An Effective Fingers Detection Method for Braille Touchscreen Keyboard on Tablet Devices

Puthnith Var, Teresa Gonçalves, and Miguel Barão

Departamento de Informática,
Universidade de Évora, Portugal
`puthnith@gmail.com,{tcg,mjsb}@uevora.pt`

Abstract. In last decade, tablet devices become more convenient and comfortable electronic devices than laptop. Its touchscreen's property creates a natural feeling for users while moving objects that appear on the screen. However, the loss of pressing-buttons sensation requires the users to look at the screen during touching and gesturing, and every taps provide the same feeling to the users' fingers. This type of problem discourages the visual impaired people to use the devices. Therefore, the paper aims to challenge the difficulty to hearten them to compose with and use tablet devices more than to only listen to the devices' audio. Additionally, learn how to type in a new way is tragedy. The new approach is developed to create an effective fingers detection for braille touchscreen keyboard application and to provide the same method of typing like they press buttons of their old style braille keyboard. The proposed method can track and identify all the eight fingers in both upright and upside-down directions of the tablet devices. The paper also explains in detail of every steps of the method and forms a complete algorithm at the end with the considerate analysis and results. The proposed method fulfills the requirement for the visual impaired people.

Keywords: fingers detection, braille, touchscreen, keyboard, tablet

1 Introduction

In the last few decades, technology has changed the way humans communicate and made their life easier. People can send e-mails and text messages, share ideas, and get feedbacks in just a few seconds. The use of laptops is reduced by smart phones and tablets for texting, talking, and sharing any moment of their life on social networkings such as Facebook and Twitter. These devices provide not only communication but also games, books, newspapers, movies, music, camera and more. The touchscreen technology helps users to interact with the devices in many different ways such drawing, scrolling through contents, pinch or zoom and rotate their photos.

However, touching on the screen's virtual keyboard provides different sensations and natural feelings than pressing buttons on computer's keyboard. On a computer's keyboard, the users are able to rest their fingers during typing and it allows them to recognize buttons staying under their fingers along with the sense of knowing a key is pressed or being pressed. However, this type of feeling, the touch-and-press action, is not possible to obtain while typing or touching on smart phone and tablet devices despite the fact that it is being touched or pressed lightly or heavily. Typically, the users are required to look at the device's screen to know whether they touch the right area of the virtual buttons or not.

After all, this type of the touchscreen devices discourage people who are visual impaired. When they touch the screen, they cannot track the location of the virtual buttons. Even though some devices provide responding voice [2][3] which is not much comfortable, the devices create problem of not letting the users to express their own thoughts and feedbacks as written texts. They, however, can record their voices but doing it in public places such as bus or train is disturbing and loses their privacy. Thus, the system has to be more intelligent to detect the fingers location to provide eyes-free composing and it is strongly required for the visual impaired people.

People with visual impairments use braille characters [1] to read by touching and they compose with a device called braille keyboard. In order to compose on a tablet device, they need an extended braille keyboard hardware that is connected to the device via the Bluetooth technology or a cable. The way they read or understand is different from examining by touch on a surface of rough braille letters. They listen to a responded voice or audio [2][3] from the tablet to get information by touching on the tablet's screen. However, composing without looking at the screen has been implemented by many companies and researchers. They provide different methods to help people to text faster without looking at a device's screen. Tinwala and MacKenzie's system [4] was developed based on graffiti strokes for eyes-free text entry. However the graffiti strokes is slow comparing to tapping and users can write in different ways from the defined graffiti strokes. Bonner, Brudvik, Abowd, and Edwards [5] divided the touchscreen area into many different blocks and the users have to learn and remember each blocks to compose words or sentences. Besides, the systems in [4][5] do not focus on only fingers detection development and the systems target normal people rather than visual impaired people, which is helpless. Frey, Southern and Romero [6] built the *Braille Touch* application only for smart phone devices which allows users to type braille letters on the defined areas of fingers. The evaluation in [7] of the application, made by the developers and their colleagues, is acceptable but it only applies to the small screen's size devices which is useless to deal with fingers detection. Azenkot, Wobbrock, Prasain, and Ladner [8] designed the *Perkinput* fingers detection using Maximum likelihood to detect which fingers touch the screen based on user-set reference points. However, the method works with only six fingers and the device is required to be in the upright position. A Stanford student, Adam Duran [9], developed the *Touchscreen Braille Writer* application which allows users to type the braille letters on a tablet device with eight fingers. However, the fingers detection method is restricted to the upright position in landscape mode. Inpris company [10] built the *UpSense* application to help both normal and visual impaired people to type faster on the tablet devices. Unfortunately, they implemented a lot of gestures which require users to learn and the fingers detection is limited and works only in the upright position. The state of the art shows that the careless of fingers detection at the fingers calibration or initialization step can lead to wrong detection in the identifying braille letters.

The paper aims to challenge the difficulty to hearten the visual impaired people to compose with tablet devices. We deal with eight fingers on the touchscreen of tablet devices. Our proposed algorithm provides the users to use the devices in upright or upside-down position of landscape or portrait mode. Along with the feature, if a visual impaired person suddenly change his position, the algorithm automatically corrects the location of the user's hands so that the user can type correctly.

The paper is divided into 4 sections. The second section is our proposed method which simplifies in detail of the proposed fingers detection method. An analysis and results shows how good the proposed method applied to a touchscreen devices is in the third section. The fourth section provides a conclusion and further works which gives a brief summary on what we have done and what our next paces are.

2 Proposed Method

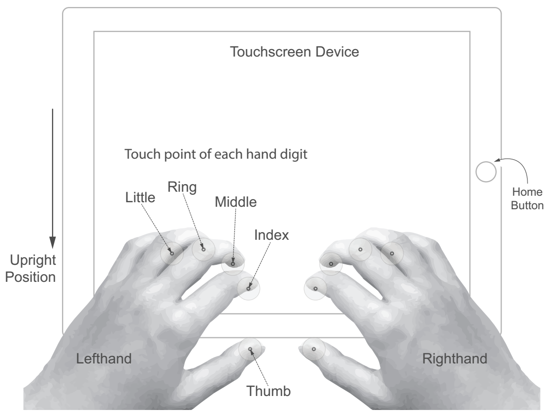

Fig. 1. The natural position and shape of a user's hands putting on a touchscreen device: The direction of the hands points toward each other rather than forms parallel lines.

The multiple-touch feature provides many possibilities to enhance the way people use smart phone and tablet devices such as gesturing with fingers. A tablet device's gesture recognizer sends information to the device how a user's fingers perform on the device's screen such as the location and the events of each touch in 2 dimensional coordinate system. The touch's events are: touches-began tells the screen that a use began to touch, touches-moved informs that the touchpoints' locations have changed, touches-ended tells that the fingers are lifted off the screen and touches-canceled informs when there is any interruption such as a phone call during gesturing. Thus, it provides a rich ability to identify and classify the natural gestures of a user's hands such as tap, swipe, pan, scroll, zoom or pinch, rotate, scroll, long press, and so on. Unfortunately, identifing which fingers are touching on the screen is unavailable. A smart phone or table device cannot give a specific information of which part of a human body is touching its screen. Likely, a toe or nose's point can also tap on the screen. Therefore, to handle multiple touches or fingers at the same time on a device's screen for people with visual impairments is the most arduous. The proposed method aims to identify all the 8 fingers for braille touchscreen keyboard application.

A single tap on a device's screen makes no probability to identify which finger or hand digit it is. However, to identify the hand digits, the proposed method studies with 8 touches at the same time, which typically focuses on dealing with the braille virtual keyboard for visual impaired people. We begin with a study of a pattern of a human's fingers.

Fig. 1 shows the natural position and shape of a user's hands using a tablet device's touchscreen keyboard. The direction of the left and right hands intersects each other by preference more than parallel lines. This is because the parallel hands' direction is difficult to achieve on the not-too-big touchscreen when a user uses the tablet close to his body. In addition, it is rare that people try to type on a tablet far away from them. The intersecting direction of hands provides the method more accurate to detect all the 8 fingers.

The paper experiments 4 fingers of both hands: index, middle, ring, and little fingers. The thumb fingers are excluded because visual impaired people use only 6 fingers to compose a braille letter and the 2 little fingers to go backward and forward, or to delete and space. The thumb fingers are used for understanding the typed braille characters. Fortunately, the tablet device can produce an acknowledgment voice which is used to give feedback to the users to know what they are doing.

The proposed algorithm are divided into five parts. The first part is finding 2 groups of touchpoints which is assigned to the left and right hands at the end of the algorithm. Finding slopes and hands direction of the 2 groups are described in the second and third parts, respectively. We provides the defined fingers for both hands in the fourth part. In the last part, we assemble all the techniques described in the previous parts and give the complete proposed algorithm.

2.1 Touchpoints Grouping

The proposed method first requires a user to put all the 8 fingers on a device's screen gently and naturally. A tablet device provides the 8 touchpoints' locations on the screen with random time sequence of the touches. Thus, there is no information in which touchpoints are located at the left and right sides of the screen, even though the user put his hands at the same time. In this section, we use hand digits instead of the name of the fingers to avoid the confusion which the algorithm may swap the hands at the last decision. There are two matrices group A and B which are the groups of four identified touchpoints $[a_1, a_2, a_3, a_4]$ and $[b_1, b_2, b_3, b_4]$, respectively. The matrices A and B are defined as below:

$$A = [a_1, a_2, a_3, a_4]^\top \tag{1}$$

$$B = [b_1, b_2, b_3, b_4]^\top \tag{2}$$

It is noted that the group A is always a set of 4 touchpoints located at the left area and the group B is another set of 4 touchpoints located at the right area of the touchscreen.

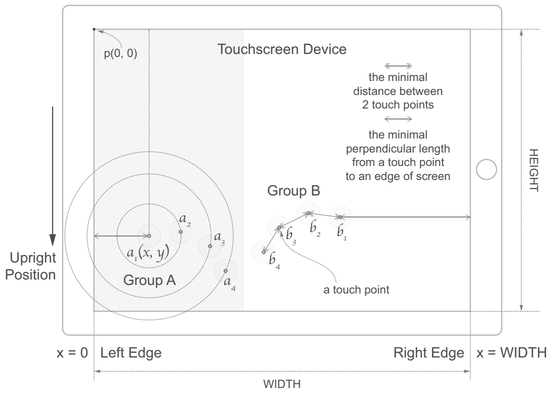

Fig. 2. The illustration of the measurement of the 8 touchpoints' location on a touchscreen device: to identify each group's touchpoints, the perpendicular length from each touchpoint to the edges of the screen and the distance between 2 touchpoints are calculated. Basically, the point a_1 and b_1 is the closest or minimal points to the left and right edges of the screen, respectively.

Fig. 2 shows that the 4 touchpoints of the group A and B are located at the left and right areas, respectively. It is noted that the left area can be bigger, smaller or equal to the right area according to the point a_4. The areas are identified at the end of this part.

Each point $p \in P$ that contains an ordered pair (x_p, y_p) for a horizontal x and a vertical y lengths from the origin, respectively, is defined as follow:

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\} \tag{3}$$

P is a set of the 8 random unidentified touchpoints on a tablet device's screen ordered by the time sequence of touches made by a user.

We know that the identified touchpoint a_1 in group A (1) is the point that has the minimal perpendicular length to the *left edge* comparing with other touchpoints. Thus, the point a_1 is defined as the minimal or shortest x_p length of $p \in P$ because the *left edge* has $x = 0$. The identified points a_2, a_3, and a_4 are found using the *distance* measurement between 2 touchpoints. In this case, we prevent the touchpoints that have the same x or y length. The *perpendicular length* and *distance* between points are defined as below:

$$Length(C, o) = c_{min}\,|x_c - x_o| \tag{4}$$

$$Distance(C, o) = c_{min}\left(\sqrt{(x_c - x_o)^2 + (y_c - y_o)^2}\right) \tag{5}$$

C is a given set of points where $C = c_1, c_2, \dots, c_N$ and $c_{min} \in C$ is constraints on $(1 \leqslant min \leqslant N)$. o is a given point which is used to compare with the points in the set C.

From the equation (1), (4), and (5), we define the group A function of x, where $x = \{1, 2, 3, 4\}$ is a touchpoint's index of $A(x)$ (1). The $A(x)$ function is defined as below:

$$A(x) = \begin{cases} Length({}^{A}E_1, 0) & \text{if } x = 1 \\ Distance({}^{A}E_x, A(x-1)) & \text{otherwise} \end{cases} \tag{6}$$

${}^{A}E$ is an unidentified subset of P created to eliminate the unidentified touchpoints in the set P that assigns to the identified touchpoints in the group A where ${}^{A}E_x = {}^{A}E_{(x-1)} \setminus \{A(x-1)\}$ and ${}^{A}E_1 = P$.

The equation (6) shows that the identified touchpoint a_1 is always the closest point to the *left edge* and other touchpoints in the group A (1) are recognized after finding the point a_1. After an unidentified touchpoint $p \in {}^{A}E$ is assigned to an identified touchpoint $a \in A$, the point p is removed from ${}^{A}E$. This eliminates the *zero* distance finding in the next touchpoint. The identified touchpoint a_2, a_3, a_4 has the minimal distance of a touchpoint $p \in {}^{A}E$ comparing with a_1, a_2, a_3, respectively.

The recognized points b_1, b_2, b_3, and b_4 in the group B (2) are identified using the similar procedure. However, we use the unidentified subset ${}^{A}E$ instead of the set P. In this case, less computational execution time and improving system performance can be achieved. From the equation (2), (4), and (5), we define the B function of x as below:

$$B(x) = \begin{cases} Length({}^{B}E_1, WIDTH) & \text{if } x = 1 \\ Distance({}^{B}E_x, B(x-1)) & \text{otherwise} \end{cases} \tag{7}$$

where $x = \{1, 2, 3, 4\}$ is a touchpoint's index the matrix group B and ${}^{B}E$ is the subset of ${}^{A}E$ which is created for eliminating the unidentified points that is assigned to the points in the group B. ${}^{B}E_x = {}^{B}E_{(x-1)} \setminus \{B(x-1)\}$ and ${}^{B}E_1 = {}^{A}E_{size(A)+1}$ where $size(C)$ returns the number of element in a set of points C.

Likewise, in equation (7), we first find the identified point b_1 in the matrix group B (2). However, we look for an unidentified point $p \in {}^{B}E$ that has the $|x_p - WIDTH|$ minimal length rather than x_p length because we compare the point p with the *right edge* of the screen with the horizontal value $x = WIDTH$. Therefore, the recognized points b_2, b_3, and b_4 have the minimal distance of touchpoint $p \in {}^{B}E$ comparing with the points b_1, b_2, and b_3, respectively.

After the matrices group A and B are defined, the algorithm requires them to be validated. The algorithm requires the 8 touchpoints are composed by 2 hands rather than random touchpoints. The validation lessens the complexity in the next part.

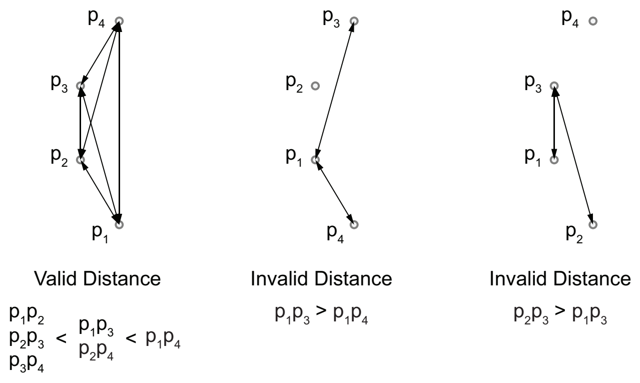

Fig. 3. The Fingers Validation technique: in order to be recognized as the 4 fingers of a hand, the identified touchpoints p_3 and p_4 have to be in between the points p_1 and p_2. The proposed valid distance has to be satisfied.

Fingers Validation We suppose there is a group $Q = [p_1, p_2, p_3, p_4]^\top$ to be validated. Fig. 3 shows the proposed valid distance used for recognizing the 4 hand digits of a hand: little, ring, middle, and index fingers. The distances $\{p_1p_2, p_2p_3, p_3p_4\}$ are shorter than the distances $\{p_1p_3, p_2p_4\}$. The distance p_1p_4 is the longest comparing with the others. If the valid distance is not fulfilled, another identified point $p \in Q$ is selected to be point p_1 where p is the next minimal perpendicular length to the related edge of a device's screen. Thus, the identified points p_2, p_3, and p_4 are re-identified again using the equation in (6) or (7) accordingly. In case that the valid distance is not satisfied after each point in Q takes turn to be the point p_1, the user are required to re-input the hands again. In other words, if the user tries to form a shape of the hands in inappropriate ways, the algorithm rejects the given touchpoints.

Fingers Validation Algorithm

```
FingersGroup* fingersValidation(FingersGroup* Q, float edge) {
  int count = 1;
  BOOL x = TRUE;
  do {
    x = validDistance(Q);
    if (x) break;
    else {
      count++;
      if (count < size(Q)) Q = nextMinimalLength(Q, edge);
      else break;
    }
  } while (1);
  if (count == size(Q)) reinput();
  else return Q;
  return nil;
}
```

After the fingers validation is verified, we have the 2 valid groups of fingers or touchpoints. We then can define that the *left area* is the area from the *left edge* to the identified point a_4 of the group A (1) and the *right area* is the area from the point a_4 to the *right edge*. However, there is also a case when a user tries to put the left hand over right hand or vice versa. In this case, the *left area* may contain some identified points of group B (2). There is no issue at all with the case.

2.2 Slopes Identifying

This section, we deal with identifying the slope of each group. By understanding the slope and the shape, which describes in the next part, we can distinguish which group is the left or right hand. The slope helps to reduce the complexity in the next part.

In this part, we study over 2 important touchpoints of each group, the 1st and 4th identified touchpoints. Fig. 4 shows 4 different types of slope that can possibly be happened, negative, positive, zero, and no slopes. We suppose that there is a group $Q = [p_1, p_2, p_3, p_4]^\top$ to be examined. An alpha α is the angle $(0 \leqslant \alpha \leqslant \pi/2)$ at the intesecting point between the horizontal line at the point p_1 and the line p_1p_4. If the tangent of the alpha $tan(\alpha) < 0$ or $tan(\alpha) > 0$, the algorithm identifies the slope as *negative* or *positive slope*, respectively. Both slopes are defined as *one slope* which will be used in the next part. If the alpha is zero $(\alpha = 0)$, we identify as *zero slope*. The last slope is *no slope*. It happens when the alpha is equal to $\pi/2$ $(\alpha = \pi/2)$. In this case, the proposed algorithm requires the user to re-input the hands again. However, its probability is so small

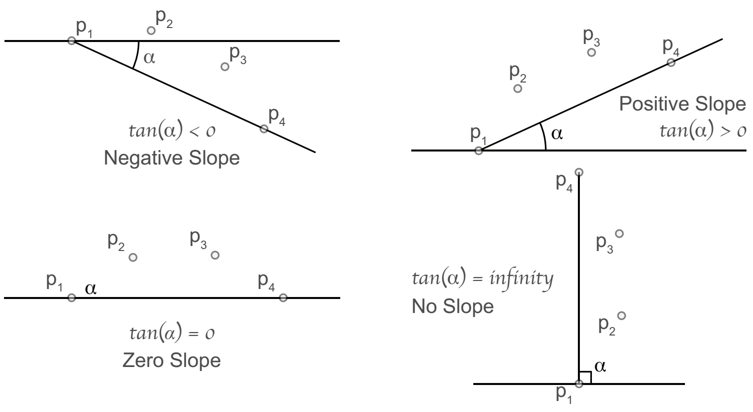

Fig. 4. Slopes Identifying: there are 4 types of slope, negative, positive, zero and no slopes. The probability of zero and no slopes are smaller than 0.5%. However, those cases are needed to avoid errors and strengthen the performance.

and it rarely happens. In the next section, we describe how the proposed method recognize the hands direction with the *one slope* and *zero slope*, accordingly.

2.3 Hands Direction Recognizing

The proposed algorithm provides a user the ability to use a tablet device in both upright and upside-down positions of the landscape or portrait mode. The user is able to distinguish whether the device is in the upright position or not. In this part, the hands direction is recognized differently according to the slopes that are identified in part 2.2.

There are only 2 types of hands direction, up and down directions. The 2 groups (1)(2) are required to be whether they both points upward or downward. In case of one-up-one-down direction, the proposed method rejects the input fingers and asks the user to re-input again. This prevents the user's confusion of left and right areas. We study a vector that is projected from the line p_1p_4 according to the slope found in part 2.2. Basically, we can understand a hand is up or down correspondingly to the vector. However, the 4 touchpoints of a group can stay in the same line or have the same slope. In this case, finding the direction of the vector is probably impossible. Thus, the proposed algorithm decides to choose up direction due to high probability that users use tablet devices in the upright position.

One Slope If the slope is not zero, the hands direction recognition has more mathematical equations. Fig. 5 show the three steps of finding the vector $\overrightarrow{\mathbf{SM}}$ where M is the intersection point of the lines p_1p_2 and p_3p_4 and S is the foot

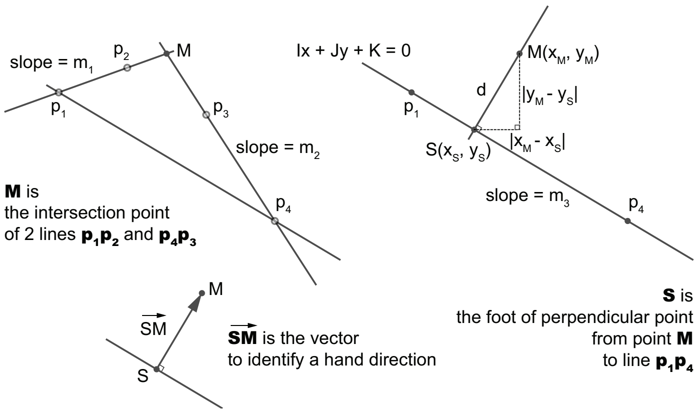

Fig. 5. Hands Direction Recognizing for *one slope*: To find a hand direction of the $\overrightarrow{\mathbf{SM}}$, points M and S are required. The point M is the intersection point of the lines p_1p_2 and p_3p_4. The point S is the foot of perpendicular from the point M to the line p_1p_4.

of perpendicular from the point M to the line p_1p_4. The point $M(x_M, y_M)$ is defined as follow:

$$\begin{aligned} x_M &= \frac{b_2 - b_1}{m_1 - m_2} \\ y_M &= -\frac{b_1 m_2 - b_2 m_1}{m_1 - m_2} \end{aligned} \tag{8}$$

The variables m_1, m_2 and b_1, b_2 are the slopes and y-intercepts of the line p_1p_2 and p_4p_3, respectively. The equation of a line is written as $y = mx + b$.

The point $S(x_S, y_S)$ is Fig. 5 is defined as follow:

$$\begin{aligned} x_S &= \frac{x_M + m_3 y_M - m_3 b_3}{m_3^2 + 1} \\ y_S &= -\frac{m_3 x_M + m_3^2 y_M + b_3}{m_3^2 + 1} \end{aligned} \tag{9}$$

The variables m_3 and b_3 are the slope and y-intercept of the line p_1p_4. The equation of the line p_1p_4 can be written as $-m_3x + y - b_3 = 0$. Thu,s $I = -m_3$, $J = 1$ and $K = -b_3$ as $Ix + Jy + K = 0$. The x_S and y_S are derived from $\frac{x_S - x_M}{I} = \frac{y_S - y_M}{J} = \frac{-(Ix_M + Jy_M + K)}{I^2 + J^2}$.

The vector $\overrightarrow{\mathbf{SM}}$ shows the direction of each hand points to. If the vector is has negative in y axis, it is the *up direction.* The *down direction* is when the vector has positive in y axis.

Zero Slope If the slope is zero or flat, the hands direction recognition is easier and simpler to be recognized. The vector $\overrightarrow{\mathbf{p_1p_2}}$ and $\overrightarrow{\mathbf{p_4p_3}}$ tells the exact direction of both hands point to.

Hands Detection Algorithm If the direction is *up*, this means a user use the touchscreen device in upright direction. Thus, we define the matrices group A (1) and B (2) as the *left hand* and the *right hand*, respectively. However, if the direction is *down*, the tablet device is used in upside-down direction. Thus, the matrices group A is the *right hand*, and group B is the *left hand*. In this perspective, if one group is identified as the *left hand*, another group has to be the *right hand*. The script below is the hands direction recognizing procedure.

Hands Detection Algorithm

```
Hands* handsDirection(FingersGroup* A, FingersGroup* B) {
  if (direction(A) == UP && direction(B) == UP) {
    A.hand = LEFT;
    B.hand = RIGHT;
  }
  else if (direction(A) == DOWN && direction(B) == DOWN) {
    A.hand = RIGHT;
    B.hand = LEFT;
  else {
    return nil;
  }
  Hands* Hs = HandsInitial(A, B);
  return Hs;
}
```

direction method uses the *one slope* or *zero slope* algorithm accordingly to return *UP* or *DOWN* signal from the input group A and B.

2.4 Fingers Defining

Until this part, we are able to identify which group (A in (1) and B in (2)) is the left or right hand. Thus, we define hand digits accordingly to the left and right hands. If the group A is the left hand and the group B is the right hand, we define:

$$L = [ll, lr, lm, li]^\top \tag{10}$$

$$R = [rl, rr, rm, ri]^\top \tag{11}$$

$$L = A \quad \begin{bmatrix} ll \\ lr \\ lm \\ li \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \qquad R = B \quad \begin{bmatrix} rl \\ rr \\ rm \\ ri \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{12}$$

L is the matrix of the *left hand* where ll, lr, lm, and li are the left hand digits, *little*, *ring*, *middle*, and *index* fingers, respectively. R is the matrix of the *right hand* where rl, rr, rm, and ri are the right hand digits. Thus, the group A and B are equal to the left hand matrix L and the right hand matrix R, respectively.

If group A is the right hand and group B is the left hand, we inverse the equation (12) as $R = A$ and $L = B$.

$$F = \begin{bmatrix} ll \\ lr \\ lm \\ li \\ \begin{bmatrix} 0\,0\,0\,1 \\ 0\,0\,1\,0 \\ 0\,1\,0\,0 \\ 1\,0\,0\,0 \end{bmatrix} \times \begin{bmatrix} rl \\ rr \\ rm \\ ri \end{bmatrix} \end{bmatrix} = \begin{bmatrix} L \\ aD \times R \end{bmatrix} = [ll, lr, lm, li, ri, rm, rr, rl]^\top \quad (13)$$

aD is the anti-diagonal identity matrix used to swap all rows in the matrix R. The equation in (13) identifies the fingers and hands after the matrices in equation (12) are assigned. F is a matrix of the 8 hand digits which is used for identifying braille characters in braille touchscreen keyboard.

2.5 The Proposed Algorithm

The complete proposed algorithm of fingers detection for braille touchscreen keyboard application is written in pseudocode as shown below.

The Proposed Algorithm for Fingers Detection

```
Divide all given 8 touches points into two groups -> A and B
    Use equation (6) and (7) to group
    Validate each group using the valid distance in Fig. 3
Identify the slope of each group -> slopes (one, zero)
    Ask the user to re-input the fingers if the slope is infinity
Recognize the hands direction of the two groups -> LEFT and RIGHT
    Hand Direction is:
    If (all points = same slope) UP
    Else
        If (one slope)
            Use equation (8) and (9) to find points M and S
            Find vector SM
        Else / zero slope
            Find vector p1p2 and p4p3
        If (y axis of the vector = negative) UP
        Else DOWN
    Use Hands Detection Algorithm -> LEFT or RIGHT hand
Define the fingers accordingly to LEFT or RIGHT hand of each group
```

```
-> all hand digits are assigned
  Use equation (10), (11), and (12) to assign to all fingers
  Use equation (13) to identify braille letters
```

3 Analysis and Results

The proposed method is designed to implement at the calibration or initialization step of a braille touchscreen keyboard application on tablet devices. The method requires 8 touchpoints from a user and provides the system with an accurate information of each hand digits. It is flexible to any position of the user's hand which allows the user to use the device straightforwardly.

The proposed fingers detection method is created specifically for a braille touchscreen keyboard, which does not apply to any general fingers detection that a user can touch on a tablet device's screen in different forms and positions of the fingers. This is not a drawback of the system because to understand every positions of a user's fingers, the fingers' preference is provided and it requires a large computational execution and time. This is not the case of the paper to deal with.

The proposed algorithm identifies accurately all the 8 given touchpoints with the hand digits. However, a user can imitate the proposed shape, hands direction, and slope in different ways. For example, they can use the *right hand* instead of the *left hand* and vice versa by crossing the hands. In this case, the alogrithm will provide them an incorrect information. The algorithm only gives the right information if the user input the fingers in the natural way. However, after the input, the user than can change the hands to any forms he likes without affecting the change of the fingers order.

Another case, if the user puts the device in the landscape mode and tries to use it as the portrait mode, the proposed algorithm definitely rejects the input fingers because the directions of the hands are opposite each other.

However, the user can put one hand on top of another hand in the case of the 2 hands stay far away from each other. In this case, the user may use the tablet device in portrait mode which only provides him a small space to align the hands horizontally. The proposed algorithm provides a great fingers detection for this case because after 4 adjacent touchpoints are eliminated or grouped, the remaining 4 touchpoints are selected to be another group.

Eventually, the performance and results are acceptable and accurate for implementing in a virtual touchscreen keyboard on tablet devices for people with visual impairments.

4 Conclusion and Further Works

In conclusion, we can see that tablet devices provide a lot of advantages and functionalities to aid people in many different ways. However, the devices cannot fulfill the requirement of people with visual impairments because of the lack of pressing buttons sensation. Therefore, the paper challenges the difficulty and provides the proposed algorithm to deal with the problem. We see that the method provides the accurate hand digits of all the given touchpoints. The algorithm rejects all the invalid given touchpoints if they cannot recognize as a human's left and right hands. Furthermore, a user can use a tablet device in any position whether it is upright or upside-down of the landscape or portrait mode. This gives them a wild range of uses of the keyboard. With this result, we are grateful to work on our next paces which we expected to provide the best braille touchscreen keyboard application to people with visual impairments.

This paper is our first step to let visual impaired people be able to compose and express their thoughts on smart phone and tablet devices. This paper only deal with fingers dection method for tablet devices. Thus, fingers detection method on smart phone devices is also our next pace. Besides, we planned to work on fingers prediction which analyzes and tracks the change of the fingers' location and provides the best estimation. The complete braille touch keyboard application will be implemented using all the algorithms we have done and are working on. Finally, we hope that the output of the research gives advantages to not only the people with visual impairments but also to the researchers and developers to contribute and build a better world together. Living in this world, we are all equal and we can do everything as long as there is hope.

Acknowledgements

I would like to express my noteworthy thanks of gratitude to my professors Miguel Barão and Teresa Gonçalves who gave me the golden idea and opportunity to use my kownledge to help visual impaired people, which also provided me in doing a lot of research and knowing many new things.

I also would like to thank my parents, my sister Puthneath Var, my fiancée Rithreaksa Khourn, my friend Prakash Poudyal, who lended me a helping hand to finalizing the paper within the limited time.

References

1. Braille, L.: Procedure for writing words, music and plain song using dots for the use of the blind and made available to them. Royal Institution Of Blind Youth, Paris (1829).
2. Speech Enabled Eyes Free Android Applications `http://code.google.com/p/eyes-free/`
3. VoiceOver `http://www.apple.com/accessibility/iphone/vision.html`
4. Tinwala, H., MacKenzie, I. S. Eyes-Free Text Entry on a Touchscreen Phone. In: Proceedings of the IEEE Toronto International Conference Science and Technology for Humanity TIC-STH 2009, 83-89. Toronto (2009)
5. Bonner, M., Brudvik, J., Abowd, G., Edwards, K.: No-Look Notes: Accessible Eyes-Free Multi-Touch Text Entry. In: Proceedings of Eighth International Conference on Pervasive Computing, pp. 409–426. Springer, Heidelberg (2010)

6. Frey, B., Southern, C., Romero, M.: BrailleTouch: Mobile Texting for the Visually Impaired. In: Proceedings of Human-Computer Interaction International, HCII. Orlando (2011)
7. Southern, C., Clawson, J., Frey, B., Abowd, G., Romero, M.: An Evaluation of BrailleTouch: Mobile Touchscreen Text Entry for the Visually Impaired. MobileHCI 2012, San Francisco (2012)
8. Azenkot, S., Wobbrock, J. O., Prasain, S., Ladner, R. E.: Input Finger Detection for Nonvisual Touch Screen Text Entry in Perkinput. In: Proceedings of Graphics Interface, Canadian Information Processing Society, Toronto (2012)
9. Stanford Summer Course Yields Touchscreen Braille Writer `http://news.stanford.edu/news/2011/october/touchscreen-braille-writer-100711.html`
10. Inpris UpSense `http://www.inprisltd.com/`