mBrailler: Multimodal Braille Keyboard for Android

Randolph Duke II, Hugo Nicolau, Vicki L. Hanson
B. Thomas Golisano College of Computing & Information Sciences
Rochester Institute of Technology
Rochester, NY, USA
rd7847@rit.edu

Abstract— Touchscreen devices have paved their way into the mobile scene, presenting a wide set of possibilities but a comparable number of new challenges, particularly for people who are blind. While these devices have a small number of tactile cues, such as buttons, they provide the opportunity to create novel interaction techniques. In this paper, we present mBrailler. mBrailler is a mobile braille keyboard that combines the benefits of physical keyboards (speed and accuracy) and gestural interfaces (flexibility and personalization). We built an 8-button Braille keyboard that can be attached to the back of mainstream smartphones allowing fast and familiar chorded input. On the other hand, the touchscreen enables thumb entered gestures for more complex text editing operations, such as caret movement, text selection, copy, and paste. This project combines the tactile benefits of Braille typewriters with the customization of smartphone applications. We aim to provide a more efficient and effective typing experience for blind users, thus increasing their productivity with current mobile devices.

Keywords— Accessibility; Smartphone Typing; Braille Multimodal Interfaces; Text Entry.

I. INTRODUCTION

With the ever increasing amount of smartphones that are sold and used on a daily basis, there also continues the need for growing accessibility features. Not all smartphone users share the same physical capabilities required to use a smartphone right out of the box. Blind and low vision people who use smartphones have not found all built in accessibility features to suit their every need. Moreover, screen readers, speech entry, and text-to-speech (TTS) software are not always ideal in crowded areas. Sending personal and sensitive information with a smartphone makes blind and low vision users susceptible to “shoulder surfing” by others in public spaces [2].

People who are blind or low vision use built in accessibility software that may be overly complex to use with third party software and apps. Although smartphone manufacturers include optional accessibility features to enhance the user experience, there remain several shortcomings that can make typing a challenge. Some of these challenges include taking additional time to complete trivial tasks, erratic behavior by the software, or lack of controls to customize the personal user experience [1].

mBrailler is looking to address the following concerns:

- What would a fully portable Braille keyboard act like if it was attached to the phone?
- How can blind and low vision users get to the advanced text editing features normally hidden deep within menus very quickly?
- Can typing and editing be made easier without using just the touchscreen?

II. CURRENT CHALLENGES

A. Exploration

Touchscreen typing can be challenging due to the lack of physical keys. Tactile QWERTY keyboards and Perkins Braille keyboards allow for fingers to find the home row without committing any characters to the document being typed. Smartphone touchscreen keyboards conversely rely on screen readers and vibrations for screen exploration. Scanning to find the right keys to type can become overwhelming with the amount of feedback smartphones respond with as audio-vibrotactile feedback. The lack of non-visual cues may have a negative impact in the amount of words per minute (WPM) [8].

The Android operating system uses a functionality known as Explore by Touch that lets people use their thumbs to...
receive feedback about items on the screen. Explore by Touch lets users swipe their fingers either following the x- or y- axis of their screen. For every swipe there is a vibration and sounds that read text or labelled images. To help give users an understanding of how far they have navigated there is a pitch sound that increases or decreases intensity based on the location of the Explore by Touch indicator. After scrolling to find what they are looking for, the user can double tap anywhere on the screen to activate the button. This can be a challenge as single tapping on the screen makes Explore by Touch focus on any on screen item in that location. Any stray taps can change where the Explore by Touch selector is on the screen.

The difficulty of this exploration lies with the on-screen keyboard. The keyboard does not follow the same navigation style as the rest of Explore by Touch and forces people to find a key and drag along the keyboard and release to type that character. Explore by Touch makes users have to find the keyboard and any fingers that leave the letters triggers the normal Explore by Touch drag interaction. These inconsistences are very difficult to navigate and perform simple tasks such as typing, let alone the advanced settings.

B. Portability

![mBrailler Braille keyboard on the back of an Android smartphone.](image)

Users who wish to type using hardware Braille keyboards have several options. The costly keyboards must all be carried in addition to the smartphone blind users travel with [4]. As of now, there are no physical Braille keyboards that connect onto commercially available smartphones. Using the back of the phone for typing could potentially make traveling with the Braille keyboard more convenient and private [5]. Although there are several applications that use the screen for Braille chord entry, there may be more of an opportunity to use the screen for more advanced operations. Utilizing the back of the phone for typing could allow for the software to augment the smartphone experience [6].

C. Software

The software on smartphones that exists to aid blind and low vision users may sometimes get in the way of completing some tasks. The Android smartphone operating system contains software to help read the screen. Along with the screen reader is an exploration software to assist in providing feedback for buttons that users can interact with on the touchscreen. The difficulty in using the accessibility features together is when screen touches and gestures correlate to both the screen reader (TalkBack) and the navigation (Explore by Touch). The inconsistency of finger touch functionality on screen when a software keyboard is present is a very challenging experience that can deter users from wanting to continue trying to use it for daily purposes [4].

D. Personalization

Customizing how Explore by Touch operates (reading speed, text selection, text editing, etc.) is found in a TalkBack rotary menu that is triggered by making an L-shaped thumb gesture. The menus that appear are contextual, meaning if a keyboard is open and the TalkBack settings are opened, there are menu options that will appear to achieve text editing functions. These functions include the text selection tools, the granularity in the screen reader (letter by letter to full words), and access to the text clipboard functions cut, copy, and paste. To select an item a user must slide their finger along the certain part of the menu and release their finger to select that option. Although the gesture is referred to in the initial Explore by Touch tutorial, many people are not aware of it or the text editing functions [3]. Having access to toggle when and how text is read can make using smartphones a more functional and enjoyable experience.

III. TECHNICAL SPECIFICATIONS

mBrailler uses the front and the back of the phone for text input and edit. The back of the phone has a Perkins-like Braille keyboard for typing characters and symbols. The front of the phone uses a custom keyboard app that connects to the physical Braille keyboard. Touchscreen thumb gestures are used for text editing through taps and swipes.

A. Back Keyboard

In order to type without using the touchscreen, current solutions have resorted to using hardware keyboards. For Braille, there are Bluetooth enabled typewriters and notetakers with refreshable displays. For mBrailler there is a custom Bluetooth Perkins-like typewriter fashioned to the back of the phone.

The construction of the keyboard prototype uses an Arduino Uno microcontroller and three small protoboards for the soldered buttons, wires, and resistor electronics. The keys on the back of the phone follow the six-dot Braille alphabet and communicate with the phone over Bluetooth. There are eight keys in total, with two additional keys for the “space” and “delete” functions. The Braille keyboard itself follows the American English Braille library, which includes chords to toggle text capitalization and the number keyboard.

B. Coding

The code is built on the open source Android project, “Open Braille” [7]. The communication between the buttons and the Android smartphone is handled using an open source connection library known as Amaro. Amaro is used to facilitate the communication of button presses over Bluetooth for interpretation by Android. Communication is established and handled by the open source library imported on the Arduino microcontroller and the Android service installed on the phone. The BlueSMiRF Bluetooth modem is not
configured to use the human-interface device (HID) protocol to use Amarino for communication.

The Android service acts as a system wide keyboard, allowing for usage in any app that requires typing. Once the keyboard has been added for usage on the Android phone, it may be set as the default keyboard. When the keyboard is asked for in an app, this triggers the phone to connect to the Arduino. Upon a successful connection, the LED on the BlueSMiRF goes from blinking red to steady green. On the phone screen, instead of the default keyboard coming on screen, a screen overlay is present that still allows for Explore by Touch to work.

The button presses are interpreted as byte code on Android. That code is referenced to a hash map consisting of American English Braille letters, symbols, and numbers. American English Braille is used to map each chord structure to a letter. There are two toggles in place to adjust the printed characters: a shift function and a number function. The shift function toggles off after the next character is typed. The number function must be explicitly toggled off by entering the number toggle function again. When the keyboard is no longer needed, the Android system back button on screen is used to end its usage. The connection between the phone and the microcontroller stays on until either explicitly disconnected by a user in the Amarino app or the connection times out after a couple of minutes automatically.

The gesture operations require the usage of several different libraries included with the Android operating system including MotionEvents and GestureRecognizers. The touchscreen processes finger presses and allows for different classes to be called and recognized from there based on the “pointers” that were used. Because the location of the pointers change the functional properties of the gesture recognizers, the X and Y coordinates of each finger press are recorded along with the any changes to those values. Any change in the X and Y coordinates are recorded until that finger is lifted. The pointer up functions are then compared to a table of possible text editing functions that are executed by the finger movements.

Fig. 4. mBrailler gesture using both thumbs.

The gestures for the project use the left and right side of the screen as toggles for entering swipes on the opposite side of the screen. The gestures used for the editing functions simply follow up, down, left, and right motions of the thumb. When the left or right thumb press the screen by themselves, they move the text caret either left or right. Right thumb swipes control the cursor granularity and moving to the beginning or end of text. Left thumb swipes select text and scroll through the clipboard editing commands: cut, copy, and paste. The up and down scrolling reads off the command and actually performs it when the thumb has been lifted.

IV. OBSERVATIONS

Setting up mBrailler is relatively simple. Because this system uses the Amarino library for communication between the physical Braille keys and the phone, there is an additional app that must be downloaded onto the Android smartphone. The Bluetooth communication had to have a specific transmission baud rate set by the Arduino microcontroller to 115200 baud for the connection to work. The Braille keyboard pairs as any other bluetooth device and continues to re-establish the connection every time the keyboard is needed to type in an app.

Because mBrailler replaces the phone keyboard it was tested for usage in several applications and succeeded. Anywhere a keyboard is evoked by the phone the mBrailler keyboard was able to function normally. However, due to the technical limitations of Android, to use the gesture keyboard required turning off the Explore by Touch screen reader. The Explore by Touch feature had to be turned off in the system settings since it would take any screen touches as an attempt to read the screen. Fortunately, even when the screen reader Explore by Touch was turned off the TTS function, TalkBack, still works. Having TalkBack enables users receive feedback about what they are typing.

The Braille keys on the back of the phone instantly respond to presses. Upon pressing the appropriate Braille chord the phone places the character in the appropriate text box and reads it aloud using TalkBack. We did observe that when typing too quickly, TalkBack can sometimes respond on a delay making it difficult to be sure that the proper character just read was what had just been typed. Having the Braille buttons in landscape on the back of the phone is very comfortable. The amount of space this affords the thumbs to not overlap while entering gestures made entering them easier.

Attempting to use Explore by Touch and TalkBack to type was originally very complex and stressful. What made things more difficult to comprehend was how to use them both concurrently to manipulate what was already typed. mBrailler performed very admirably in replacing parts of the rotary controlled TalkBack menu with regards to text navigation. Instead of the “L” shaped gesture required to bring up the text caret navigation controls, the ability to use two thumbs was more directly mapped to a function.
V. Future Work

The proof of concept keyboard created may be of some use to the blind community for quicker, private Braille typing. By using the back of the phone on a case, future prototypes should explore the portability of the device with improved electronics and batteries.

A. Areas to Improve

a) Software: The software is also for prototype usage only at the moment. The libraries that handle the communication between the hardware and Android require a companion app for connection. Performance of a native application would also need to be considered for real world communication (e.g. impact on phone battery life, additional languages, etc.).

b) Gestures: The viability of the gestures created by this project for blind and low vision users should also be taken into further consideration in future work. Because the new gestures require additional memorization to find the appropriate text editing function vs. the rotary control, cognitive load must be considered for ensuring the gestures’ accuracy.

c) Human factors: Button placement and its impact on users’ comfort needs to be evaluated. Having feedback by real blind typists who may be familiar with other Braille keyboards and notetakers would be of great value. The design input by blind typists who may be familiar with other Braille keyboards and notetakers could present better insight than situationally blind users and researchers [2]. Their input would be helpful in guiding hardware decisions such as button placement, key functions, and the viability of multimodal typing on a regular basis. Since there are no commercially available keyboards that follow this paradigm, rigorous testing should benefit further research and efforts in multimodal input systems.

d) Field testing: Due to the varying nature of Android smartphone sizes, for further lab testing and possible field prototypes, a removable keyboard or adaptable sized keyboard design would be worth considering. The mBrailler prototype represents the beginning stages for a more advanced multimodal input system. Because the project will be open sourced and shared with the research community, others will be able to adapt and modify the code and hardware configurations.

VI. Conclusion

To address the goals of this project, the mBrailler project demonstrates how a Braille keyboard could function if attached to a phone. The ability to not rely on an “L-shaped” gesture to access the same content seems to make access to features that were unknown by those using only Explore by Touch and TalkBack. A simple press and toggle that does not interfere with anything but the task at hand seems to be a more manageable task when looking to type. By offloading the need to peck and find keys to new using a back of phone keyboard makes the role of the screen to be simply for navigation and controls.

This project stands with positive intent on making the typing experience more palatable for blind and low-vision smartphone users. That said, there are multiple opportunities with Google’s system software that would make projects similar to this one easier to reproduce. Advancement and access to the Explore By Touch functionality would be ideal for developers who are explicitly working to enhance the user experience of screen navigation. Due to the lack of system tools required to make a functional alternative screen reader toggle off temporarily for screen input, apps and services like mBrailler can remain purely proof of concept until embraced by smartphone OS distributors.

VII. Acknowledgments

I would like to thank Vicki Hanson and Hugo Nicolau for their insight and advisement through this project. The code is a continuation of the open source library Amarino and the OpenBraille project.

REFERENCES


