage. The application will check if the registered fields are empty or one of them has invalid data, then it will show up an error message and read it out to the end users. However, if the users enter accurate and unregistered data in the registration fields then the application shows informative message “Registered Successfully” and immediately connects users to the database to store the entered data (as shown in Figure 68).

```

public void signup()
{
    if (ConnectionCheck.isConnectingToInternet(RegisterActivity.this)) {
        ShowProgress.createProgressDialog(RegisterActivity.this);
        Call<JsonElement> call = APP.getInstance().getConnector().register("register", firstname, lastname, email, password, phone);
        call.enqueue(new Callback<JsonElement>() {
            @Override
            public void onResponse(Call<JsonElement> call, Response<JsonElement> response) {
                try {
                    JsonElement j1 = response.body();
                    JSONObject js1 = new JSONObject(j1.toString());
                    String status = js1.getString("status");
                    if (status.equalsIgnoreCase(Constants.SUCCESS_STATUS)) {
                        ShowProgress.dismissProgressDialog();
                        Toast.makeText(RegisterActivity.this, "Register Successfully", Toast.LENGTH_LONG).show();
                        Speak("Register successfully");
                    }
                    else {
                        ShowProgress.dismissProgressDialog();
                        Toast.makeText(RegisterActivity.this, "Registration not success", Toast.LENGTH_LONG).show();
                        Speak("Registration does not success");
                        registerViewPager.setCurrentItem(4);
                    }
                }
                catch (JSONException e) {
                    ShowProgress.dismissProgressDialog();
                    Log.e("Exception", ""+e);
                }
            }
        });
    }
}

```

**Figure 68.** The Java code of the signup function

## 2.2 Login process

After users register in the BraillePassword, the application will immediately open the login interface. Users can enter the username in the first field in the login interface and then they swipe down to enter the password in the second field. If the entered data is valid, the



application presents a message “Login success” and then the application opens up the welcoming page, but if the entered data is invalid, the application presents an error message that says “Login failed”.

The entered data in the login fields are sent to the `signIn()` function to check if the username and password match the stored ones in the database. If the entered data matches the registered information in the database, then the application will create the session and open the main page; but if it does not match, the function displays an error message and redirects the users to login interface again (as shown in Figure 69).

At the welcome page, users can swipe up to logout and then the application will direct users to the login page.

```
public void signIn()
{
    if (ConnectionCheck.isConnectingToInternet(LoginActivity.this)) {
        ShowProgress.createProgressDialog(LoginActivity.this);
        Call<JsonElement> call = APP.getInstance().getConnector().login("login", username, password);
        call.enqueue(new Callback<JsonElement>() {
            @Override
            public void onResponse(Call<JsonElement> call, Response<JsonElement> response) {
                try {
                    JsonElement j1 = response.body();
                    try {
                        JSONObject js1 = new JSONObject(j1.toString());
                        String status = js1.getString("status");
                        if (status.equalsIgnoreCase(Constants.SUCCESS_STATUS)) {
                            ShowProgress.dismissProgressDialog();
                            Speak("Login Successfully");
                            Toast.makeText(LoginActivity.this, "Login Successfully", Toast.LENGTH_LONG).show();
                            LoginPreference loginPreference = new LoginPreference(getApplicationContext());
                            loginPreference.setLogin(true);

                            Handler h = new Handler();
                            h.postDelayed(run() -> {
                                Intent intent = new Intent(LoginActivity.this, WelcomePage.class);
                                startActivity(intent);
                                finish();
                            }, 1500);
                        } else {
                            ShowProgress.dismissProgressDialog();
                            Speak("Login not success");
                            Toast.makeText(LoginActivity.this, "Login not success", Toast.LENGTH_LONG).show();
                            vibratePhone();
                            LoginPreference loginPreference = new LoginPreference(getApplicationContext());
                            loginPreference.setLogin(true);
                        }
                    } catch (JSONException e) {
                        ShowProgress.dismissProgressDialog();
                        Speak("Internal Server Error.Login Failed");
                        signUpViewPager.setCurrentItem(1);
                    }
                }
            }
        });
    }
}
```

**Figure 69.** Snapshots of login function code

We used vibration to inform users when they entered a digit in the password field (see Figure 70).

```

public void vibratePhone()
{
    Vibrator v = (Vibrator) LoginActivity.this.getSystemService(Context.VIBRATOR_SERVICE);
    // Vibrate for 500 milliseconds
    v.vibrate(500);
}

```

**Figure 70. Vibration function**

### 3. Designing the welcome interface

This section shows the welcome interface element, which represents text that displays a welcome message (see Figure 71).

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/welcome"
    android:textColor="@color/blackColor"
    android:textSize="22sp"/>

```

**Figure 71. XML code of welcome interface**

### 4. Setting up the database

We used the free 000webhost to create a MySQL database by clicking on phpmyadmin and to create a table that has five columns. We created a database that has one table called “Registration Table” to store users’ information when they registered (Figure 72). The table has five fields which are first name, last name, phone number, email and password.

id	firstname	lastname	email	phone_number	password
52	Mrim	ALNFIAI	MRIM@HOTMAIL.COM	0566811011	ABC1133

**Figure 72. A snapshot of register database**

#### 3.4 Connecting the Android app with the database

We created two PHP files that allow the web Android application to communicate with the database on the server (see Figure 73 and 74).

### 3.4.1 Login.php

```
<?php
$host="localhost";
$user="id1716916_maraimal"; //off line we must keep user name :root and password:blank ""
$password="braille"; //in online we assign the username and password with our choice
$database="id1716916_BraillePass";
$link=mysqli_connect($host,$user,$password,$database) ;
if($link)
{
    //echo "selected";
}
else
{
    echo mysqli_error($link);
}
?>
```

Figure 73. Login PHP code

### 3.4.2 Register.PHP

```
<?php
include("db.php");
$status = "init";
$block=$_REQUEST['block'];
$item=array();
$abc=array();
$user_email="";
$user_id="";
if($block == "register")
{
    $value1 = false;
    $value2 = false;
    $value3 = false;
    $value4 = false;
    $value5 = false;
    if($_REQUEST)
    {
        if(isset($_REQUEST['first_name']))
        {
            if($_REQUEST['first_name'] == "")
            {
                $status="first_name Should Not Be Blank";
            }
            else
            {
                $first_name = $_REQUEST['first_name'];
                $value1 = true;
            }
        }
        else
        {
            $status = "first_name Parameter Missing";
        }
        if(isset($_REQUEST['last_name']))
        {
            if($_REQUEST['last_name'] == "")
            {
                $status="last name Should Not Be Blank";
            }
            else
            {
                $last_name = $_REQUEST['last_name'];
                $value2 = true;
            }
        }
        else
        {
            $status = "last name Parameter Missing";
        }
        if(isset($_REQUEST['email']))
        {
            if($_REQUEST['email'] == "")
            {
                $status="email Should Not Be Blank";
            }
            else
            {
                $email = $_REQUEST['email'];
                $value3 = true;
            }
        }
        else
        {
            $status = "email Parameter Missing";
        }
        if(isset($_REQUEST['password']))
        {
            if($_REQUEST['password'] == "")
            {
                $status="password address Should Not Be Blank";
            }
        }
    }
}
```

Figure 74. Register PHP code

We uploaded the PHP files to the server by opening up the cpanel and then file manager. After that, we uploaded the login.PHP and register.PHP files under the public html folder.

### 3.5 Connecting PHP files to Android app

```
public interface ApiEndPoint {
    @FormUrlEncoded
    @POST("registration.php")
    Call<JsonElement> register(
        @Field("block") String block,
        @Field("first_name") String first_name,
        @Field("last_name") String last_name,
        @Field("email") String email,
        @Field("password") String password,
        @Field("mobile") String mobile);

    @FormUrlEncoded
    @POST("registration.php")
    Call<JsonElement> login(
        @Field("block") String block,
        @Field("firstname") String email,
        @Field("password") String password);

    @FormUrlEncoded
    @POST("registration.php")
    Call<JsonElement> resetPassword(
        @Field("block") String block,
        @Field("password") String password,
        @Field("phone_number") String phone_number);
}
```

**Figure 75.** Connecting PHP files to Android app code

### 3.6 Connecting PHP to MySQL Database

```
public class ServiceGenerator
{
    public static String BASE_URL = "http://maraim.000webhostapp.com/BraillePassword/";

    public static OkHttpClient.Builder httpClient = new OkHttpClient.Builder();

    public static <S> S createService(Class<S> serviceClass, String base)
    {
        Retrofit.Builder builder =
            new Retrofit.Builder().baseUrl(base).addConverterFactory(GsonConverterFactory.create());
        return createService(serviceClass, null, null, builder);
    }

    public static <S> S createService(Class<S> serviceClass, String username, String password, Retrofit.Builder builder)
    {
        if (username != null && password != null) {
            String credentials = username + ":" + password;
            final String basic =
                "Basic " + Base64.encodeToString(credentials.getBytes(), Base64.NO_WRAP);
            httpClient.interceptors().add(intercept(chain) -> {
                Request original = chain.request();
                Request.Builder requestBuilder = original.newBuilder()
                    .header("Authorization", basic)
                    .header("Accept", "application/json")
                    .method(original.method(), original.body());
                Request request = requestBuilder.build();
                return chain.proceed(request);
            });
        }
    }
}
```

Figure 76. Connecting PHP file to MySQL Database

### 5. Text-to-Speech function

All the BraillePassword responses will be read out using Speak function. The Speak function uses the Text-to-Speech methods to convert text to speech. It has been explained in Chapter 4.

### 6. Validation

```
public class Validation {
    public static boolean isValidEmail(String email) {
        String EMAIL_PATTERN = "[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@"
            + "[A-Za-z0-9-]+(\\.[_A-Za-z0-9-]+)*([A-Za-z]{2,})$";
        Pattern pattern = Pattern.compile(EMAIL_PATTERN);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }
    // validating password
    public static boolean isValidPassword(String pass) {
        if (pass != null && pass.length() >= 6) {
            return true;
        }
        return false;
    }
    public static boolean isValidPhone(String phoneNumber) {
        if (phoneNumber != null && phoneNumber.length() == 10) {
            return true;
        }
        return false;
    }
}
```

Figure 77. The Java code of validation function

## 7. Improved BrailleEnter keyboard

The BrailleEnter keyboard was used as the main input method in the BraillePassword application and the BrailleEnter keyboard has been explained in detail in Chapter 4. However, we made slight changes in the BrailleEnter keyboard functions.

### 1. Changes to the keyboard layout

To type capital letters, users can tap on the screen Braille Capital sign to switch from small letters to capital letters. Once the users tap the Braille capital sign the application will say “Capital letters” and it will switch immediately to uppercase layout (see Figure 78). Similarly, to type numbers, users can type first the Braille number sign and then the number code in Braille (see Figure 79). The same process applies to allow users type symbols.

```
else if (!mTouchHelper.get(0).isPressed() && !mTouchHelper.get(1).isPressed() && !mTouchHelper.get(2).isPressed() &&
        !mTouchHelper.get(3).isPressed() && !mTouchHelper.get(4).isPressed() && mTouchHelper.get(5).isPressed()) {
    // Enter Braille uppercase sign
    mode = 1;
    Speak("Capital Letters");
}
```

**Figure 78.** The code of Braille capital sign

```
if (!mTouchHelper.get(0).isPressed() && !mTouchHelper.get(1).isPressed() && mTouchHelper.get(2).isPressed() &&
    mTouchHelper.get(3).isPressed() && mTouchHelper.get(4).isPressed() && mTouchHelper.get(5).isPressed()) {
    // Enter Braille number sign
    Speak("Numbers");
    mode = 2;
}
```

**Figure 79.** The code of Braille number sign

## 2. Transferring between pages

Swiping down gesture is used in BraillePassword to change the fields of the registration and login page. Users swipe down to move from the first name field to last name and so on. On the other hand, swiping up is used to transfer between the registration and login pages (see Figure 80). If the users are in the registration interface, they swipe up to go to login interface and vice versa. In addition, swiping up used to allow users to logout from the welcoming page.

```
if(txt.equals("SWIPED_TOP"))
{
    Intent intent1 = new Intent(RegisterActivity.this,LoginActivty.class);
    startActivity(intent1);
}
else if(txt.equals("toForgotPassword"))
{
    Intent intent1 = new Intent(RegisterActivity.this,ForgotPasswordActivity.class);
    startActivity(intent1);
}
else if(txt.equals("SWIPED_DOWN"))
{
    switch (fragType) {
        case Firstname:
            if(firstname==null || firstname.isEmpty() || firstname.equalsIgnoreCase(""))
            {
                Speak("Please enter first name");
                registerViewPager.setCurrentItem(0);
            }
            else
            {
                registerViewPager.setCurrentItem(1);
            }
            break;

        case Lastname:
            if(lastname==null || lastname.isEmpty() || lastname.equalsIgnoreCase(""))
            {
                Speak("Please enter lastname");
                registerViewPager.setCurrentItem(1);
            }
            else
            {
                registerViewPager.setCurrentItem(2);
            }
        }
    }
```

Figure 80. Moving between fields

### 6.4 BraillePassword Entropy

Password entropy is a mathematical measurement of how difficult is it to predict (i.e., hack) a password. Thus, password entropy can be described as a password's randomness. The

entropy of a password is calculated based on the character set used to create a password and the length of the password. The set of characters includes lowercase, uppercase, and numbers, as well as punctuation. Considering numbers from 0 to 9, lowercase alphabet from a to z, uppercase alphabet from A to Z, and symbols including `~! @# \$ % ^ & \* ( ) \_ + - = { } | [ ] \ : " ; ' < > ? , . / , there are 95 possible characters that can be used to create a password (10 + 26 + 26 + 32+ space). The formula to calculate entropy is:

$$\text{Entropy} = \log(C) / \log(2) * L \tag{7}$$

where C is the character set size and L the password length. In both methods implemented in this experiment, the character set size is 95 and the password length is 6 bits. Thus, the entropy of the both authentication apps is

$$\log(95) / \text{Log}_2 * 6 = 39.41 \text{ bits.}$$

BraillePassword entropy is similar to that of traditional authentication methods, with the same character set. However, BraillePassword differs from the traditional authentication method in critical ways. First, BraillePassword allows users to type anywhere on a screen without navigating to a specific object or key. Moreover, bystanders can only see the number of digits that are entered in the password field, and if the BraillePassword interface is recorded while users enter passwords, it may be very difficult to interpret the various gestures and identify the character entered, as each character requires six interactions. Thus, it should be substantially more resilient to observation attacks.

#### 4.2 6.5 USER STUDY

We conducted a comparative evaluation to examine the performance, accessibility, memorability and security of BraillePassword and a traditional alphanumeric method when



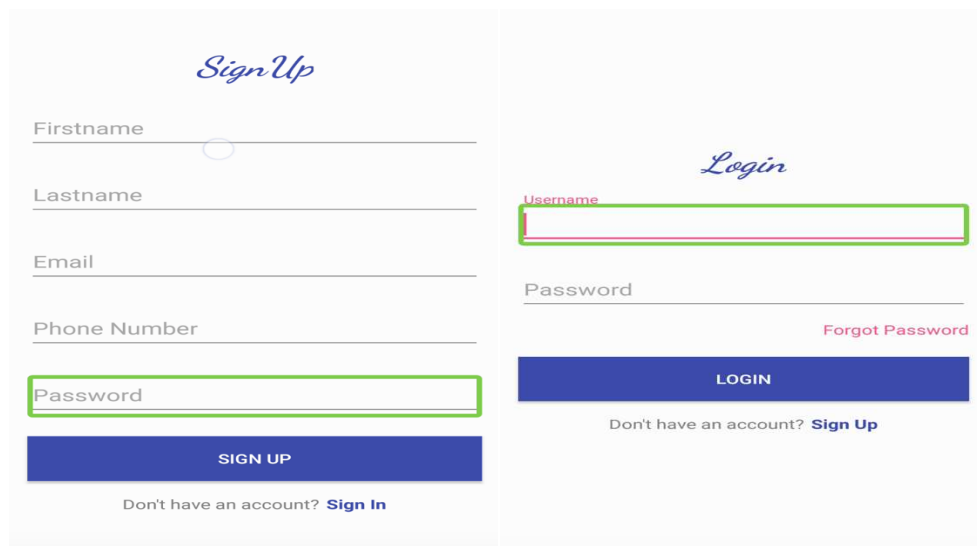
registering and logging in to web applications. We also sought to determine whether the methods are resilient to video and shoulder surfing attacks. The user study has been approved by the university.

### **6.5.1 Apparatus**

To evaluate the two methods, we built prototype applications for BraillePassword authentication method and traditional web authentication password entry. The traditional authentication interfaces were visually similar to the typical web authentication system used by most websites (as shown in Figure 81). The implemented traditional authentication app enables double-tap selection of keys from the QWERTY keyboard layout and supports Talkback service.

BraillePassword uses the BrailleEnter keyboard as an input method that uses Braille patterns, with a text-to-speech function for reading out entered data. Both password systems require users to enter six digits.

For the study tasks, we used an Android mobile phone that runs on Google's operating system, which is an ASUS mobile phone. Both the BraillePassword and traditional authentication method run within the Android operating system.



a) Registration interface

b) Login interface

**Figure 81.** The traditional authentication application

### 6.5.2 Participants

From September 2017 to January 2018, ten blind or visually impaired participants aged 22 to 50 years old were recruited from the Taif University Accessibility Center and the Canadian National Institute for the Blind (CNIB) in Halifax. An approval from the Dalhousie University Social Sciences and Humanities Research Ethics Board was acquired before commencing the study (see Appendix K). Table 21 shows participants' demographic data. Five participants were undergraduate students, three were graduate students, and two were employees. Eight participants are from Taif university accessibility center and 2 participants are from CNIB. All participants were proficient in reading and writing Braille and had at least five years' experience with touchscreen devices. All participants used Perkins keyboard with slate and stylus to type Braille texts. Nine participants used the screen reader as the main assistive technology to help them navigate

while using the touchscreen devices, while one participant used the Zoom (magnifier) to locate objects on a screen when she was in a public place or had a headache other than that she used the screen reader. All participants had experience with Android devices but had switched to iPhones because Android TalkBack service is inefficient and sometimes does not respond. All participants depend on their friends to perform some daily tasks like withdrawing money from an ATM machine or paying for goods and services through debit cards.

**Table 21. Participants’ demographic data**

<b>P</b>	<b>Age</b>	<b>Sight</b>	<b>Experience with smartphones</b>	<b>Assistive technology</b>	<b>Touchscreen keyboard</b>
<b>P1</b>	22	Legally blind	6 years	Perkins keyboard	QWERTY keyboard
<b>P2</b>	24	Legally blind	7 years		
<b>P3</b>	23	Legally blind	6 years and half		
<b>P4</b>	27	Legally blind	6 years		
<b>P5</b>	20	Visually impaired	5 years		
<b>P6</b>	34	Visually impaired	8 years		
<b>P7</b>	24	Legally blind	6 years		
<b>P8</b>	28	Legally blind	8 years		
<b>P9</b>	51	Legally blind	8 years		
<b>P10</b>	38	Legally blind	8 years		

### 6.5.3 Experimental Design

The experiment used within-subject design. The main factors are the authentication methods: BraillePassword, traditional authentication method and the levels of order were (1, 2).

The main objective of this experiment was to evaluate the authentication methods’ memorability and performance, including both password speed and accuracy as well as

accessibility. The experiment used a within-subjects design in which each participant would use both authentication methods; the order of methods used was alternated across participants to account for Authentication applications. The dependent variables were creation time, login time, password entry speed, failure rates, memorability, and whether the authentication method is resilient to video and shoulder surfing attacks.

### **6.5.3.1 Study procedure**

At the beginning of the study, the study goal and tasks were explained. After the participants gave informed consent, they answered a brief background questionnaire to provide us with basic demographic data and information about which authentication methods participants use to access to websites on a touchscreen, the challenges they face while entering credentials, and the ways they mitigate these challenges (see Appendix I). The questionnaire was completed orally by the researcher and participants.

As noted above, the order in which the methods were tested was alternated across participants. For each of the two authentication methods, each participant went through a training session followed by a test session. In the training phase, the investigator showed the participant how the authentication method works, and explained how participants can enter their authentication information. Participants were able to correct typing errors during password entry; a password could be “reset” if the user makes an error by clicking on password recovery link and re-entering the phone number. Such errors and corrections were included in the time measured for a given password entry. To make them familiar with the authentication technique interfaces, we had them register and login twice. The first registration and login information, including user name and password, were given by the

researchers. The registration and login information for the training session is shown in Table 22.

**Table 22. Registration information for the training phase**

<b>User name</b>	<i>Mrim</i>
<b>Last name</b>	<i>Alnfiai</i>
<b>email</b>	<i>Alnfiai@hotmail.com</i>
<b>Phone number</b>	9023297444
<b>password</b>	<i>Mrim123@</i>

In the test session, participants were asked to register their username and password for the first account. Participant were asked to create their own username and password to simulate a realistic password creation and entry scenario, and to memorize this information. We also informed participants that the password they generate should be secure, easy to memorize, and difficult for others to guess. In addition, researchers emphasized that participants should not use the passwords that they use in “real life,” nor use any password managers to help memorization. They were told to behave as though entering real information but, but to use fake usernames and passwords for both applications. The registration task was immediately followed by a corresponding login task. After completing the tasks with the first authentication method, participants went through the same training and test sessions with the second method.

Participants’ interactions with both authentication methods were screen recorded, and time was recorded using an app that capture activity on smartphone screens. This provided the researcher with the time it took users to register and to login.

Participants then took the Android smartphone device with them to perform tasks in their daily lives; these tasks included logging in to the two authentication applications created by the researcher.

On the first day of the week, participants were asked to log in to web applications on the Android device at different times. The participants' login data was stored in a database at [www.000webhost.com](http://www.000webhost.com), a free web and database hosting plan. For each account, the intervals between logins were one hour, one day, two days, and one week following the creation task. The order related to tasks of those accounts was the same for each participant. After a week, we invited each participant back to do the final task, which is logging in to both accounts once again.

Participants were asked to screen record the entirety of their interactions with their smartphones while registering or logging to both applications during the week, from the first to the last character that they entered. These records were used to examine the creation time, login time, failure rate, recall rate, and overall error rate, and to identify typing errors. In addition, the recordings aided in identifying technical problems participants encountered while registering or logging in to the applications to determine whether and to what extent the application is accessible.

The video recordings were used to determine the degree to which the application is secure and resilient to shoulder attacks. After completing the study tasks with the both authentication applications, participants had an interview about the applications' strengths and limitations and to discuss the features they liked and disliked (see Appendix J). At the end of the study, The researcher watched the video recordings three times trying to guess the passwords.

#### **4.3 6.5.4 RESULTS**

The results of the experiment include quantitative findings related to registration time, login time, failure time, password entry speed, and password strength, as well as qualitative findings related to the participants' experiences using the authentication methods and how resilient the methods are to observation attacks alongside with suggestions for improvement from participants.

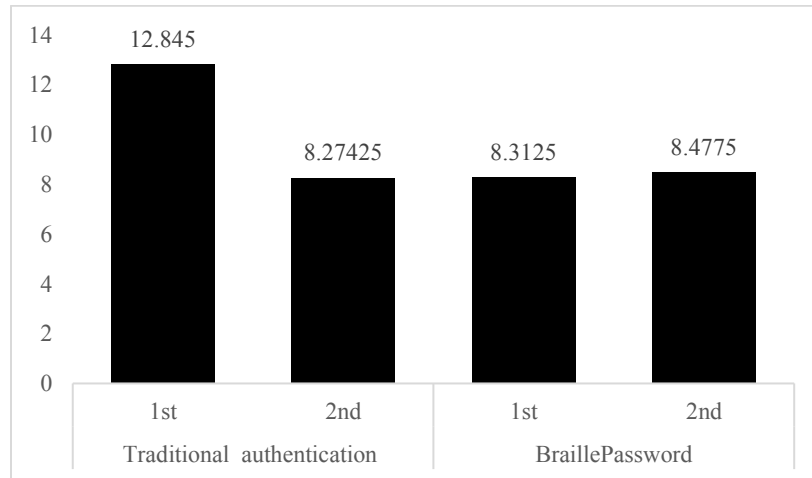
##### **6.5.4.1 Quantitative results**

###### **6.5.4.1.1 Registration time**

There was a significant difference in the registration time for the BraillePassword (mean = 7.91 minutes; SD = 3.39) and the traditional authentication method (mean = 10.17 minutes, SD = 3.13), paired t-test;  $P < 0.03$ ). The shorter registration time seen with BraillePassword is likely because, with the traditional method, users must find the location of the wanted key by moving their fingers over the screen, listening to all characters until they have found the desired key. Another reason for the difference in is that in the traditional authentication method, users have to search for the right authentication areas on the registration interface by swiping from left to right trying to move between the fields in the registration interface with the support of the screen reader.

Results revealed no significant effect of order ( $F_{1, 8} = 0.065$ ,  $P = 0.80$ ) and no significant interaction of authentication method by order ( $F_{1, 8} = 0.95$ ,  $P = 0.36$ ). As Figure 82 shows, there is no significant difference between participants' speed who registered in the traditional authentication account after they had registered in BraillePassword than participants who registered in the traditional authentication method first ( $F_{1, 8} = 2.95$ ,

P=0.12). Similarly, order did not affect registration time for BraillePassword ( $F_{1,8} = 0.95$ ,  $P=0.36$ ).



**Figure 82.** Registration time for both authentication methods and order

#### 6.5.4.1.2 Login time

There was not a significant difference in the authentication time for BraillePassword login time (mean= 1.56 minutes, SD= 0.27) and the traditional authentication method (mean= 1.68 minutes, SD= 0.73), paired t-test;  $P=0.25$ ). This lack of difference is likely because the login interface in the traditional authentication method has only two fields, and locating the authentication fields does not consume too much time, unlike the registration interface. Having so few elements in the user interface reduces the confusion of locating the intended authentication field, resulting in similar performance between methods.

#### 6.5.4.1.3 Password entry speed

Password entry time was measured from the time the users touched the screen to the time they stopped touching it by lifting up their finger after entering the sixth digit on the



password field. In the BraillePassword method, participants spent an average of 61 seconds entering six Braille characters in the password field, with the average time dropping from 67 seconds to 59 seconds over the course of the week. In the traditional authentication method, the average time spent to enter the password was 57 seconds; over the course of the week, mean time dropped from 71 seconds on the first day to 43.5 seconds at the end. Overall, there was not a significant difference in the password entry speed for the BraillePassword (mean = 61 seconds, SD= 0.63) the password entry speed for the traditional authentication method (mean = 57 seconds, SD= 0.8), paired t-test; P = 0.43).

Even though the traditional authentication sometimes takes additional time to locate the authentication areas, there was no overall difference. Here we only calculated the time when the user locates the password field in the traditional authentication method to compare only the time spent to enter six digits in both methods. Participants may need more time to do mental and physical efforts to enter each Braille digit in the password field.

#### **6.5.4.1.4 Failure rate**

The failure rate was slightly lower for the traditional authentication method. There was not a significant difference in the failure rate for the traditional authentication method (Mean= 7.5%, SD= 0.38) and the BraillePassword (Mean=17.5%, SD=0.266), paired t-test; P = 0.20).

#### **6.5.4.1.5 Password accuracy and memorability**

All participants were able to remember their password using the BraillePassword interface and the traditional authentication method both over a short time (logging immediately after

registering to an account) and over a longer period (a week). Even though participants did not succeed at logging in to their accounts 100% of the time, this was due to interaction problems associated with each application interface. All participants remembered their traditional method passwords as well as the BraillePassword digits after a short and long period of time.

After both short and long periods, when participants failed to log in to their account, they did not try to reset their password because they remembered their passwords but had entered them incorrectly due to the accessibility issues related to the user interface or the limitations of the input methods.

#### **6.5.4.1.6 Password Strength**

We examined password strength by observing the 16 passwords generated by users on the traditional authentication and BraillePassword methods. We found that the most common password digits used in the traditional authentication methods were the characters located in the left corner of the QWERTY keyboard, like ‘a’ ‘s’, ‘d’ and ‘z’, because they are easy to locate. This differs somewhat from previous research by Burnett (2006), which found that the most common digits used to create passwords in the traditional authentication method are ‘a’, ‘1’, ‘e’, ‘o’, and ‘r’. We also observed that blind participants selected characters from one keyboard layout that might be due to the difficulty and efforts required to switch to uppercase letters or special characters, thus making their passwords comparatively easy to guess and observe.

In BraillePassword, we found out that the most common characters were ‘a’ or ‘1’, demonstrating that blind people tend to choose the easiest characters to enter, those being

characters with few active Braille dots. The Braille pattern of letters ‘a’ and number ‘1’ is easier to guess, so users should avoid those guessable characters.

Blind and visually impaired people tended to choose passwords with low entropy while using the traditional authentication method that due to the efforts are required to switch between keyboard layers or locate characters on the QWERTY keyboard. However, they entered the same passwords on the BraillePassword method, but that does not affect the strength of BraillePassword entropy as surfers cannot differentiate between characters as the method provide no aural and visual feedback.

#### **6.5.4.1.7 Guessing of passwords from camera videos**

To measure the degree of protection against video and shoulder attacks afforded by each method, the researcher watched 96 videos that show the participants’ attempts while registering or logging into both applications over a week. The researcher was able to guess Braille password digit if the entered digits were repeated an ‘a’ or ‘1’. The guessable rate of the BraillePassword was 12.5%. However, if BraillePassword contained characters that have more than one active Braille dot, it was very difficult to guess characters, probably because of the difficulty in tracing gestures associated with user interactions. Entering Braille dots in one position without moving the participant’s finger on the screen also makes it very hard to trace the entered character. Thus, only once was the researcher able to replicate the Braille password, and this was only when the participants entered ‘a’ six times as a password.

In the traditional authentication method, the researcher was able to easily guess the password digits by tracing participants’ fingers during interaction with the QWERTY keyboard, or by listening to the entered characters. The researcher was always able to guess

the entered passwords when participants were logging in to the web application account and the guessable rate was 100%.

#### **6.5.4.2 Qualitative result**

##### **6.5.4.2.1 Pre-questionnaire**

The participants were asked about the type of touchscreen platforms they have used, the authentication methods that participants have used to access to websites on a touchscreen, and the challenges they face while entering credentials, as well as how they mitigate these challenges.

All participants have experience in both the iPhone and Android platforms. All of them use the Android platform for a period of time and then they transferred to the iPhone device due to several accessibility reasons that participants mentioned. The iPhone platform provides a free and proficient screen reader that supports different languages and offers multiple tones. In contrast, the Android platform does not provide a free and efficient screen reader.

All participants used the traditional authentication method with the TalkBack or VoiceOver service to access websites on smartphone devices, using the QWERTY keyboard with audio feedback to fill out the registration and login fields. Only one participant (p2) had tried the Braille keyboard on the iPhone device; she said it is complicated and hard to place three fingers from each hand on a smartphone screen, making it error prone.

All participants have used the traditional authentication method to access websites or PIN for the device, which is the common authentication method provided by most web

applications. They do not have experience with the graphical password or pattern password because these techniques require vision to select the intended image or object. Participant 5 used the pattern technique for a short period, but she said it is easy to crack and can be easily overseen by an observer. All participants use the biometric authentication method to access the PIN, but it is not the main method that they have used. Participants reported that they did not completely rely on biometric authentication because biometric reader sometimes does not read their fingerprint info or does not respond. Thus, they use the traditional authentication method as the main method to access their phone.

The main difficulties that participants have faced while registering or logging in to web applications, such as Facebook, Twitter or bank accounts, are that participants are not able to locate the authentication area on a web application interface, like user name, email, etc. Another challenge they reported is that they are not able to enter the CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) digits because, as participant 2 stated, the voice is not clear. This outcome is supported by previous work, as Bigham and Cavender (2009) found that entering the audio CAPTCHAs is very hard and time consuming. These types of challenges lead blind users to give their login credentials to others (participant 7: “she always asks friend to help her enter the CAPTCHA info”), indicating that this population is willing to give their login credentials to others in order to overcome the accessibility issues.

As mentioned above, most blind participants use headphones and screen curtains (or decrease the screen light) while logging in to a web application on a smartphone device in a public place. Visually impaired participants also use confusion techniques, where they enter erroneous digits and then edit them. This finding is supported by previous research;

for example, De Luca et al. (2009) found that the easiest method used to protect their credentials is to input incorrect information to confuse a bystander. Reflecting security concerns, none of the participants reported opening bank applications from their touchscreen devices. Participant 1 stated that “I have never entered my bank information data on my mobile device. I do not trust the app and I feel it is not safe.” Participant 5 agreed and stated that “I do not open the bank account from the phone because I feel it is not safe and the web application is not accessible because it does not support the VoiceOver service.” Furthermore, only participants 1 and 2 store private and personal information on their device, with the remainder reporting that they do not do so.

All participants except participant 5 automatically store their passwords in web applications so they do not have to enter the password repeatedly. Participant 5 stated that she “uses one password for all websites so she does not need to store the password.”

All totally blind users who completely rely on the screen reader while interacting with touchscreens have concern about aural eavesdropping. They use their headphones to prevent people around them from hearing what has been typed on the screen. P1 stated that “I always use the headphones. If I do not have the headphone, I cannot access my social accounts in public places because I am concern about my information and the VoiceOver sound is noisy and annoying.” Similarly, participant 2 stated that “I use the headphones because the Voiceover is noisy and the device reads out clearly my private information.” Participant 7 said “I have never used my phone in public places”. Participant 8 stated that “I am worried if someone around me might be able to hear my personal info. If I do not have the headphone, I go to my room and use my device there.”

Similarly, all participants had concerns about visual eavesdropping while interacting with smartphone devices in public places and they use the screen curtains to hide their screens. Participant 5, who is visually impaired, said “I sometimes try to hide the screen or type then delete to confuse who is looking to what I am typing”. Participant 6 also stated that “I do not like to use my phone in public place, I am worried people around me might see my information.” However, some of the participants said they cannot use the screen curtains in public places because people think they do not know how to open their device or they have difficulty using the device.

Overall, the questionnaire answers highlighted participants’ behaviors and indicated that most blind users are aware of security threats, including observation attacks like aural and visual eavesdropping, as well as unauthorized user access. These results contrast findings by Azenkot et al. (2012), who reported that most people with no vision have a lack of awareness related to protecting their private information on touchscreen devices against security threats, such that they do not use security features like screen lock or screen curtains.

#### **6.5.4.2.2 Interview**

We developed approximately 15 qualitative codes to summarize the most relevant findings we learned from each participant, which we clustered into sets of high level themes, including lack of independence, lack of accessibility, finding items, navigation and transportation, visual threat, aural threat, phishing, embarrassment, social barriers (Stigma), and time consuming.

All participants successfully registered and logged in using the BraillePassword, with all participants successfully entering their credentials with the help of text to speech feature and haptic feedback that have been built in the method interface. When the test session was completed, all participants were asked to reflect about their experience while using the BraillePassword, in terms of both strengths and limitations.

In terms of strengths, participants liked that the use of Braille allowed them to type their credentials anywhere on the screen without worrying about activating unwanted objects on the screen. Participant 3 stated that “The thing I have liked best in this method is that it allows me to type anywhere on the smartphone screen using Braille. I prefer the Braille interface more than the traditional authentication because I do not have to find a particular object on the interface.” Likewise, participant 6 stated that “I like it because I can type in one spot on the screen I do not have to spend time looking for a particular key.” Participant 8 said, “I like it where I do not have to move my finger just tap Braille dots and wait for the response.” Interviews also disclosed that using the haptic feedback to enter the password in the registration and login interfaces is very helpful. Participant 4 stated that “Using the vibration is very good, it helps me to know I have entered a digit and increases the privacy.” Similarly, participant 3 stated that “The vibration is very helpful. It informs me when I am entering the digit, but I am just worried when I entered a wrong digit, how can I know. I think you can inform me immediately once I entered the digit whether it is correct or wrong (one by one) that will save my time.”

Most participants also revealed that the use of swiping gestures to move between interfaces and fields is accessible and easy to perform. As participant 4 stated, “it is very easy I just swipe from top to bottom to move between fields. Unlike other interfaces where



I have to listen to all options until I find the object that I want.” Participant 7 stated that “Swiping between fields is easier than locating the fields’ locations on the traditional authentication screen.”

The most important advantage of the BraillePassword is that it presents only one field to be completed on each web application page. All participants liked that, and they reported that it eliminated the need to locate specific authentication areas that need be completed. Participant 7 agreed with that and added “It is great, it provides each authentication field in an independent page and I swipe to transfer between fields. This is unlike the traditional method where I have to find the exact location of each field and the chance of hitting the wrong object on the screen is very high.” Another strength in the BraillePassword, highlighted by most participants, is that it overcomes that navigation problem. Participant 5 reported that “I do not have to find the location of an object on the screen. I can just type anywhere on the screen.”

All participants also liked the protection of the password by eliminating the audio and visual feedbacks and using haptic feedback instead. As Participant 7 stated “I like how it protects the password info, it does not provide any aural or visual feedbacks.” All participants stated that the BraillePassword is more secure than the traditional authentication method, reporting that no one nearby can hear or see the entered password. Participant 5 stated that “BraillePassword does not read out the character and it does not have any visual feedback. It prevents people around me from hearing or seeing what has been typed on the screen. It is like a special code no one will be able to understand it.” Participant 4 agreed and added “The BraillePassword does not read out my password. It reduces the chance of someone hearing my personal data.”

Interviews also revealed some drawbacks of the BraillePassword. Participant 1 and 3 faced difficulty changing the cursor to another location while typing. One limitation addressed by four participants (4, 5, 7 and 9), was that when users tap 7 times, the BraillePassword opens a page that has the forget password steps. Participant 5 stated that “I do not like it when I tap more than six digits accidentally. It opens other screen.” Participant 7 also agreed, stating that “Tapping accidentally 7 times makes me lose the page that I want to type on”.

Most participants also suggested the implementation of BraillePassword interface in the ATM machine in order to reduce the risk of observation attacks. Participant 3 stated that “using the BraillePassword interface in the ATM is very great and secure.” Another suggestion was associated with the password entry strategy: participant 7 suggested that “The app should let me know whether the password digit is wrong or right. By doing so I can correct the password digit before inserting the whole password digits.” Some participants prefer to protect all their credentials and not only their password; thus they suggested to allow them to choose which feedback they want based on their settings. Participant 6 also suggested that the interface “provide an option at the beginning of the web application allowing me to select which feedback I want to receive (aural or vibration feedback) in case I am in public place, I can choose the vibration feedback and I will not annoy any one nearby.” Participant 8 agreed and added “I wish you applied the same concept [haptic feedback] on the other authentication area like the first name field, by doing so no one nearby will be able to know my information.” In addition, participants suggested using other haptic feedback such as tone rather than speech while switching between

numbers, letters, and symbols. This would increase the security level where attackers cannot distinguish which keyboard layout the user in.

During the interview, participants also discussed the strengths and limitations of the traditional authentication method. All participants entered their credentials using the traditional authentication method successfully. Most of the participants like the support of the screen reader to help them navigate through fields in the registration and login interfaces. However, participants did not highlight any particular strength in this technique; for example, Participant 4 noted that “I have been using this method for a long time and I used to it because it is the only available way and we used this method to interact with most interface on touch screen.”

Participants identified several weaknesses associated with the traditional authentication method. One limitation, brought up by three participants, was that the traditional authentication method is slow. Participant 4 stated that “It is not flexible and it does not let me type as fast as I want.” Participant 7 agreed, noting that the traditional authentication method is time consuming and requires many steps to enter a character. Most participants reported that using the traditional authentication interface with the screen reader is annoying. As participant 2 stated, “If I have a headache, I cannot use the phone neither the headphones.” Participant 5 reported that “I tried using the QWERTY keyboard with the screen reader, but it sometimes gets freezing and it is annoying. I am using the magnifier to interact with the screen even though it is slow and requires me to zoom out many times because the smartphone screen is very small.” This indicates that blind users are willing to use other methods that enable them to interact with the screen with less noise while providing only the amount of audio feedback necessary.

Another drawback is that locating a particular area in which to type specific information was very difficult to perform using the traditional authentication method. Participant 8 reported that “Transferring between authentication fields is very hard and confusing.” This is similar to participant 4’s statement, that “The user interface of the traditional authentication method is not efficient and it is really hard to move between fields.”

Participants tended to type particular information in the wrong place, which led to login failure. Three participants discussed the confusion that occurs when they accidentally tap an unwanted link or button. They usually lost their current pages when they accidentally pressed an unwanted object. This error consumes time until the user’s return back to the intended page. Using the traditional authentication method on a web application, blind people faced unexpected distractions, such as other screens popping up or new dialog boxes opening up while they are entering their credentials.

### **6.5.5 Discussion**

Our findings reveal the accessibility and security issues that blind and visually impaired face while authenticating themselves to web applications on touchscreen devices. We also discuss the approaches that blind and visually impaired people have used to overcome the accessibility challenges and reduce the security risks.

The results of the study show that the main causes of security and accessibility issues faced by blind people related to assistive technology like the screen reader. In the traditional authentication application, the user interface provides too many audio notifications to users. This can annoy and overwhelm the user, which might directly affect user’s attention and cause confusion and errors. Participants get annoyed and confused

while the screen reader reads unnecessary movements or web page elements. The screen reader also suddenly freezes and stops responding to users' interactions.

Participants also highlighted their difficulty locating particular authentication areas on the web application interface, with the screen reader inefficiently specifying such locations. Most participants struggled to locate username or password fields in the traditional authentication method, unlike with BraillePassword where they just swipe to move to the intended field. This issue consumes too much time and leads to high levels of confusion for the users. Another accessibility issue that causes confusion for users is unexpected screens or dialogue boxes popping up. These issues require users to wait a long period of time figuring out what the user interface looks like and returning back to the desired interface. Another thing that consumes time is waiting for the screen reader to identify which object is under their finger.

The most common difficulty that participants complained about is inserting CAPTCHAs, which is very difficult and time consuming because the sound is not clear and there is no alternative way to enter the captcha digits. One participant stated that it is easier for her to ask an assistant to enter the CAPTCHA digits for her. This introduces a new direction of research on how to detect captcha image info more easily for blind users. From our findings, most participants prefer to ask for assistance to help them fill out the CAPTCHA section, which introduces a critical privacy and security risk.

Our result demonstrated that most participants are willing to provide their login credentials to others. In fact, all participants reported asking others to help them withdraw money from the ATM machine. This introduces additional privacy challenges related to allowing others to see person bank information, and suggests a new direction of research

on how to improve the accessibility and security of using ATM for blind and visually impaired people. We need also more work on how to reduce the risk of shoulder and video attacks while a blind user is using the ATM Machine. Similarly, participants' feedback indicated another area that represents other privacy concern for blind users, which is when blind users use their bank cards at point of sale machines in a store. Those credit and debit machines are completely based on touchable buttons and do not provide vibration or audio feedbacks to inform the users that they entered a digit.

The majority of participant concerns were about the risks of aural and visual attacks while using smartphone devices; and most use headphones and screen curtains to hide their private information.

The test study session revealed that all participants were able to register to the BraillePassword application successfully without any assistance. The BraillePassword method was perceived as more secure and trustworthy compared to the traditional authentication application user interface. The proposed password entry system was designed mainly to protect users from shoulder and video attacks, and all participants agreed that the BraillePassword keeps their passwords hidden and protected against observation attacks.

We divide participants' fingertip movements into three categories, which are typing in one spot, typing in a vertical line (moving the finger in a vertical line until the enter all the Braille dots), and typing the Braille dots as a matrix (2 columns by 3 rows). Entering six Braille characters that have two or more Braille active dots was clearly more secure, especially if the users interacted with one location on the smartphone screen without moving their finger or if they interacted with the screen in an unorganized shape. In term

of perceived security, all participants reported that using Braille keyboard with the haptic feedback is more secure than the traditional authentication method and the authentication interface in the ATM interface.

Most participants were able to recall their passwords with BraillePassword to a similar degree as their ability to remember passwords in the traditional authentication method. Even though the failure rate to log in using the BraillePassword was slightly more than the traditional method, this is because participants made typing errors while inserting the digit. We know this since they attempted to enter the password next time and succeeded. However, participants described their main challenge while using the BraillePassword as being the concentration required while entering the Braille dots of each character, because entering one dot wrong can make the whole password incorrect. To reduce the problem they encountered, two participants suggested that allowing the application to inform them of the correctness of a password digit directly using haptic feedback, thus reducing the time needed to enter the password again.

All blind and visually impaired users strongly agreed that haptic feedback increased the security and privacy levels of the application. Most of them expressed confidence in using the BraillePassword in public places, believing that no one nearby would be able to track the user's password aurally or visually. At the same time, participants were not requiring to use any tool to protect their passwords like screen curtains or headphones. Based on the participants' feedback, using gestures to enter password reduces the risk of observation attacks.

In addition, the process of entering Braille dots for each character individually increases the entropy of the BraillePassword application, which produces higher levels of

perceived security and privacy. On the other hand, the process of entering Braille dots demands concentration, particularly if participants enter six different characters that contain two or more active dots.

In terms of BraillePassword improvements, one participant suggested using a sensor to detect if there is a person or camera located nearby; once the sensor locates a person or camera, it turns off the screen and stops reading the content. This feature could improve the privacy and security on smartphone devices for blind people against observation attacks, but it would not be practical in some environments, such as an ATM, because it has a camera and there are people around the user. In addition, participants indicated the need to change the interaction technique to reset the password and suggested using a Braille pattern that enable users to access to the rest password page. Using Braille to reset the password will definitely eliminate the chance of opening this page accidentally. To improve the BraillePassword accessibility, participants suggest that when a user enters a password digit, a vibration informs the user whether the entered digit correct or wrong.

Many researchers have tackled the observation attacks that face blind and visually impaired people on smartphone devices (De Luca et al., 2014; Sherman et al., 2014; Ali et al., 2016). Our findings highlighted that BraillePassword was really hard to crack via video attack. Passwords containing Braille characters with more than one active dot make it very complicated to trace the gesture's type and order, as a result it becomes very difficult to guess the whole password. Someone who wants to attack the BraillePassword web application would need to professionally learn Braille characters' patterns, as well as the Braille signs to switch between numbers and between lower and upper cases, as well as remembering the gestures' types that were entered. The user interface design with the



haptic feedback also proved that it allows blind users to enter their passwords safely and prevent others from guessing or tracking their credentials aurally or visually. So, it is the first method that uses Braille patterns with haptic feedback and no aural or visual feedback while entering a password. It is much more secure than passwords in the traditional authentication method, which were easy to guess aurally and visually due to the use of screen reader and the QWERTY keyboard that broadcast users' private information.

The study also indicated that the traditional authentication method is not secure against observation attacks due to the need to locate authentication areas on a screen and the limitations associated with the use of assistive technology with the QWERTY keyboard. Therefore, most blind users tend to choose guessable passwords that are easy to enter and find on the QWERTY keyboard, but also easy to guess or hack. These findings mirrored the results of other studies, such as von Zezschwitz et al. [27].

The main limitation of this study is that the sample is from a single geographic area and very small; therefore, it does not fully represent all vision statuses and ages (e.g., all participants were between 20 and 27 years old). We also did not study those who have limited experience with touchscreen. Our participants described themselves as blind or visually impaired, meaning they can see only the light or things that are very near to their eyes. Thus, it is difficult to generalize our findings to greater population and to understand how functionality might differ for those with no versus limited vision.

In summary, this work has described a novel authentication technique called BraillePassword that uses gestures to encode Braille dots for characters, with an interface that uses haptic feedback and has no buttons. Results revealed that performance with this

technique was at least as good as a traditional method, while being much more resistance to security issues such as password observation attacks.

#### **6.5.6 Study limitation**

The main limitation of this study is that the participants sample is very small and it does not cover all possible variations in vision status and age. It is also limited to two geographic areas (Taif university accessibility center in Saudi Arabia and CNIB center in Canada). As we were only able to recruit participants who are college students (their age range between 20 to 27) from Taif accessibility center and two employees from CNIB center (their age range between 36 to 49). We did not study those who have limited experience with touchscreen. Based on our collective data, our participants described themselves as blind or visually impaired, who can see only the light or things that are very near to their eyes. Thus, it is very difficult to generalize our findings to greater population due to the variations of vision status and the interaction techniques' differences between people with no or low vision.

Designing an accessible tool for this population is extremely challenging due to that it is not easy to conduct usability studies with this population. That is because most of them have problems with transportation and some of them cannot participate without assistants to accompany them to the center to show them the direction. Some of them are very concerned about their safety, so they are not willing to be involved in any extra-curricular activities at the university. Another challenge is that the ways visually impaired people interact with smartphone devices vary due to their different abilities and various degree of

vision. People with multiple disabilities face additional challenges that prevent their participation.

## **6.6 Conclusion and future work**

Smartphone devices have been becoming an indispensable part of the lives of people with visual impairments. Unfortunately, smartphone devices present various privacy and security risks for blind and visually impaired people. For example, using the screen reader software to enter their authentication information causes aural attacks and the QWERTY keyboard causes visual attack. As mentioned above, current web authentication interfaces on smartphone devices are inaccessible and not secure to use for users with vision impairment. Therefore, this paper described BraillePassword as a proof of concept that enables users to authenticate to web application on touchscreens using gestures and haptic feedback to code Braille characters. Results have shown that BraillePassword is an accessible authentication method that is secure and resilient to observation and shoulder attacks even when an attacker observes or camera-records the victim while entering the password. It also allows blind users to enter their credentials without using any protection like headphones or screen curtains. Furthermore, it is implementable on almost any touchscreen device or platform.

Findings have recommended that security software developers design a new authentication method that takes into account the challenges that face people with disabilities. They should also create a new method that tackle the limitations of assistive technology and the size of smartphone device. In addition, designing a user interface for

the authentication method plays an important role in the accessibility and security of the authentication method.

We hope that our research will motivate other researchers to propose and design a security and privacy solutions that meet the needs of the blind and visually impaired people. Some blind people have multiple disabilities, which introduces potential research directions to address and overcome the challenges that meet people with multiple disabilities. Designing technical solutions for people with disabilities demands a strong cross-disciplinary approach leading to collaborative research with technical experts, web application experts, Human Computer Interaction experts and related disabilities centers. In the future, we aim to conduct a user study where users have to recall multiple Braille passwords for different web applications over a long period. We will also strength and refine the BraillePassword interface by modifying the rest password procedures as well providing a unique haptic feedback for each keyboard layout.

## **CHAPTER 7: DISCUSSION AND CONCLUSION**

### **7.1 Discussion**

Input methods are the most heavily relied upon applications in smartphone devices. Touchscreen keyboards are often used to perform a variety of tasks, including typing notes, inserting contacts, or sending emails and messages. Most smartphone devices have a touchscreen keyboard, which is the standard QWERTY keyboard. The QWERTY keyboard follows the same layout as the physical keyboard although the soft QWERTY keyboard still does not reach the same quality and level of performance as the physical keyboard for both blind and sighted people. This is due to the proximity of the keys and small size of the keyboard. Another reason interacting with the touchscreen is especially difficult for blind people is the absence of tangible buttons. These limitations affect blind people more negatively because touching or feeling is the most important way for them to understand things around them.

There have been significant improvements in the accessibility of touchscreen devices in recent years. One of the most significant improvements is the screen reader, such as the Voiceover service. A visually impaired user can touch the screen to identify the screen layout, and the screen reader reads out the item under the user's finger.

Despite these improvements, touchscreen devices and the VoiceOver service pose accessibility and security issues for blind users. Touchscreen devices require a user to visually locate a button on the screen, which is a challenging task for people with limited or no vision (Nicolau et al., 2010; Azenkot et al., 2014). For example, it is difficult for blind people to insert text on a touchscreen, even with audio feedback. The user has to find a particular character on the soft QWERTY keyboard and double tap to insert it. This is

time-consuming, as users have to listen to every character when they explore the screen. The possibility of inserting the wrong character is high because the keys are small and close to each other (Nicolau et al., 2010). They also found that the soft QWERTY keyboard with Voiceover is slow, annoying, and causes errors (2010). As such, there is a need for accessible touchscreen entry methods that reduce the accessibility and security challenges for blind and visually impaired people.

The touchscreen devices and VoiceOver service do not only cause accessibility challenges, they also pose unique privacy and security threats to people with no or low vision (Azenkot et al., 2014; Kuber et al., 2015). The Voiceover service speaks out anything under the user's finger on a screen. Bystanders might hear what has been typed on the screen. This occurs even when a user logs into an online account. The VoiceOver broadcasts user names and passwords to people around blind users. Thus, the audio threat is increased when people use their smartphone devices in public places. Another security threat that emerges from the touchscreen device is that it visualizes the keys and displays the inserted information on the screen. That enables people around a blind person to see the displayed information on a screen (visual threat).

To address the accessibility issues, this research presents eyes-free input methods on touchscreen devices that use gestures on a button-free user interface, which allows blind users to overcome the limitations of button-based input methods. User studies with blind users that examine the developed input methods' accessibility and performance are conducted.

The issue of security threats is tackled in the research by creating an accessible non-visual web authentication method on touchscreen devices. The authentication method

provides no visual and audio feedbacks to reduce privacy and security concerns that blind people encounter during password entry. In addition, the study conducts a user study with blind users that examines how the novel authentication method is resistant to shoulder surfing issue.

This research contributes to the study of input design for visually impaired individuals, and to the approach of input method design more generally. In designing a specific entry method for the visually impaired, a new way of interaction technique and user interface security is introduced. Redesigning problematic input methods based on the practical, everyday lives of the target population by using a Braille-inspired system allows for the user to adapt faster to the touchscreen devices and save time. This research shows that the connection of the everyday activities of those with visual impairments to input methods increases expedience. Such theoretical implications of the research allow future researchers to conceptualize the design of input and authentication methods.

## **7.2 Conclusion**

We presented new methods and user studies that aim to improve the accessibility of input method as well as security of password for blind smartphone device users. In this research, we explain four single gesture methods that uses button free interfaces, which are SingleTapBraille keyboard, BrailleTap calculator, BrailleEnter keyboard and the BraillePassowrd method. SingleTapBraille is a novel nonvisual text input approach for touchscreen devices. With SingleTapBraille, a user enters characters including text, numbers, and punctuation by tapping anywhere on the screen with one finger or a thumb several times based on Braille shapes of characters. The research also developed a non-visual touchscreen calculator, which is called BrailleTap. This calculator allows users to

tap on a screen based on Braille coding to enter numbers and it supports audio feedback to facilitate editing of spelling mistakes. In addition, it allows users to insert operations by swiping vertically to switch between operations. This calculator overcomes the main limitation associated with the button calculator by removing the fixed keys from the calculator layout.

This research also presented the BrailleEnter, which is a button free text entry that enables blind people to enter text on touchscreens based on Braille. This keyboard allows blind people to type more easily on a touchscreen by overcoming most of the limitations of the QWERTY keyboard as well as other proposed keyboards. Users can type based on two types of gesture anywhere on a screen and they do not need to worry about locating a key on a touchscreen. The algorithm that has been used to implement this keyboard does not depend on the coordinates of user interaction, which leads to improved keyboard accuracy, and at the same time, reduces the error rate. It also allows blind users to type with one finger and hold the mobile phone with a single hand.

This research also develops a new web authentication system for blind and visually impaired people, which is called a BraillePassword method. To login using the BraillePassword, a user enters six digits of selected Braille characters informed by haptic feedback (vibration). The BraillePassword provides no aural or visual feedback to eliminate the risk of observation or shoulder attack without any extra fees for special hardware.

We conducted user studies to evaluate the performance of the proposed methods and we compared them to the most common applications used by blind users like soft QWERTY keyboard, Swift Braille keyboard, regular calculator and the standard



authentication system. The button free number and text entry methods were more accurate and accessible than button-based keyboards. Both the SingleTapBraille and BrailleEnter keyboards are more accessible than the button based keyboards. In addition, The BrailleTap calculator was more accurate and faster than the regular calculator. BraillePassword was more secure and resilient against observation attacks than the traditional authentication method. We believe designing accessible input method and secure authentication method on touchscreen devices like smartphone or wearable device is very important to allow users with low or no vision to perform a variety of tasks.

We conducted comparative evaluation studies to examine the developed Braille methods via mixed methods using both qualitative and quantitative methods to obtain deep understanding of research problems as well as to reduce the limitation of using only single method (John, Creswell, 2013). The comparative evaluation results address the most important keyboard and password system features and requirements that help blind people to interact easily with touchscreen devices. The quantitative and qualitative data showed that using single touch approach on button free touchscreen interface to create accessible number and text entries with voice feedback to facilitate editing enable blind and visually impaired users to type easily on a smartphone compared to button based keyboards. In addition, following the same approach with the usage of haptic feedback instead of voice feedback can noticeably reduce the risk of observation attacks and improve the security and privacy of passwords on smartphone devices.

The primary objective of the evaluation studies was to evaluate Braille methods' performance and accessibility. The studies also aim to identify the methods' strengths and limitations, and to obtain users' suggestions regarding the design of accessible techniques

on small touchscreen devices. The comparative evaluation results address the most important keyboard features and requirements that help blind people to interact easily with touchscreen devices.

The input methods that use a single touch approach based on Braille patterns do not restrict users to locate an object on touchscreens. These entry methods rely on the ability to type using their mother language (Braille) which does not require time for them to learn. The proposed methods, which employ a set of accessible gestures that can be easily performed by blind users, lower the rate of errors that occur when users need to switch between layers or do editing. Additionally, entry methods which allow for exploration of the touchscreen that allows people with no vision to interact anywhere on the touchscreen without any concern about hitting undesired objects or opening up another screen enables them to learn the methods in a short time, and allows them to use the methods more easily. Ultimately, the interaction approach that uses a single touch approach based on Braille with haptic feedback provides an accessible and secure web authentication method that is resistant to observation attacks and allows for simple gestures and relatively effortless screen exploration. This research introduces a new interaction technique for input and a secure user interface for web authentication applications.

### **7.3 Future work**

This research unveiled a novel design approach on non-visual keyboards using gestures to encode Braille patterns of characters. This approach makes the keyboard interaction more accessible for blind users where they do not need to locate keys on a screen and they do not need to concern themselves with navigation between keyboard layouts. Understanding

the proposed methods' performance in the long term is area for future work. We also need to test the proposed methods with larger groups of people in order to better understand their strengths and weaknesses. Further, we will try to conduct longitudinal studies to understand long-term behaviors and the improvements of Braille tools performance.

While our research work concentrates mainly on visually impaired people, we believe that the proposed work presents ideas and approaches that will improve the accessibility of small touchscreen devices for people with other disabilities, for example, deaf blind population. Designing technical solutions for people with disabilities requires their feedback from the first stage of design, in order to respond to for the actual needs of this population. In addition, integrating the eyes-free entry approach is a critical concept in developing touchscreen technology that improves people's lives. As touchscreen technology expands, we will have to develop assistive technologies that allow blind users to access a wider range of applications.

Based on the findings from the evaluation studies, we need to improve the performance of Braille keyboards by implementing the predictive text, automatic correction and grade 2 Braille.

The authentication method study revealed the need to design accessible web authentication methods for blind people that allow them to register and sign in independently. It also indicates the necessity to build an accessible password interface for the ATM machine and the retailer point of sale that meets the needs of people with visual impairment. Future work will increase the tools' usefulness by integrating them in different platforms including iPhone, Apple Watch, and iPad. This will make them accessible for intended users for the developed tools can be easily adapted to work on any kind of devices.

We hope our research provides valuable insights for assistive technology researchers to further explore single touch interaction based on Braille.

## APPENDICES

### Appendix A - Background Questionnaire

PART I - PLEASE FILL IN THE FOLLOWING INFORMATION:

1. Age: \_\_\_\_\_
2. Gender: Male Female Other
3. I am presently: Legally blind Visually Impaired Other: \_\_\_\_\_
4. Do you know Braille language? Yes No
5. What type of Braille-based devices you use?
6. Do you use a touchscreen device? Yes No
7. If yes, how long have you been using a touchscreen device?
8. What are you using now to enter text on a touchscreen?
9. What are the main difficulties that you have faced while using the above method to enter text?
10. If you are using a normal keyboard, finding a specific letter, number or punctuation is difficult.
  - Strongly agree
  - Agree
  - Neither agree nor disagree
  - Disagree
  - Strongly disagree
11. How long do you usually take to write a message contains 20 words?
  - Once minute
  - Two minutes
  - Three minutes
  - More than three minutes
12. How often do you send short messages (SMS)?
  - Everyday
  - Several times a month
  - Never
13. Do you send messages while you are walking?

- yes
  - no
14. If yes, how do you manage holding the mobile phone while you are walking or other hand is busy?
15. How do you hold the smartphone device when you want to type a text?
16. How do you tap on a touch screen to enter a text? Do you use only your thumb or your point finger?
17. It is difficult to switch from letter keyboard to number keyboard.
- Strongly agree
  - Agree
  - Neither agree nor disagree
  - Disagree
  - Strongly disagree
18. It is difficult to type a small or capital letter.
- Strongly agree
  - Agree
  - Neither agree nor disagree
  - Disagree
  - Strongly disagree
19. Using VoiceOver service is a good solution to make sure you chose the correct letter.
- Strongly agree
  - Agree
  - Neither agree nor disagree
  - Disagree
  - Strongly disagree

## Appendix B – Post Questionnaire of SingleTapBraille

Part1: Describe how you feel about each statement.		
<b>1 = strongly disagree, 2 = somewhat disagree, 3 = neutral, 4 = somewhat agree, 5 = strongly agree</b>		
<b>PERCEIVED USEFULNESS</b>		<b>Scale</b> 1 2 3 4 5
1.	Using the keyboard to enter text would enable me to accomplish typing tasks more quickly.	
2.	The keyboard would enable me to enter text more independently.	
3.	Using the keyboard would improve my typing performance.	
4.	Using the keyboard to enter text would increase my productivity.	
5.	Using the application would enhance my effectiveness on entering text.	
6.	Using the application would make it easier to enter texts and send messages.	
7.	I would find the keyboard useful in general.	
<b>PERCEIVED EASE OF USE</b>		
8.	It was easy to use the keyboard to enter what I wanted to text.	
9.	My interaction with the keyboard was clear and understandable.	
10.	It would be easy for me to become skillful at using the keyboard.	
11.	The letters, numbers, and punctuation was easy to type.	
12.	The interaction techniques was easy to understand	
<b>PERCEIVED EASE OF LEARNING</b>		
13.	Learning to operate the keyboard would be easy for me.	
14.	Editing text was clear.	
15.	Switching between keyboard was clear	
16.	Remembering the gestures and touch commands was easy.	
17.	Performing tasks like entering text, making capital and small letters were straightforward	
18.	Voice feedback were helpful to define the gesture function	
19.	I found my skills at using the app improved with practice	

### **Appendix C: Interview of SingleTapBraille**

1. Do you feel that you successfully entered text using the improved SingleTapBraille/ QWERTY keyboard?
2. Did you find the interaction with touchscreen without button or looking for a specific location was more accessible? Why or why not?
3. What is your opinion about the command for opening the letter keyboard, number keyboard and punctuation keyboard? Did you find these commands accurate and easy to perform? Why or why not?
4. Did you feel your typing was faster than usual? Why or why not?
5. Did the Braille application assist you to send SMS messages easily? Why or why not?
6. What are the application's advantages?
7. What are the application's disadvantages?
8. What is your overall impression of the improved SingelTapBraille/ QWERTY keyboard?
9. Was the application user interface easy to use?
10. Did you have any difficulties when you used the application? What were they and why do you think that you had this/these problems?
11. Do you have any suggestions to fix these problems?
12. What do you think the benefits of this system are?
13. What overall suggestions do you have for the system?
14. Did this system address your typing concerns? How and what didn't it address?

**Participant Number:** ..... **Date:** .....



## Appendix D: Letters of approval from Dalhousie Research Ethics Board for the SingleTapBraille study



### Social Sciences & Humanities Research Ethics Board

Letter of February 23, 2016

Ms Mrim Alnfai  
Computer Science\Computer Science

Dear Mrim,

**REB #:** 2016-3785 **Project Title:** Single-Tap Braille: Single Touch Braille Text Entry Approach Based on Braille Patterns

**Effective Date:** February 22, 2016 **Expiry Date:** February 22, 2017

The Social Sciences & Humanities Research Ethics Board has reviewed your application for research involving humans and found the proposed research to be in accordance with the Tri-Council Policy Statement on *Ethical Conduct for Research Involving Humans*. This approval will be in effect for 12 months as indicated above. This approval is subject to the conditions listed below which constitute your on-going responsibilities with respect to the ethical conduct of this research.

Sincerely,

Dr. Karen Beazley, Chair

#### Post REB Approval: On-going Responsibilities of Researchers

After receiving ethical approval for the conduct of research involving humans, there are several ongoing responsibilities that researchers must meet to remain in compliance with University and Tri-Council policies.

##### 1. Additional Research Ethics approval

Prior to conducting any research, researchers must ensure that all required research ethics approvals are secured (in addition to this one). This includes, but is not limited to, securing appropriate research ethics approvals from: other institutions with whom the PI is affiliated; the research institutions of research team members; the institution at which participants may be recruited or from which data may be collected; organizations or groups (e.g. school boards, Aboriginal communities, correctional services, long-term care facilities, service agencies and community groups) and from any other responsible review body or bodies at the research site

##### 2. Reporting adverse events

Any significant adverse events experienced by research participants must be reported **in writing** to Research Ethics **within 24 hours** of their occurrence. Examples of what might be considered “significant” include: an emotional breakdown of a participant during an interview, a negative physical reaction by a participant (e.g. fainting, nausea, unexpected pain, allergic reaction), report by a participant of some sort of negative repercussion from their participation (e.g. reaction of spouse or employer) or complaint by a participant with respect to their participation. The above list is indicative but not all-inclusive. The written report must include details of the adverse event and actions taken by the researcher in response to the incident.

### 3. Seeking approval for protocol / consent form changes

Prior to implementing any changes to your research plan, whether to the protocol or consent form, researchers must submit a description of the proposed changes to the Research Ethics Board for review and approval. This is done by completing an Amendment Request (available on the website). Please note that no reviews are conducted in August.

### 4. Submitting annual reports

Ethics approvals are valid for up to 12 months. Prior to the end of the project’s approval deadline, the researcher must complete an Annual Report (available on the website) and return it to Research Ethics for review and approval before the approval end date in order to prevent a lapse of ethics approval for the research. Researchers should note that no research involving humans may be conducted in the absence of a valid ethical approval and that allowing REB approval to lapse is a violation of University policy, inconsistent with the TCPS (article 6.14) and may result in suspension of research and research funding, as required by the funding agency.

### 5. Submitting final reports

When the researcher is confident that no further data collection or participant contact will be required, a Final Report (available on the website) must be submitted to Research Ethics. After review and approval of the Final Report, the Research Ethics file will be closed.

6. Retaining records in a secure manner Researchers must ensure that both during and after the research project, data is securely retained and/or disposed of in such a manner as to comply with confidentiality provisions specified in the protocol and consent forms. This may involve destruction of the data, or continued arrangements for secure storage. Casual storage of old data is not acceptable.

It is the Principal Investigator’s responsibility to keep a copy of the REB approval letters. This can be important to demonstrate that research was undertaken with Board approval, which can be a requirement to publish (and is required by the Faculty of Graduate Studies if you are using this research for your thesis).

Please note that the University will securely store your REB project file for 5 years after the study closure date at which point the file records may be permanently destroyed.

7. Current contact information and university affiliation The Principal Investigator must inform the Research Ethics office of any changes to contact information for the PI (and supervisor, if appropriate), especially the electronic mail address, for the duration of the REB approval. The PI must inform Research Ethics if there is a termination or interruption of his or her affiliation with Dalhousie University.

8. Legal Counsel The Principal Investigator agrees to comply with all legislative and regulatory requirements that apply to the project. The Principal Investigator agrees to notify the University Legal Counsel office in the event that he or she receives a notice of non-compliance, complaint or other proceeding relating to such requirements. 9. Supervision of students Faculty must ensure that students conducting research under their supervision are aware of their responsibilities as described above, and have adequate support to conduct their research in a safe and ethical manner.

## Appendix E - Background Questionnaire of BrailleEnter keyboard

Thank you for your participation. Before we begin, we would like to get to know a little bit more about you and your current use of touchscreen keyboards. Please take a few minutes to answer this questionnaire.

### PART I - PLEASE FILL IN THE FOLLOWING INFORMATION:

1. Age: \_\_\_\_\_
2. Gender: Male Female Other
3. I am presently: Legally blind Visually Impaired Other: \_\_\_\_\_
4. Do you know Braille language? Yes No
5. What type of Braille-based devices you use?
6. Do you use a touchscreen device? Yes No
7. If yes, how long have you been using a touchscreen device?
8. Which types of phone have you used?
9. What are you using now to enter text on a touchscreen?
10. What are the main difficulties that you have faced while using the above method to enter text?
11. How do you hold the smartphone device when you want to type a text?
12. How do you tap on a touch screen to enter a text? Do you use only your thumb or your point finger?
13. Which tasks have you used touchscreen keyboard to perform? (Check all that apply)  
( ) Adding contacts' information , ( ) Typing note, ( ) Texting, ( ) Emailing , ( ) Other

## Appendix F: Letters of approval from Dalhousie Research Ethics Board for the BrailleEnter study



### Social Sciences & Humanities Research Ethics Board

Letter of September 21, 2017

Mrim Alnfai  
Computer Science\Computer Science

Dear Mrim,

**REB #:** 2017-4230 **Project Title:** Evaluation of touchscreen authentication methods for blind people

**Effective Date:** September 21, 2017 **Expiry Date:** September 21, 2018

The Social Sciences & Humanities Research Ethics Board has reviewed your application for research involving humans and found the proposed research to be in accordance with the Tri-Council Policy Statement on *Ethical Conduct for Research Involving Humans*. This approval will be in effect for 12 months as indicated above. This approval is subject to the conditions listed below which constitute your on-going responsibilities with respect to the ethical conduct of this research.

Sincerely,

Dr. Karen Beazley, Chair

#### Post REB Approval: On-going Responsibilities of Researchers

After receiving ethical approval for the conduct of research involving humans, there are several ongoing responsibilities that researchers must meet to remain in compliance with University and Tri-Council policies.

##### 1. Additional Research Ethics approval

Prior to conducting any research, researchers must ensure that all required research ethics approvals are secured (in addition to this one). This includes, but is not limited to, securing appropriate research ethics approvals from: other institutions with whom the PI is affiliated; the research institutions of research team members; the institution at which participants may be recruited or from which data may be collected; organizations or groups (e.g. school boards, Aboriginal communities, correctional services, long-term care facilities, service agencies and community groups) and from any other responsible review body or bodies at the research site

## 2. Reporting adverse events

Any significant adverse events experienced by research participants must be reported **in writing** to Research Ethics **within 24 hours** of their occurrence. Examples of what might be considered “significant” include: an emotional breakdown of a participant during an interview, a negative physical reaction by a participant (e.g. fainting, nausea, unexpected pain, allergic reaction), report by a participant of some sort of negative repercussion from their participation (e.g. reaction of spouse or employer) or complaint by a participant with respect to their participation. The above list is indicative but not all-inclusive. The written report must include details of the adverse event and actions taken by the researcher in response to the incident.

## 3. Seeking approval for protocol / consent form changes

Prior to implementing any changes to your research plan, whether to the protocol or consent form, researchers must submit a description of the proposed changes to the Research Ethics Board for review and approval. This is done by completing an Amendment Request (available on the website). Please note that no reviews are conducted in August.

## 4. Submitting annual reports

Ethics approvals are valid for up to 12 months. Prior to the end of the project’s approval deadline, the researcher must complete an Annual Report (available on the website) and return it to Research Ethics for review and approval before the approval end date in order to prevent a lapse of ethics approval for the research. Researchers should note that no research involving humans may be conducted in the absence of a valid ethical approval and that allowing REB approval to lapse is a violation of University policy, inconsistent with the TCPS (article 6.14) and may result in suspension of research and research funding, as required by the funding agency.

## 5. Submitting final reports

When the researcher is confident that no further data collection or participant contact will be required, a Final Report (available on the website) must be submitted to Research Ethics. After review and approval of the Final Report, the Research Ethics file will be closed.

## 6. Retaining records in a secure manner

Researchers must ensure that both during and after the research project, data is securely retained and/or disposed of in such a manner as to comply with confidentiality provisions specified in the protocol and consent forms. This may involve destruction of the data, or continued arrangements for secure storage. Casual storage of old data is not acceptable.

It is the Principal Investigator’s responsibility to keep a copy of the REB approval letters. This can be important to demonstrate that research was undertaken with Board approval, which can be a requirement to publish (and is required by the Faculty of Graduate Studies if you are using this research for your thesis).

Please note that the University will securely store your REB project file for 5 years after

the study closure date at which point the file records may be permanently destroyed.

8. Current contact information and university affiliation The Principal Investigator must inform the Research Ethics office of any changes to contact information for the PI (and supervisor, if appropriate), especially the electronic mail address, for the duration of the REB approval. The PI must inform Research Ethics if there is a termination or interruption of his or her affiliation with Dalhousie University.
9. Legal Counsel The Principal Investigator agrees to comply with all legislative and regulatory requirements that apply to the project. The Principal Investigator agrees to notify the University Legal Counsel office in the event that he or she receives a notice of non-compliance, complaint or other proceeding relating to such requirements.
9. Supervision of students Faculty must ensure that students conducting research under their supervision are aware of their responsibilities as described above, and have adequate support to conduct their research in a safe and ethical manner.

**Appendix G – Post Questionnaire (Lewis Questionnaire)**

Please rate the method you just used. Circle the number that best represents your judgment. And why do you choose that number 1. Very Likely, 2. Likely, 3. Somewhat likely, 4. Neither likely, 5. Somewhat unlikely, 6. Unlikely, 7. Very unlikely

Easy to insert letters	1--2--3--4--5--6--7	Hard to insert letters
Keyboard interaction method acceptable	1--2--3--4--5--6--7	Keyboard interaction method unacceptable
Easy to type fast	1--2--3--4--5--6--7	Hard to type fast
Easy to type accurately	1--2--3--4--5--6--7	Hard to type accurately
Easy to learn how to type letters	1--2--3--4--5--6--7	Hard to learn how to type letters
Easy to type	1--2--3--4--5--6--7	Hard to type
Using one finger is accessible	1--2--3--4--5--6--7	Using one finger is inaccessible
Audio feedback is accessible	1--2--3--4--5--6--7	Audio feedback is inaccessible
Editing is accessible	1--2--3--4--5--6--7	Editing is inaccessible



**Appendix H: Interview of BrailleEnter keyboard**

1. Do you feel that you successfully entered text using the BrailleEnter/swift keyboard approach? Why or why not?
2. Do you like the keyboard? Why or why not?
3. Did you find the interaction with touchscreen without buttons or looking for a specific location was more accessible? Why or why not?
4. Did you find using one finger to interact with touchscreen devices accessible? Why or why not?
5. What are the keyboard's advantages?
6. What are the keyboard's disadvantages?
7. What is your overall impression of the keyboard prototype?
8. Did you have any difficulties when you used the keyboard? What were they and why do you think you had this/these problems?
9. Do you have any suggestions to fix these problems?
10. What do you think the benefits of this keyboard are? Where do you suggest us to integrate this keyboard?

**Participant Number:** .....**Date:** .....

## Appendix I - Background Questionnaire of BraillePassword

Thank you for your participation. Before we begin, we would like to get to know a little bit more about you and your current use of touchscreen devices. Please take a few minutes to answer this questionnaire.

### PART I - PLEASE FILL IN THE FOLLOWING INFORMATION:

1. Age: \_\_\_\_\_
2. Gender:  Male  Female  Other
3. I am presently:  Legally blind  Visually impaired  Other:
4. Do you know Braille language?  Yes  No
5. What type of Braille-based devices you use?
6. Do you use a touchscreen device?  Yes  No
7. If yes, how long have you been using a touchscreen device?
8. Which types of phone have you used? (single- choice: iPhone, Android, Windows Phone, Other, I don't use a smartphone)
9. How do you log-in to a web application, such as twitter or your bank account? (using the QWERTY keyboard, other Braille keyboard)
10. What are the authentication methods have you used?  
(Biometric authentication method, graphical authentication method, alphanumeric authentication method).
11. What are the main difficulties that you have faced while registering or login to web applications, such as Facebook, Twitter or bank account?
12. How do you secure your private information (passwords) while login to your account on a smartphone device in public place?
13. Do you access your private information, such as bank account, email communications, social networking sites, and location-tracking applications such as Four Square from your smartphone device?
14. Do you store a wealth of private and personal information on your device?
15. Do you open your bank application from your phone?
16. Which authentication features do you use to protect your information on your device?

17. Do you enter your password every time you login to a web application (such as Facebook and Netflix) or is the password already stored in the application so you do not have to enter the password repeatedly?
18. Do you have any concern about aural eavesdropping? What did you use to reduce the aural threat? (headphones, ...)
19. Do you have any concern about visual eavesdropping? What did you use to reduce the visual threat? (screen curtains, ...)

**Appendix J: Interview of BraillePassword**

1. Do you feel that you successfully registered and logged in using the BraillePassword /traditional authentication method? Why or why not?
2. What are the BraillePassword/traditional authentication method's advantages?
3. What are the BraillePassword/traditional authentication method's disadvantages?
4. What is your overall impression of the BraillePassword/traditional authentication method prototype?
5. Did you have any difficulties when you used the BraillePassword/traditional authentication method? What were they and why do you think you had this/these problems?
6. Do you have any suggestions to fix these problems?
7. What do you think the benefits of this BraillePassword are?
8. Which method do you think you are more secure while using it? Why?
9. Which method do you think would reduce the chance of having observation attacks? Why?

**Participant Number:** ..... **Date:** .....

**Appendix K: Letters of approval from Dalhousie Research Ethics Board for the BraillePassword study**



**Social Sciences & Humanities Research Ethics Board**

**Letter of Approval** September 21, 2017

Mrim Alnfai

Computer Science\Computer Science

Dear Mrim,

**REB #:** 2017-4230 **Project Title:** Evaluation of touchscreen authentication methods for blind people

**Effective Date:** September 21, 2017 **Expiry Date:** September 21, 2018

The Social Sciences & Humanities Research Ethics Board has reviewed your application for research involving humans and found the proposed research to be in accordance with the Tri-Council Policy Statement on *Ethical Conduct for Research Involving Humans*. This approval will be in effect for 12 months as indicated above. This approval is subject to the conditions listed below which constitute your on-going responsibilities with respect to the ethical conduct of this research.

Sincerely,

Dr. Karen Beazley, Chair

**Post REB Approval: On-going Responsibilities of Researchers**

After receiving ethical approval for the conduct of research involving humans, there are several ongoing responsibilities that researchers must meet to remain in compliance with University and Tri-Council policies.

**1. Additional Research Ethics approval**

Prior to conducting any research, researchers must ensure that all required research ethics approvals are secured (in addition to this one). This includes, but is not limited to, securing appropriate research ethics approvals from: other institutions with whom the PI is affiliated; the research institutions of research team members; the institution at which participants may be recruited or from which data may be collected; organizations or groups (e.g. school boards, Aboriginal communities, correctional services, long-term care facilities, service agencies and community groups) and from any other responsible review

body or bodies at the research site

## 2. Reporting adverse events

Any significant adverse events experienced by research participants must be reported **in writing** to Research Ethics **within 24 hours** of their occurrence. Examples of what might be considered “significant” include: an emotional breakdown of a participant during an interview, a negative physical reaction by a participant (e.g. fainting, nausea, unexpected pain, allergic reaction), report by a participant of some sort of negative repercussion from their participation (e.g. reaction of spouse or employer) or complaint by a participant with respect to their participation. The above list is indicative but not all-inclusive. The written report must include details of the adverse event and actions taken by the researcher in response to the incident.

## 3. Seeking approval for protocol / consent form changes

Prior to implementing any changes to your research plan, whether to the protocol or consent form, researchers must submit a description of the proposed changes to the Research Ethics Board for review and approval. This is done by completing an Amendment Request (available on the website). Please note that no reviews are conducted in August.

## 4. Submitting annual reports

Ethics approvals are valid for up to 12 months. Prior to the end of the project’s approval deadline, the researcher must complete an Annual Report (available on the website) and return it to Research Ethics for review and approval before the approval end date in order to prevent a lapse of ethics approval for the research. Researchers should note that no research involving humans may be conducted in the absence of a valid ethical approval and that allowing REB approval to lapse is a violation of University policy, inconsistent with the TCPS (article 6.14) and may result in suspension of research and research funding, as required by the funding agency.

## 5. Submitting final reports

When the researcher is confident that no further data collection or participant contact will be required, a Final Report (available on the website) must be submitted to Research Ethics. After review and approval of the Final Report, the Research Ethics file will be closed.

6. Retaining records in a secure manner Researchers must ensure that both during and after the research project, data is securely retained and/or disposed of in such a manner as to comply with confidentiality provisions specified in the protocol and consent forms. This may involve destruction of the data, or continued arrangements for secure storage. Casual storage of old data is not acceptable.

It is the Principal Investigator’s responsibility to keep a copy of the REB approval letters. This can be important to demonstrate that research was undertaken with Board approval, which can be a requirement to publish (and is required by the Faculty of Graduate Studies if you are using this research for your thesis).

Please note that the University will securely store your REB project file for 5 years after the study closure date at which point the file records may be permanently destroyed.

7. Current contact information and university affiliation The Principal Investigator must inform the Research Ethics office of any changes to contact information for the PI (and supervisor, if appropriate), especially the electronic mail address, for the duration of the REB approval. The PI must inform Research Ethics if there is a termination or interruption of his or her affiliation with Dalhousie University.

8. Legal Counsel The Principal Investigator agrees to comply with all legislative and regulatory requirements that apply to the project. The Principal Investigator agrees to notify the University Legal Counsel office in the event that he or she receives a notice of non-compliance, complaint or other proceeding relating to such requirements. 9. Supervision of students Faculty must ensure that students conducting research under their supervision are aware of their responsibilities as described above, and have adequate support to conduct their research in a safe and ethical manner.

## REFERENCES

- American Foundation for the Blind. (2017). Facts and Figures on Adults with Vision Loss. Retrieved from <http://www.afb.org/info/blindness-statistics/adults/facts-and-figures/235>
- Accessibility. (2016). *Accessibility features*. Retrieved from <http://www.lenovo.com/lenovo/us/en/accessibility/>
- Authentication technologies. (2009). *Authorization. Recognition. Verification. identification. screening*. <http://biometrics.pbworks.com/w/page/14811351/authentication%20technologies#limitationsofbiometrics>
- AppleVis. (2014). A Guide to Braille Screen Input on iOS. Retrieved from <https://www.applevis.com/guides/Braille-ios-voiceover/guide-Braille-screen-input-ios>
- Alnfiai, M., Sampalli, S. (2016a). 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), Procedia, Computer Science, 248-255. ISSN 1877- 0509, <http://dx.doi.org/10.1016/j.procs.2016.08.038>.
- Alnfiai, M. and Sampalli, S. (2016b). An Evaluation of SingleTapBraille Keyboard: A Text Entry Method that Utilizes Braille Patterns on Touchscreen Devices. In Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '16). ACM, New York, NY, USA, 161-169. DOI: <http://dx.doi.org/10.1145/2982142.2982161>
- Alnfiai, M., Sampalli, S. (2017a). BrailleTap Calculator. *The 19th International Conference on Human-Computer Interaction. HCI International 2017*. Vancouver, Canada, 9 - 14 July 2017, pp. 9 - 14.
- Alnfiai, M., Sampalli, S. (2017b). BrailleEnter: A Touch Screen Braille Text Entry Method for the Blind. The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017), Procedia, pp.257-264.
- Alnfiai, M., Sampalli, S. (2017c). An Evaluation of the BrailleEnter Keyboard: An Input Method Based on Braille Patterns for Touchscreen Devices. International Conference on Computer and Applications (ICCA'17), Dubai, pp.107-119.
- Alnfiai, M., Sampalli, S. (2017d). Improved SingleTapBraille: Developing a single tap text entry method based on Grade 1 and 2 braille encoding". *The International Journal of Ubiquitous Systems and Pervasive Networks (JUSPN)*, pp.1-9.



- Alnfiai, M., Sampalli, S. (2018). BraillePassword: Accessible Web Authentication Technique on Touchscreen Devices. *Journal of Ambient Intelligence and Humanized Computing*. Springer, pp.1-26. DOI: 10.1007/s12652-018-0860-x
- Ali. (2015). Sequential Gestural Passcodes on Google Glass. The 17th International ACM SIGACCESS Conference on Computers and Accessibility - Student Research Competition (ASSETS-SRC 2015), Lisbon, Portugal, October 26-28, 2015 <http://dx.doi.org/10.1145/2700648.2811326>
- Ali, A., Kuber, R., & Aviv, A. J. (2016). Developing and evaluating a gestural and tactile mobile interface to support user authentication. *Proceedings of the iConference*, 2016. doi:10.9776/16141.
- AlBanna, M. (2016). SwiftBraille-Android Braille Soft Keyboards. Retrieved from [www.mbanna.info/swiftBraille-soft-keyboard/](http://www.mbanna.info/swiftBraille-soft-keyboard/)
- Azenkot, S., Ladner, R., Wobbrock, J., Borning, Alan, Landay, James, & Levow, Gina-Anne. (2014). Eyes-Free Input on Mobile Devices, *ProQuest Dissertations and Theses*.
- Azenkot, S., J., Wobbrock, O., Prasain, S., and Ladner., R. E. 2012. Input finger detection for nonvisual touch screen text entry in Perkinput. In *Graphics Interface*.
- Azenkot, S. Rector, K., Ladner, R., and Wobbrock, J. 2012. PassChords: secure multi-touch authentication for blind people. In *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '12)*. ACM, New York, NY, USA, 159-166. DOI=<http://dx.doi.org/10.1145/2384916.2384945>
- Azenkot, S., Bennett, C., and Ladner, R. (2013). DigiTaps: eyes-free number entry on touchscreens with minimal audio feedback, *Proceedings of the 26th annual ACM symposium on User interface software and technology*, October 08-11, 2013, St. Andrews, Scotland, United Kingdom [doi>10.1145/2501988.2502056]
- Bigham, J., P., & Cavender, A., C. (2009). Evaluating existing audio CAPTCHAs and an interface optimized for non-visual use. *Proceedings of the 27<sup>th</sup> International Conference on Human factors in Computing Systems*, p.1829, 2009.
- Burnett, M. (2006). Perfect Password: Selection, Protection, Authentication. Syngress, 2005, pp. 109–112, the password list is available at: [http:// boingboing.net/2009/01/02/top-500-worst-passwo.html](http://boingboing.net/2009/01/02/top-500-worst-passwo.html).

- Bonner, M., N., Brudvik, J., T., Abowd, G., D. & Edwards., K. (2010). No-look notes: accessible eyes-free multi-touch text entry. Proceedings of the 8th international conference on Pervasive Computing, Springer, May 17-20, 2010, Helsinki, Finland [doi>10.1007/978-3-642-12654-3\_24]
- BrailleEasy. (2018). Retrieved from <https://itunes.apple.com/us/app/Brailleeasy/id1124391970?mt=8>
- Caute A., & Woolf, C.(2016). Using voice recognition software to improve communicative writing and social participation in an individual with severe acquired dysgraphia: An experimental single-case therapy study. *Aphasiology*, 30(2-3), 2016; 245-268.
- CNIB. (2001). About the Braille System. Retrieved from <http://www.cnib.ca/en/living/Braille/Braille-system/Pages/default.aspx>
- Cassidy, B., Cockton, G., & Coventry, L. “A haptic ATM interface to assist visually impaired users,” Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 1–8, 2013.
- Dhamija, R., Perrig, A., Deja. (2000). A user study using images for authentication. In Proc. USENIX Security Symposium, pages 45–58, Berkeley, CA, USA, 2000. USENIX Association.
- De Luca, A., von Zezschwitz, E., & Hußmann, H. (2009). Vibrapass: secure authentication based on shared lies. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 913–916.
- De Luca, A., Harbach, M., von Zezschwitz, E., et al. (2014). Now you see me, now you don't: protecting smartphone authentication from shoulder surfers. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). ACM, 2937-2946.
- Dhamija, R. and Perrig, A., (2000), “Déjà Vu: A User Study Using Images for Authentication”, in Proceedings of the 9th USENIX Security Symposium, Denver, Colorado, USA.
- Eiband, M., Khamis, M., von Zezschwitz, E., Hussmann, H., Alt, F. (2017). Understanding shoulder surfing in the wild: stories from users and observers. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 4254–4265.
- Frey, B., Southern, C., and Romero. M. (2011). BrailleTouch: mobile texting for the visually impaired. In *Universal Access in Human-Computer Interaction. Context Diversity*. Springer,19-25.

- Fritsch, L., Fuglerud, K., Solheim, I. (2010). Towards inclusive identity management. *Identity in the Information Society Online First™*, 7 October 2010:1–24.
- Guerreiro, T., P. Lagoa, P., Santana, P., Goncalves, D., and Jorge, J. (2008). NavTap and BrailleTap: non-visual texting interfaces. In *Resna*.
- Helkala, K. (2012). Disabilities and authentication methods: Usability and security. *Seventh International Conference on Availability, Reliability and Security*, pp. 327–334.
- Heni S., Abdallah W., Archambault D., Uzan G., Bouhleb M.S. (2016) An Empirical Evaluation of MoonTouch: A Soft Keyboard for Visually Impaired People. In: Miesenberger K., Bühler C., Penaz P. (eds) *Computers Helping People with Special Needs. ICCHP 2016. Lecture Notes in Computer Science*, vol 9759. Springer, Cham
- He, L., Wan, Z., Findlater, L., Froehlich, J. (2017). TacTILE: A Preliminary Toolchain for Creating Accessible Graphics with 3D-Printed Overlays and Auditory Annotations ASSETS '17, October 29–November 1, 2017, Baltimore, MD, USA
- Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.* 1098–1102.
- Hesselmann, T., Heuten, W., & Boll, S. (2011). Tap2Count. *Proc. ITS '11*, New York, NY: ACM Press (2011), 256–257.
- Holman, J., Lazar, J., and Feng, J. (2008). Investigating the Security-related Challenges of Blind Users on the Web. In P.L. BSc, J.C.M., CEng MIEE and P.R.M. Ceng, eds., *Designing Inclusive Futures*. Springer London, 129–138.
- iOS Braille. (2018). Retrieved from <https://www.applevis.com/guides/Braille- ios-voiceover/guide-Braille-screen-input-ios>
- John, W., C. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, Los Angeles, CA.
- Jalaliniya, S., Mardanbegi, D., Sintos, I., and Garcia, D. (2015). EyeDroid: an open source mobile gaze tracker on Android for eyewear computers. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers (UbiComp/ISWC'15 Adjunct)*. ACM, New York, NY, USA, 873–879. DOI=<http://dx.doi.org/10.1145/2800835.2804336>

- Kuber, R., Sharma, S. (2010). Toward tactile authentication for blind users. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility (ASSETS '10)*. ACM, New York, NY, USA, 289-290. DOI=<http://dx.doi.org/10.1145/1878803.1878875>
- Kuber, R. & Sharma, S. (2012). Developing an Extension to an Existing Tactile Authentication Mechanism to Support Non-Visual Interaction. In *proceedings of IASTED Conference on Human-Computer Interaction*, Baltimore, USA, 190-198.
- Kane, S., Jayant C, Wobbrock, J, Ladner, R.(2009). Freedom to roam: a study of mobile device adoption and accessibility for people with visual and motor disabilities. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*;115–122.
- Karlson, A. K., Bederson, B. B., Contreras-Vidal, J. L., (2006). Understanding Single-handed Mobile Device Interaction. University of Maryland. HCIL-2006-02. Martin, G.L., 1988.
- Kane, S. K., Wobbrock, J. O., & Ladner, R. E. (2011). *Usable gestures for blind people: Understanding preference and performance*. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, May 07-12, 2011, Vancouver, BC, Canada [doi>10.1145/1978942.1979001]
- Keane, J. (2016). Facial Recognition Apps Are Leaving Blind People Behind. Face-scanning apps are the latest trend in biometrics, but do they work for people with sight issues?[https://motherboard.vice.com/en\\_us/article/facial-recognition-apps-are-leaving-blind-people-behind](https://motherboard.vice.com/en_us/article/facial-recognition-apps-are-leaving-blind-people-behind)
- Keeble. (2018). Accessible keyboard. Retrieved from <https://itunes.apple.com/ca/app/keeble-accessible-keyboard/id918497054?mt=8>
- Lowry, R. (2005). Concepts and applications of inferential statistics [Electronic Version]. Web Site:<http://faculty.vassar.edu/lowry/webtext.html>.
- Lin, F. X., Ashbrook, D., and White. S. (2011). Rhythmlink: securely pairing i/o-constrained devices by tapping. In *Proc. UIST'11*, pages 263–272, New York, NY, USA, 2011. ACM.
- Lee, S., Hong, S. H., Jeon, J. W., Choi, H. G., and Choi, H., Design of Chording Gloves as a Text Input Device, in *Computer Human Interaction – Lecture Notes in Computer Science*, 3101/2004 (2004), 201- 210.
- Ladner, R., Kane, S., Wobbrock, J.(2011). Usable gestures for blind people: understanding preference and performance. In *Proceedings of the 2011 annual conference on Human factors in computing systems*. ACM, 2011.

- Marques, D., Carrico, L., Guerreiro, T. (2015). Assessing Inconspicuous Smartphone Authentication for Blind People. Retrieved from <https://arxiv.org/abs/1506.00930>
- MBraille. (2018). Accessible keyboard. Retrieved from <https://itunes.apple.com/ca/app/mBraille/id639199558?mt=8>
- Michael, T., Sirivianos, G., Solis, J., Tsudik, G., Uzun, E. (2006). Loud and Clear: Human-Verifiable Authentication Based on Audio, Proceedings of the 26th IEEE International Conference on Distributed Computing Systems, p.10, July 04-07, [doi>10.1109/ICDCS.2006.52]
- Mudholkar, S., Shende, p., Sarode, M. (2012). Biometrics authentication technique for intrusion detection systems using fingerprint recognition. International Journal of Computer Science, Engineering and Information Technology (IJCEIT), Vol.2, No.1, February 2012 , DOI : 10.5121/ijceit.2012.2106
- MacKenzie, S., William, R. and Helga, J. (2011). 1 thumb, 4 buttons, 20 words per minute: design and evaluation of H4-writer. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11). ACM, New York, NY, USA, 471-480. DOI=<http://dx.doi.org/10.1145/2047196.2047258>
- Mattheiss, E., Georg, R., Johann, S., Markus, G., Schrammel, J., Garschall, M., & Tscheligi M. (2015). EdgeBraille: Braille-based text input for touch devices. *Journal of Assistive Technologies*, 9(3), 2015; 147–158.
- Mascetti, S., Bernareggi, C., Belotti, M. (2011). TypeInBraille: A Braille-based typing application for touchscreen devices. In: *Proc. ASSETS 2011*, 2011;pp.295–296.
- Meng, Y., Wong, D., and Kwok, L. (2014). Design of touch dynamics based user authentication with an adaptive mechanism on mobile phones. In Proceedings of the 29<sup>th</sup> 48Annual ACM Symposium on Applied Computing (SAC '14). ACM, New York, NY, USA,1680-1687. DOI: <http://dx.doi.org/10.1145/2554850.25>
- Ma, Y., Feng, J.H., Kumin, L., Lazar, J., and Sreeramareddy, L. (2012). Investigating authentication methods used by individuals with down syndrome. Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility, ACM, 241–242.
- Nitesh, S, Watt, J. (2009). Authentication technologies for the blind or visually impaired, Proceedings of the 4th USENIX conference on Hot topics in security, p.7-7, August 11, Montreal, Canada.
- Nicolau, H., Guerreiro, T., Jorge, J. & Gon, D. (2010). Proficient blind users and mobile text-entry. In Proc. of the 28th Annual European Conference on Cognitive Ergonomics, ECCE '10, New York, NY, USA,. ACM,19–22.

- Natã, M. Barbosa, Hayes, J & Wang, Y. (2016). UniPass: design and evaluation of a smart device-based password manager for visually impaired users. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '16)*. ACM, New York, NY, USA, 49-60. DOI: <http://dx.doi.org/10.1145/2971648.2971722>
- Oorschot, P. C., Thorpe, J. (2008). On predictive models and user-drawn graphical passwords. *ACM Trans. Inf. Syst. Secur.*, 10(4):5:1–5, 33.
- Oliveira, J., Guerreiro, T., Nicolau., H, Jorge, J., and Gonçalves, D. (2011). BrailleType: Unleashing Braille over touch screen mobile phones. *INTERACT '11*. Heidelberg: Springer, 100-107.
- Orf., D. (2015). Tactus Technology. Tactile Keyboards That Rise Out of Touchscreens Are Finally Here. Retrieved from <https://gizmodo.com/here-is-facebooks-dubious-plan-to-prevent-its-2016-elec-1825072042>
- Paisios, N. (2012). Mobile accessibility tools for the visually impaired. *PHD Thesis*. 2012, Available from: URL: <http://cs.nyu.edu/web/Research/Theses/nektariosp.pdf> (retrieved online September 19, 2012).
- Pierce, B. (1995). Braille--what is it? What does it mean to the blind? *The World Under My Fingers: Personal Reflections on Braille*. [serial online] 1995 [cited 2016 Mar 14]; 15(1), Available from: URL: Retrieved from <https://nfb.org/images/nfb/publications/fr/>
- Poh, N., Blanco-Gonzalo, R., Wong, R., and Sanchez-Reillo, R. (2016). Blind subjects faces database, in *IET Biometrics*, vol. 5, no. 1, pp. 20-27, 3 2016. doi: 10.1049/iet-bmt.2015.0016
- Ruamviboonsuk, V., Azenkot, S., Ladner, R. (2012). Tapulator: a non-visual calculator using natural prefix-free codes. In *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility (ASSETS'12)*. ACM, New York, NY, USA, 221-22
- Refreshable Braille Displays. (2018). American Foundation for the Blind. Retrieved from <http://www.afb.org/prodBrowseCatResults.aspx?CatID=43>
- Said, K., Kuber, R., Murphy, E. (2015). AudioAuth: Exploring the Design and Usability of a Sound-Based Authentication System. *Int. J. Mob. Hum. Comput. Interact.* 7, 4 (October 2015), 16-34. DOI=10.4018/IJMHCI.2015100102 <http://dx.doi.org/10.4018/IJMHCI.2015100102>
- Siqueira *et al.* (2016). BrailleÉcran: A Braille Approach to Text Entry on Smartphones," *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Atlanta, GA, 2016, pp. 608-609. doi: 10.1109/COMPSAC.2016.5

- Sepic, B., Ghanem, A., and Vogel, S. (2015). BrailleEasy: One-handed Braille Keyboard for Smartphones. *Qatar Computing Research Institute, Qatar Foundation*
- Shabnam, M., Govindarajan, S. (2014). Braille-Coded Gesture Patterns for Touch-Screens: A Character Input Method for Differently Enabled Persons using Mobile Devices. *International Conference on Communication, Computing and Information Technology (ICCCMIT-2014)*.
- Southern, C., Clawson, J., Frey, B., Abowd, G., & 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, September 21-24, 2012, San Francisco, California, USA [doi>10.1145/2371574.2371623]
- Soukoreff, R.W., & MacKenzie, I.S. (2001). Measuring errors in text entry tasks: An application of the Levenshtein string distance statistic. Extended Abstracts, *Proc. ACM Conference on Human Factors in Computing System – CHI 2001*, New York, NY, 319-320.
- Sherman, M., Clark, G., Yang, Y., Sugrim, S., Modig, A., Lindqvist, J., Oulasvirta, A. & Roos, R. (2014). User-generated free-form gestures for authentication: security and memorability. Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, ACM, 176–189.
- Saulynas, S. & Kuber, R. Towards BCI and Gestural-Based Authentication for Individuals who are Blind. In Proceedings of the 19th International ACM Conference on Computers and Accessibility –ASSETS’17, Baltimore, MD, 403-404.
- Sauer, G., Holman, J., Lazar, H. Hochheiser & J. Feng. (2010). Accessible privacy and security: A universally usable human-interaction proof. *Universal Access in the Information Society*, 9 (3), 239- 248.
- Kane, S., Wobbrock, J., Ladner, R. (2011). Usable gestures for blind people: understanding preference and performance, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, May 07-12, 2011, Vancouver, BC, Canada, doi:10.1145/1978942.1979001
- Subash, N., Nambiar, S., and Kumar, V. (2012). Braillekey: An alternative Braille text input system: Comparative study of an innovative simplified text input system for the visually impaired, in *Intelligent Human Computer Interaction (IHCI), 2012 4th International Conference on*, Dec 2012, pp. 1–4.
- Schneier., B. (2009). The secret question is: why do IT systems use insecure passwords? The Guardian, <http://www.guardian.co.uk/technology/2009/feb/19/insecure-passwords-conflickerb-worm>.

- Soft Braille. (2018). Keyboard. Retrieved from <https://play.google.com/store/apps/details?id=com.dalton.Braillekeyboard&hl=en>
- Sangore, R. B, Patil, G., Ramani, S., Pasare, S. (2014). Authentication Using Images and Pattern. International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering. Retrieved from <https://www.ijareeie.com/upload/2014/april/27XAuthentication.pdf>
- Sauer, G., Holman, J., Lazar, J., Hochheiser, H., & Feng, J. Accessible privacy and security: A universally usable human-interaction proof. Universal Access in the Information Society, 9(3), 2010, 239- 248.
- Suo, X. Zhu, Y. Scott, G. O. (2005). Graphical Passwords: A Survey, Proceedings of the 21st Annual Computer Security Applications Conference, p.463-472, December 05-09, 2005 [doi>10.1109/CSAC.2005.27]
- Sae-Bae, N. Memon, N. and Isbister, K. (2012). Investigating multi-touch gestures as a novel biometric modality. Proc. of IEEE Fifth Intl' Conf. on Biometrics: Theory, Applications and Systems (BTAS), 14:156–161.
- Tobe, C. B., Callahan, E., and Callahan, M. (2000). Embossed Printing in the United States, in *Braille: Into the Next Millennium*, J. M. Dixon, Editor. National Library Service for the Blind and Physically Handicapped of the Library of Congress: Washington, D.C., 41-71.
- Von Zezschwitz, E., De Luca, A., & Hußmann, H. (2014). Honey, I shrunk the keys: Influences of mobile devices on password composition and authentication performance. Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational, ACM, 461–470.
- WritingBraille. (1999). MarvelSoft Enterprises, Inc., DBA Future Aids, The Braille Superstore. Supplying products to blind people and their families since 1999. Retrieved from <http://www.Braillebookstore.com/Writing-Braille>
- What is Braille. (2008). The Tennessee Council of the Blind (TCB). Retrieved from <http://www.acb.org/tennessee/Braille.html>
- Wolf, F., Kuber, R., Aviv, a. Perceptions of Mobile Device Authentication Mechanisms by Individuals who are Blind. ASSETS'17, Oct. 29–Nov. 1, 2017, Baltimore, MD, USA
- Yamada, H. (1980). A Historical Study of Typewriters and Typing Methods: from the Position of Planning Japanese Parallels. Journal of Information Processing, 2, 175-202.