INEXPENSIVE BRAILLER: Writing Braille Using IMPAD

by
Merve Keles

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science
Department of Computer Science Courant Institute of Mathematical Sciences New York
University May 2009

Professor Ken Perlin
Acknowledgements

I would like to thank my advisor, Professor Ken Perlin, for his motivation and guidance. I am also thankful for providing useful ideas and sharing his knowledge. I would also like to thank Nektarios Paisios, who inspired me for this research topic, for bringing new ideas and helping to access many resources related to visually impaired environments. I would also like to thank Professor Davi Geiger for being my second thesis reader and for his support.

I am very grateful to the entire UnMousePad team who made this research possible by designing the UnMousePad. I am grateful to Alexander Grau, for explaining the implementation of UnMousePad and Charles Hendee for solving any problems I had during the implementation. I am also very thankful to Ilya Rosenberg, for providing me the UnMousePad any time I need. I would also like to thank Nadim Awad, for his patience in answering my questions.

Finally, I would like to thank my family, for their support. I would like to thank my father for his endless support throughout my education and for making it possible to receive a Masters degree in Computer Science at New York University.
Abstract

Blind people use a special system, called Braille, to read or write. There are special devices or software that enables the blind users to write Braille. Most commonly used Braille writers, such as Perkins Brailler, are mechanical. Due to their mechanism they are heavy to carry, noisy and still expensive. Some blind users choose to use electronic devices, such as electronic Braille note takers, to be able to take notes in a less noisy way. However, these electronic versions are uncommon due to their high costs. Also, both mechanical and electronic Braille writers can be easily damaged by the age group of school kids.

To address these problems, we are proposing an inexpensive Braille writer, which will be used with a multi touch device connected to a computer. During the preparation of this paper, we worked together with a visually impaired PhD student. We adopt his idea of bringing Inexpensive Multi-touch Pressure Acquisition Device (IMPAD) to the Braille. Since he is one of the users who come across with the difficulties of using Braille writers, he helped us address these problems. He also helped us improve our techniques by testing the Brailler weekly and giving feedback about it.

First of the three methods to enter Braille input turned out to be a good Brailler, such that, a blind person can write as fast with this Brailler as he/she writes with any other Braille writer. On the other hand, the second method was not that usable as any other device. In theory, the third one can be the best way for a blind user to interact with a multi touch device; however it has not been tested by a blind user.
CONTENTS

Acknowledgements...........................................................................................................iii

Abstract..............................................................................................................................iv

List of Figures....................................................................................................................vii

1 Introduction......................................................................................................................1

2 Braille..............................................................................................................................3

2.1 Overview .....................................................................................................................3

2.2 Braille Encoding..........................................................................................................3

2.3 Braille Writers.............................................................................................................5

3 IMPAD - Inexpensive Multi-Touch Pressure Acquisition Device.................................7

3.1 Overview.....................................................................................................................7

3.2 Technology..................................................................................................................8

4 IMPAD for Braille Writing............................................................................................10

4.1 Input Methods..........................................................................................................10

4.2 Speech Output..........................................................................................................16
5 Implementation........................................................................................................17

5.1 By Pressing Down.................................................................................................17

5.2 By Lifting..............................................................................................................22

5.3 Kobigraphs..........................................................................................................23

6 Results....................................................................................................................28

7 Conclusions and Future Work..............................................................................30

Bibliography.............................................................................................................32
List of Figures

Figure 2.1.1: Braille Cell .................................................................3

Figure 2.2.1: Braille Letters ............................................................4

Figure 2.2.2: Braille Numbers ........................................................4

Figure 2.3.1: Perkins Brailler ............................................................5

Figure 3.1.1: IMPAD ................................................................8

Figure 4.1.1: Brailler with IMPAD ..................................................11

Figure 4.1.2: IMPAD Brailler Display with Six Touching Fingers ..........13

Figure 4.1.3: Displaying Input for Space .........................................14

Figure 4.1.4: Letter M ................................................................15

Figure 4.1.5: Entering M by Pressing Down .....................................15

Figure 4.1.6: Entering M by Lifting the Fingers ..................................15

Figure 4.1.7: Words and Abbreviations .........................................16

Figure 5.1.1: Zones for Touches .....................................................19
Figure 5.3.1: Kobigraphs.................................................................................................................23

Figure 5.3.2: Directions for Drawing Kobigraphs.............................................................................24

Figure 5.3.3: Two Different Ways to Represent N..............................................................................25

Figure 5.3.4: Characters with Similar Kobigraph Representations..................................................26
CHAPTER 1

Introduction

Use of Braille has been declined because computers are easier for blind users to use, since they come with the screen readers [1]. With computers, they can connect to web and send emails and get more connected with their friends. This results in losing interest in Braille, and in losing skills to read or write Braille.

The high cost of common Braille writers limits the usage of the devices hence the learning necessity to write Braille. At schools, students are still being taught how to read or write Braille using these devices, but they have to carry their heavy Braille writers in order to be able to learn. The use of Braille writers in the class is also problematic, because they are noisy. This brings out the need to make Braille more attractive for many users so that they can still train themselves reading or writing Braille.

There has been a highly interest in multi-touch devices, by sighted users. However, due to interaction techniques that require the user to visually locate objects on the screen, these devices are still inaccessible by blind people [6].
In response to these problems, we implemented a Brailler, which is a Java-based software used with IMPAD, which is a light, flexible multi touch device. It is paper thin, hence, easy to carry. It is also flexible, and can be found in various sizes. So a blind person can even carry the device in his pocket, and connect it to his/her laptop computer. She/he will be able to enter and edit text, and take notes in any environment without causing a lot of noise. With this design, we are also making the Braille writing more fun with a multi-touch device, which they are not always able to use.

Our Brailler will receive the user’s input which is basically the Braille cells. So, the input type is the same as the mechanical or electronic Braille writers. We describe different methods for the way the user enters the Braille input. It will play the sound of any letter entered by the user and the sound of a word if it is recognized by the system.

Throughout this paper, we will first describe how blind people write Braille using Braille writers. We will also give a brief explanation of IMPAD’s technology. Then we will focus on how to write Braille using IMPAD. We will propose different ways of entering Braille input into IMPAD.
CHAPTER 2

Braille

2.1 Overview

Braille, created by Louis Braille in 1821, is being used by the blind people to read or write.

Braille is a system consisting of Braille cells (or Braille characters). These cells are composed of six dots, two columns across and three rows down. Dots in the six-dot cell numbered 1, 2, and 3 from the top of the left column, and 4, 5, and 6 from the top of the right column.

<table>
<thead>
<tr>
<th>1</th>
<th>●</th>
<th>●</th>
<th>●</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 2.1.1: Braille cell.

2.2 Braille Encoding

Letters are represented by cells in which certain dots are embossed and others left blank. For example letter ‘a’ is represented by a Braille cell in which only one dot (the top of the left
column) is embossed. Using the top four dots of the Braille cell, one can represent letters a through j. To represent letters k through t, dot 3 is added to each of the Braille characters, a through j. Finally to represent letters u, v, x, y and z, both of the dots 3 and 6 are added to the Braille characters, a through e. The letter w is an exception to the pattern, and represented with dots 2, 4, 5 and 6.

![Braille Letters](image1)

**Figure 2.2.1: Braille Letters.** The Braille cells are shown with the letters they represent.

Numbers 1 through 0 is represented in the same way as letters a through j, with a Braille character added to the front (dots 3, 4, 5, and 6) as a prefix.

![Braille Numbers](image2)

**Figure 2.2.2: Braille Numbers.** The Braille cells that represent the numbers 1-0.
Braille Alphabets can belong to any of the three groups known as grades. Grade 0 has simple letter by letter translation. Grade 1 has all the letters plus the punctuations marks and numerals, and also very limited contractions. Grade 2 is the most commonly used one and it has a rich set of contractions. Some of the contractions are common words such as “could”, “and”, “for” and “the”, common word prefixes and suffixes such as “ing”, “ation” and “part”, double letters such as “cc”.

2.3 Braille Writers

Braille writers are the equivalent of the typewriters. The difference is Braille writer has a few as seven keys, one for each dot of a six-dot Braille cell plus a space key. On the Braille writer, standard arrangement of the six keys corresponds to dot positions 3-2-1-4-5-6 from left to right. So dot 3 is pressed by the left ring finger, dot 2 by the left middle finger and dot 1 is pressed by the left index finger. Dots 4, 5 and 6 are pressed by the right index, middle and ring fingers respectively.

![Perkins Brailler](image)

**Figure 2.3.1: Perkins Brailler.** A commonly used mechanical Braille writer, Perkins Brailler, has six keys for the dots of a Braille cell, one key for space, new line and backspace keys.
A Braille cell is produced by pressing down any combination of the six keys that correspond to the desired Braille cell’s filled dots. This process of simultaneously pressing down the keys is usually called chording.

Braille writers can be manual and electronic. In a manual Braille writer, dots are embossed on the paper mechanically as a direct result of the user’s pressure on the keys. On the other hand, in an electronic Braille writer, the keys require only light pressure to send an electrical signal that causes the machine to emboss a dot.

In addition to special Braille writer devices, a blind user can use the regular QWERTY keyboard to type with six-dots. If the keyboard is capable of accepting simultaneous signals from two up to six keys, then the user can be chording using s, d, f and j, k, l keys on the keyboard. This requires software such as Duxbury’s Perky Duck or Pokadot to combine the input from the keys into the correct numerical code corresponding to the Braille cell [1].

Mechanical brailleers are heavy machines and can be difficult for children and tiring for anyone. Due to their mechanism they are too noisy to be used in certain circumstances. Electronic brailleers are generally faster and easier to use compared to mechanical ones. Both versions are reasonably expensive and they can be easily damaged by the age group of school children.
CHAPTER 3

IMPAD - Inexpensive Multi-Touch Pressure Acquisition Device

3.1 Overview

Recently there has been a great interest in multi-touch interfaces in various forms of systems such as optical systems (Microsoft Surface) or capacitive systems (Apple iPhone). At NYU Media Research Lab, an inexpensive, paper thin and flexible multi-touch device, called Inexpensive Multi-Touch Pressure Acquisition Device, IMPAD, has been designed. By being flexible, we mean that it can be designed in any size and also can be wrapped around curved surfaces. This device can sense different levels of pressure which adds a third dimension of input to touches.

IMPAD can be used with existing computers and displays, and more importantly is as inexpensive as a keyboard or mouse. It is also called UnMousePad, because it is similar to a mouse-pad but doesn’t require a mouse to operate.
3.2 Technology

IMPAD consists of five layers: the first and fifth layers consist of parallel wires [5]. The wires on layer 1 are designed in parallel and they are perpendicular to the direction of the parallel wires on layer 5. The second and fourth layers consist of resistive material which allows a current between adjacent wires of layer 1 and layer 5. The third (middle) layer consists of a material whose electrical resistance decreases when compressed allowing for the sensing of outside forces [5].

When external pressure is applied to the sensor, a path is created that allows current to flow from layer one to layer two, and through the point of contact in layer three [5]. Then it flows along layer four and finally passes an electrode in layer five. The proximity of the contact point from an intersection and the pressure exerted at that point determines resistance of the path created when the pressure is applied [5]. Then the path’s resistance specifies the resulting voltage.
at that intersection point. Finally, the data received with this process is read by the micro-
controller and sent to a computer over a USB [5].
CHAPTER 4

IMPAD for Braille Writing

4.1 Input Methods

Blind people use Braille writers in the following way: Their left index, middle, ring and right index, middle, ring fingers are continuously placed on the device. The thumb is used to press space bar, and some Braille writers also have keys for new line and backspace characters which are pressed with the left and right little finger, respectively. But these fingers are not placed on the device continuously.

During the implementation of our Braille Writer, we have tried different input methods, considering layout and number of the keys and the way the input will be entered. First, we implemented a Brailler with nine keys, six for each of the Braille dot, one for space, one for backspace and one for new line. The positions of the keys were predefined, so the user would place his/her six fingers on the keys, that are for six Braille dots, as described above - left ring, middle, index and right index, middle, ring fingers, respectively - and would be applying more pressure on any combination of these keys in order to enter the input. To enter the new line, space and backspace characters, they would use the left little finger, the thumb, the right little finger, respectively. For the user to feel the exact position of the keys, we designed soft keys made out of simple fabric, and placed them on the IMPAD. The user would use these as the
buttons, and with this tactile feedback, they would not need to locate the keys after they lift their finger(s). This version was working correctly except the number of the keys was even more than some Braille writers. So we tried to come up with something that will have less number of keys so that the user will be able to select the six keys that correspond to the six Braille dots faster.

Figure 4.1.1: Brailler with IMPAD. A paper, with buttons on it, is placed on IMPAD, for the user to feel the exact points of their fingers on IMPAD.

Secondly, we tried a method without the predefined positioning of the keys. This time the user would place his/her six fingers in the order, left ring, left middle, left index and right index, right middle, right ring, anywhere on the IMPAD and the position of the nine keys would be registered once the user applies more pressure on these already touching fingers. The position for the nine keys would be such that, the keys for the six dots would be the points where the fingers are touching the IMPAD; the space key would be in the middle of index fingers; the backspace key would be on the left of the left ring finger; the newline key would be on the right of the right ring finger and to enter the new line, space and backspace characters, they would use the left little finger, the thumb, the right little finger, respectively. Basically, the layout was the same as...
seen in the picture above and we were using the soft buttons described above again, but this time, we removed the buttons for the backspace, new line and space keys. This way, the user will be able to locate the exact six fingers that they suppose to press, in order to start the device. We also made the buttons smaller and circler instead of elliptical. This help the user feel the area of their touch and this way they would not touch somewhere so far away from their first touch point on the IMPAD. In this method, the number of the soft buttons was six, less than the first method’s layout but there were still 9 keys used in order to enter input.

In the third version of the IMPAD Brailler, we use the same methodology as the second one, except the use of little fingers. Instead of having separate keys for new line and backspace, we use one of the Braille cells which are not mapped into any character. The six fingers will be on the IMPAD continuously, without applying pressure. If the user lifts his/her hands, they will be able to replace their fingers, which will result in new positions for the keys. This is convenient since they are able relocate the points where their fingers are touching the IMPAD.

With the six fingers placed on the IMPAD, they will apply pressure on all of these six fingers at the same time. The device will register the positions of the six dots, plus the position for the space key, which will be in the middle of the left and right index fingers.
Figure 4.1.2: IMPAD Brailler Display with Six Touching Fingers. The program displays the keys and the touches for sighted users. The red dots indicate that the fingers are touching the IMPAD exactly on that point.

The device will beep indicating that it is ready to receive input. Then the user can remove the pressure from the fingers but still will be touching the IMPAD in the same position. If they do not touch for a little while, meaning they lift their hands, the device will beep to alert the user that the repositioning of the six fingers is necessary.

The input will be received as the user applies more pressure on any combination of these touching fingers. To press space key, they will press down with their thumb anywhere between their left and right index fingers.
Figure 4.1.3: Displaying Input for Space. The space key is colored to black indicating that it is pressed down. The user is not applying pressure on the other fingers.

If they want to press new line key, they can do so by just pressing dot 3, which is already touched by their left ring finger and if they want to press backspace, they will press dot 2, which is underneath their left middle finger.

Addition to the third method, we also implemented a method that lets the user enter the input by lifting the fingers. In the methods that work by pressing down, the user will need to apply more pressure on any combination of the touching fingers. In this method, the user will lift any combination of the touching fingers. To enter the space character, user will click anywhere on the device, while pressing down with the six finger. So there will be no separate key for the space in this implementation.

For example, if the user wants to type the character ‘a’, using ‘by pressing ’method, he/she will apply more pressure on the left index finger, which corresponds to the Braille cell with only dot 1 is embossed. Using ‘by lifting’ method, the user will lift the left index finger while the other five fingers touching the IMPAD. If he/she wants to type the character ‘m’, which is the Braille cell with dots 1, 3, and 4 is embossed, using ‘by pressing’ method, he/she
will apply more pressure on the left ring finger, left index finger and right index finger. Using ‘by lifting’ method, the user will lift the left ring, left index and right index fingers, while other three fingers are on the IMPAD.

**Figure 4.1.4: Letter M.** The Braille symbol for the letter M

![Figure 4.1.4: Letter M](image)

**Figure 4.1.5: Entering M by Pressing Down.**

**Figure 4.1.6: Entering M by Lifting Fingers.**

As the user enters the input, there will be a mapping to a letter, number or some symbols.

In this Braille writer, we use Grade 2 Braille which is used more commonly by the blind users in most cases. The user will be able to enter a set of abbreviations of the Grade 2 Braille. To switch from letters to words, we used the Braille cell with dot 4 is embossed. So the user can switch between letter and words by pressing down or lifting her/his right index finger.
4.2 Speech Output

Our Braille writer includes a sound file for every letter of the English alphabet and numbers through 1 to 0. When the user enters a letter in the way described above, the corresponding sound for that letter will be played.

For example, when the user enters the letters, ‘T’, ‘H’, ‘I’, ‘S’, and then presses the space key, the word “this” will be recognized and the corresponding sound will be played. After each word recognized by the Braille writer, the user will be able to hear the word they just typed.
CHAPTER 5

IMPLEMENTATION

5.1 By Pressing Down

The program starts with the checking the connection of the USB Port that connects the UnMousePad (IMPAD) to the computer. This is done with the execution of the line:

UnMousePad ump=new UnMousePad(this);

If it is successfully connected, UnMousePadCallback() function will start to listen for touches and handles them. The functions; UnMousePadDown() and UnMousePadUp(), act as a regular mouseDown and mouseUp or keyDown and keyUp functions. Instead of having x and y as parameters, UnMousePadDown() and UnMousePadUp() functions has an integer ‘touch’ as a parameter.

With the creation of the connection to the UnMousePad, we can reach the array of touches by simply saying, ump.touches(). This array will give us the number of the touches, which is: ump.touches().length.
We can also retrieve the pressure, x and y positions of the touches in this array, which are:

\[
\text{ump.touches()[touch].w, ump.touches()[touch].x, ump.touches()[touch].y}
\]

Integer ‘touch’ representing the index of the array, is the same as the touch parameter of \text{UnMousePadDown()} and \text{UnMousePadUp()} functions. It allows access to that particular touch in the array of touches.

With this information we are able to show (x, y) coordinates of where the fingers are interacting with the UnMousePad. When the user places six fingers (left ring, middle, index and right index, middle, ring fingers respectively) on the UnMousePad, we draw little red circles with the center point (x, y) that we retrieved as described above. This is good for the sighted users to see where they are actually touching. When the user applies a little pressure on six fingers at the same time, the user will hear a beep, indicating that they registered the position of the six keys that corresponds to the six Braille dots and also the position of the space key on which they are not currently touching:

\[
\text{space.x = ( thirdtouch.x + fourthtouch.x ) / 2 }, \text{ space.y=thirtdtouch.y;}
\]

We draw rectangles that bound the touches, and also the rectangle that will act as the space key. We will refer to these rectangles as zones. There are seven zones numbered from 0 to 6.
Figure 5.1.1: Zones for Touches. Zones are numbered from 0 to 6, in the order seen in the figure.

From this point on, as the user does the chording, meaning as she/he presses multiple key combinations, they will create touches. These touches will be represented as points which will be checked whether they are within one of the seven zones. If their point of touch is within one of these zones and they are applying enough pressure, they are actually pressing down the key that corresponds to that zone. If they are applying little more pressure on their left index finger, which is within the third key (represented as zone 2), they will be pressing down the dot 1 of the six Braille dots. This is done by, keeping track of every touch: Within a loop, going through the touches in the `ump.touches()` array one by one, we get the parameters: pressure of this particular touch; p(x, y), point where this touch is interacting the UnMousePad; the index of the touch in the array. `computeTouches()` function takes these parameters and checks whether this particular touch is within a zone, and also checks whether the pressure applied to create this touch is enough to accept it as a ‘press down’ action. After the check returns true, we send the zone to an array of ‘pressed keys’. When all the touches are sent to the `computeTouches()` function, and the pressure on the fingers are released, the `UnMousePadCallback()` function will check if the
number of touches is 0, meaning the fingers are on the UnMousePad but the pressure on them are not enough to create a touch. If the number of touches is zero, and array of pressed keys is not empty, the elements of the pressed key array will be checked. When the user touches with enough pressure on the UnMousePad, the `UnMousePadCallback()` will run many times resulting in an array of pressed keys with duplicated elements. So, we check this and if an element appears in the array more than once, we remove the duplicates. Since, we have seven zones, six for Braille dots and one for space; we should have an array of pressed keys with length less than seven. With no duplicate zones in the pressed key array, we send it to `selectKey()` function which will map the zones in the pressed key array, into their corresponding binary representation. The way we map the zones to their binary representation is as follows: zone ‘a’ will have a binary representation as $2^a$ for $a=0, 1, 2$; however, there is no need to map the zone that is for the space key, so for $b=4, 5, 6$, the binary representation is $2^{b-1}$.

Zone 0: $2^0 = 1$,

Zone 1: $2^1 = 10$,

Zone 2: $2^2 = 100$,

Zone 4: $2^3 = 1000$,

Zone 5: $2^4 = 10000$,

Zone 6: $2^5 = 100000$,

As the chording done by applying pressure on any combination of the fingers, these binary numbers will be received as a result of the process so far. Then we add these binary
numbers, and the sum will give us the key to the look up table which will return the corresponding Braille cell.

For example, if user applies more pressure on the left ring finger and left index finger, the pressed key array will contain the zones 0 and 2, which corresponds to 1 and 100 in the binary representation. So the sum of the binary representation of the zones that contain a touch will be 101. This corresponds to the letter ‘K’ in the look up table.

If the user wants to press down one key at a time, then we do not need to map the zone number of that key into binary. For example, if the user wants to enter the letter ‘a’, he will simply press down on his left index finger, which corresponds to zone 2 (the first dot of the Braille cell). selectKey() function will only send the number 2 to the look up table.

The same process applies for entering backspace or new line characters. The user will press down with their left middle finger for backspace and with their left ring finger for new line. For the backspace character, zone 1; and for the new line character, zone 0, will be sent to the look up table.

If the user lifts a finger while trying to press down, when they place it back to the zone under their lifted finger, the corresponding key will be reregistered according to the exact point, p(x, y), of the new touch. Since we do not want this key to enter any other key’s zone, we set the minimum, \( X_{\text{min}} \) and the maximum, \( X_{\text{max}} \) for each key. This way, when the point of the touch relocates the key, x coordinate of the new key will still be within \( X_{\text{min}} \) and \( X_{\text{max}} \), that was defined when the user registers the keys at the start. The y coordinate will be able to change without any constraints, because movement on the y axis will not cause an intersection of zones. Roughly
speaking, the keys will be moving with the fingers, but within a limited area on the x axis, so that they do not intersect with each other.

5.2 By Lifting

In this method, the same UnMousePad functions are used. However, this time, most of the work is done by the UnMousePadUp() function.

The user will place six fingers in the same way as in the previous method, and will hear the beep, as well, when the positions of the keys are registered. Every time a touch or touches occur, the red circles will be shown again. UnMousePadCallback() function will receive these touches. However, this time, UnMousePadUp() function will also receive the touches and will process them in the following way: all six keys are currently pressed down. The user will be continuously pressing these keys and lift any combination of them to enter the Braille input. These six keys, with zone numbers (0-1-2-4-5-6), are put into an array of ‘lifted keys’, even though they are not actually being lifted. The space key, is not added to this array, since it will be entered by clicking anywhere on the device. Then, a loop will go through all the current touches that is retrieved from ump.touches() array. If a touch is within a zone, that zone is removed from the lifted keys array. This way, only the zones that are not currently touching, meaning they are lifted, will be in the array. Then these zones, will be sent to select key array, but after the removal of the duplicates, and the same process will be followed as described in the by pressing method above.

For example, after the user presses down with six fingers, she/he lifts the left ring and index finger, the lifted keys array will contain zones 0 and 2, which corresponds to 1 and 100 in
the binary representation. The sum of these, 101, will be mapped into the letter ‘K’ in the look up

table.

To enter space character, the user will simply touch with his thumb on the fourth key,
which is between the keys that are touched by index fingers.

The movement of the keys is also the same as the first method. This time, after a lift of a
finger occurs, the user will place the finger back to it is original position. The point of touch,
when he/she places the finger on the corresponding zone, will be slightly different. So we again
let this touch to reregister the new position for that key with the same constraints described
above.

5.3 Kobigraphs

During our research, we tried to find the best way for a blind user to write Braille using a multi
touch device. Addition to regular Braille typing, as done with the mechanical and electronic
Braille writers, we found an interesting way to write Braille.

Kobigraphs was developed as a simple way of writing Braille in inkprint, since it is much
simpler than dots, to write by hand [2]. The idea is to form a symbol by joining the dots of a
Braille cell. All the lines are straight, except for the letter ‘K’.

![Figure 5.3.1: Kobigraphs.](image)

Kobigraph symbols are represented by joining the dots of the Braille cells. This is done by using as few number of lines as possible.
We adapted this method to our Brailler. The user can draw strokes anywhere on the IMPAD. This is a position independent method. So she/he doesn’t need to know on which point she/he is touching the IMPAD. The only thing is to track the direction of the stroke, which is whether it will be drawn in the NORTH, EAST, SOUTH, WEST, NORTH EAST, SOUTH EAST, SOUTH WEST or NORTH WEST directions. These directions are indexed from 1 to 8, respectively.

![Figure 5.3.2: Directions for Drawing Kobigraphs. The directions are numbered from 0 to 8.](image)

When the user starts to draw a stroke, the `unMouseMove()` function will keep track of `moveX` and `moveY` parameters which represent how much the finger moved in x and y axis, respectively. Then these parameters will be checked to see which direction the movement is actually made. For example, if `moveX` is positive on the x axis, but `moveY` is almost 0, we say an ‘East’ move has been made and if `moveX` is positive and `moveY` is negative, it will be
accepted as a ‘South-West’ move. Every move will correspond to an index number from 1 to 8, as seen in the figure, and these numbers will be added to a buffer.

The user can make more than one stroke for input, without lifting hand. When she/he lifts her/his hand, the `UnMousePadCallback()` function will check that the number of the touches is equal to zero, meaning the input is entered and ready to be mapped into the corresponding Braille character. It will also check the buffer that has the indexes for the moves. If the buffer is not empty, then it will be send to the look up table.

For example, letter ‘B’ is represented with dots 1 and 2 in a Braille cell. When these dots are connected, the result will be a straight line in a downward direction. Using Kobigraphs, the user will draw a straight line in the South direction. This will be sent to the buffer as 3, which will correspond to ‘B’ in look up table. To enter the letter ‘N’, one can draw in the order: East, South, South-West, or in the order: North-East, North, West. These directions of the moves will be sent to the buffer as 2-3-7 or 5-1-4, respectively. Both 237 and 514 will correspond to letter ‘N’ in the look up table.

![Diagram showing two different ways to represent 'N'.]

**Figure 5.3.3: Two Different Ways to Represent N.**
Some Braille cells are similar in the Kobigraph representation. For example, letter ‘B’ is represented by joining the first and the second dots; letter ‘L’ is represented by joining the first, second and the third dots. In both cases, Kobigraph representation will be a straight line that is in the downward (south) direction, but the line for letter ‘B’ is shorter than the line for letter ‘L’. To differentiate these two symbols, and to show that one is actually longer than the other, we added another gesture. The user will be touching the IMPAD with his thumb (or any finger) while his index finger will be making the stroke for letter ‘B’. So while making the stroke for ‘L’, the unMousePadDown() function will receive two touches, which will not happen in the case for letter ‘B’. This additional touch is applied to differentiate the similar Kobigraph symbols: ‘B’ from ‘L’, ‘O’ from ‘K’, ‘F’ from ‘P’, ‘G’ from ‘Q’ and finally, ‘H’ from ‘V’.

Figure 5.3.4: Characters with Similar Kobigraph Representations. The characters that are circled in same color have the same directional strokes. To differentiate these symbols, the user will touch with another finger while making these strokes.
To enter the space character, a stroke in the West direction; to enter new line character, the user will have to touch with one finger while making the stroke in the West direction with the other finger. For backspace, the user will make a long stroke in the North East direction, using two fingers. To switch between letters and numbers, a longer version of letter ‘C’ will be made, which is again done by touching with one finger and making a line in the East direction with the other with the other.
CHAPTER 6

RESULTS

The first method, that is entering the Braille input by pressing, is able to satisfy the need of an inexpensive Brailler. Our user's feedback was positive in the sense that he was able to write Braille as fast as he does on a Braille writer. For now, he was able to create sentences and delete the wrong typed letters. He tested the device using a paper with six soft buttons on it, placed on the IMPAD. This way, he knew the exact position of the keys as he first registers them. So, he was able to lift his hand and replace it in the correct position. Without these soft buttons, the Brailler was error prone compared to any other Braille writers. However, our latest version of the Brailler do not require these buttons. With the addition of some code, now the keys can be repositioned, even after lifting the hands.

The second method, in which the Brailler receives the input, as the user lifts her/his fingers, turned out to be inefficient. The reason is, while entering an input, the user has to keep their fingers on the IMPAD continuously and lift any combination of the fingers that corresponds to the dots of the desired Braille cell. The user does not need to apply a lot of pressure on the touching fingers, but still to keep hands on the device can be tiring. Also, it is confusing to lift some fingers while some others are touching the device. Because, in every day usage of the keyboards or Braille writers, entering input is done by pressing down, this method, as being different, would require training to get used to it. The experience showed that first method is more usable and efficient in entering Braille input on the IMPAD.

Another problem with both the methods was the rate of the speech output. As he enters the Braille letters one by one, with a speed that he would enter on a keyboard or a Braille writer, the corresponding
sound files for those letters are played in the order he enters them. If he enters another Braille character at a time that the sound for the previous letter is about to be played, the program will jump to the part where the sound for the current letter is played, hence the previous one will not be heard. This problem can be addressed by adding text to speech software, which can be much faster in outputting the sound.

The third method, which is based on the Kobigraphs, has not been tested by any blind user. It has been tested by sighted users, and turned out to be easy to write if the user is familiar with writing Braille. Since it is position independent, it can be easily used by a blind user.
CHAPTER 7

Conclusion and Future Work

Our research about how to write Braille on a multi touch device, resulted in three different methods. First one -by pressing- turned out to be usable as tested by a blind user. The second one –by lifting- resulted in an uncomfortable input device. The last one –based on Kobigraphs- has not been tested by a blind user. However, the experience by sighted users, shows that it is easier to use compared to other methods, since it requires at most two fingers. The user can be holding the small sized IMPAD in their one hand and make strokes with the other hand, just like they use their PDAs. Because it is position independent, it does not suffer from the absence of visual display [4]. This way, the inaccessibility of the multi-touch devices can be addressed.

These results show that our research can be a good step to remove the gap between visually impaired environments and multi touch devices. Also, as the research continues, an inexpensive and flexible Braille writer can be finally used every day, at schools or offices.

Our future work will be to improve the techniques we used in designing our Brailler. Also, we will continue to look for other ways to increase the accessibility of blind people entering input on IMPAD. Since blind users already have PDAs that come with screen readers,
by connecting IMPAD to their PDA, we will implement more methods which make their interaction with PDAs more efficient.
Bibliography


