SwipeKey: A Swipe-based Keyboard Design For Smartwatches

1st Author Name
Affiliation
City, Country
e-mail address

2nd Author Name
Affiliation
City, Country
e-mail address

3rd Author Name
Affiliation
City, Country
e-mail address

ABSTRACT

The rise of smartwatches calls for efficient, convenient and suitable text input methods for these small computers. The minuscule size of these screens brings many challenges on how to interact with these devices. Most of users are familiar with QWERTY keyboard layout. But the smartwatches' screen size are usually too small for simple QWERTY without modification or assistant. Even with a fast recognition-based approach may need a deterministic fallback input method for error-correction or difficult-to-predict words. We introduce SwipeKey, a text input design that uses swipe directions to allow multiple inputs per button and thus allow for an increase in the effective size of input buttons. We have conducted thorough experiments optimizing SwipeKey to enable a fast, low-error, and easy to learn keyboard for smartwatches. These benefits result from having a keyboard that emphasizes the use of swipe motions. Our user study results show that with a specific combination of swipe directions and its corresponding button size, SwipeKey users achieved SwipeKey 4 speed of 11 in words per minute (WPM), a 53% improvement from baseline (7.2 in WPM) and dramatically decreased character error rate (CER) from baseline 10% to SwipeKey 4 3.4%.

ACM Classification Keywords

H.5.2 Information interfaces and presentation:: User Interfaces - Graphical user interfaces; Input devices and strategies.;

Author Keywords

Text entry, input, smartwatch, SwipeKey, keyboard design

INTRODUCTION

Smartwatches have become more and more popular in the consumer market. There are many recently released products such as the Apple Watch [3], LG Watch Urbane [19], Sony SmartWatch 3 [37], and Moto360 [26] that are more powerful and power efficient than ever before and make current smartwatches much more capable and practical than prior models.

Figure 1: SwipeKey 4 implemented on Sony SmartWatch 3

Smartwatches rely on small touchscreens and voice recognition for input. Most smartwatches only provide voice input or a selection of predetermined short messages as text entry. Because typing on such a small screen is very difficult with ordinary QWERTY soft-keyboards, a number of solutions to mitigate this problem of text entry on small devices have been proposed. Some of the solution use wristbands, chording, or other sensors [8, 21, 31, 2, 25], and some work on the design of the keyboard [18, 13, 29, 20, 11].

The major issue for smartwatch text entry is the difficulty of using these extremely small screens for touch input. For example: an iPhone 4 has a screen size of 76.65mm x 51.60mm, the Sony SW3 smartwatch has a screen dimension of 28.7mm x 28.7mm. The dimensions of the onscreen keyboard on an iPhone 4 are 51.6mm x 33.5mm, which is much larger than the entire screen size of the Sony SW3 smartwatch. Furthermore, we cannot reserve the entire screen of the smartwatch to display a software keyboard since it would leave no space to display text as the user is typing. If we fill half of the smartwatch screen to display an iPhone 4 software keyboard, each button of the keyboard would be 4.2 times smaller on the Sony SW3. A tiny keyboard and small button size decreases the users input speed and accuracy when typing on a smartwatch keyboard.[36, 15].

When we talk about the effective button size, we try to describe the ease of pressing buttons defined by several factors. One factor is size. Buttons that are both small and tightly packed together will make pressing a specific button difficult. Since smartwatch screens are naturally small, there are few

options that would allow for increased button size. This is why we need to incorporate other techniques to make the buttons bigger. One way Keyboard designers can achieve this is by using multiple gestures on a single button to reduce the button numbers and increase each button size.

There are lots of keyboards designated for small screen mobile devices that attempt to increase the effective button size [29, 4, 18]. They use many different methods to increase the effective size of the keyboard keys. However, Leiva et al. [18] shows that the proper design for various small-screen mobile devices can differ for each one of them. These works primarily focus on general mobile devices; for smartwatches specifically however, there is still a room to be explored.

In our paper, we focus on designing a keyboard layout that is suitable for most of the commercially available smartwatches. We have designed a keyboard and have optimized and evaluated several parameters crucial for smartwatch keyboard design through our design process and user studies. Figure 1 shows SwipeKey 4, one of our keyboard implementation with 4 direction swipe per button.

Our first contribution is that we implemented SwipeKey: a one swipe or tap keyboard for smartwatches. We tried to optimize several crucial parameters and have shown that SwipeKey can outperform one existing keyboard for small devices in speed, accuracy, learning difficulty, and user preference.

Our second contribution is that we study some of the parameters of keyboard design and touch screen input on smartwatches. We also provide some design principles for keyboards on smartwatches that have proven themselves in our user studies.

RELATED WORK

In this section, we will review some existing text entry methods on smartwatches.

Voice Input

Voice input is a popular input method available on most mobile devices.[14, 34] Currently, smartwatches also use voice input. For example, Apple Watch, Samsung Gear S [35], and Huawei Watch [12] support voice input for performing a variety of actions. Voice input provides a hands-free and high-speed text entry method for mobile device users [39, 33, 34]. However, voice input may not be socially acceptable in certain situations. For example, in public spaces it is not common to see people interacting with their devices using voice control. Furthermore, users may find voice input unreliable in regards to speech recognition accuracy. Voice recognition is not perfect, especially when the quality of the user's voice or accent and ambient sounds in the environment negatively affect accuracy. Therefore, to make text entry reliable across different scenarios, an alternative text input method is necessary for smartwatches.

Software Keyboard

Since smartphone users are accustomed to using touchscreen keyboards, software keyboard designers naturally want to find ways of fitting full-featured keyboards onto smartwatches.

Ambiguous Keyboard

Ambiguous keyboard is one approach to support text entry on small mobile devices with limited screen space. There are researches [20, 5, 9, 10] and products[28] to support a full alphabetic keyboard with a reduced number of buttons. These systems required disambiguation techniques to find out the associated letter from a multiple representing key set. However, the dictionary-based disambiguation may still need deterministic fallback input method for error-correction or difficult-to-predict words.

Word Gesture Keyboard

Word gesture keyboard such as ShapeWriter [42, 41] and SHARK [16], are stroke gesture-based text input method. User writes each word through trajectory of input key sequence that links the letters of the intended word on a conventional soft keyboard. A language model is required to select the most likely word from candidate words associated to the word gesture. Word gesture keyboards are common on smartphones. However, there are little studies of word gesture keyboard on small mobile devices to reveal its performance on small display.

ZoomBoard

ZoomBoard [29], designed by Carnegie Mellon University & Friendship House, is designed for small mobile devices. ZoomBoard maps a QWERTY soft-keyboard onto the small display. ZoomBoard zooms in on the keyboard area that the user tapped on. On this now enlarged section of the keyboard, the user can tap on a specific button to type the selected character. The keyboard will then zoom out to its original size. Since zoom navigation is common on smartphones, users will find ZoomBoard easy to learn as it does not require users to memorize any kind of special input techniques. This text entry method is designed for small screen devices (with keyboard width=16.0mm, height=6.5mm) and it is shown to be more suitable for small screens rather than medium (width=21.3mm, height=8.6mm) or large (width=28.4mm, height=11.4mm) screen sizes.[18]

Swipeboard

Swipeboard [4], by Autodesk Research, is another software keyboard made for small devices. This text input method, based on swipe and tap combinations, is shown to outperform ZoomBoard after less than 2 hours of training and allows for fast eyes-free text entry. The Swipeboard input method requires 2 steps for a single character: the Swipeboard layout divides a QWERTY keyboard into 9 regions (Figure 7-a), and the first swipe or tap specifies the region of the character (8 swipe directions specifies to their corresponding outer regions and tap specifies to the center); the second swipe is to select the target character from the enlarged region. Because this kind of input method relies on swipe motions more than tap motions, users may need some time getting used to this input technique. Hence, the initial speed of text entry on Swipeboard is slightly slower compared to Zoomboard. However, Autodesk Research claims that users can practice in order to achieve higher typing speeds.

FlickKey

FlickKey [7], a keyboard originally designed for smartphones, consists of a keyboard divided into six sections each containing nine characters. Users can swipe on the button towards a character to type out one of the characters located on the edges of the button, or tap on the button to select the character in the middle of the button. The idea of FlickKey is analogous to that of SwipeKey, but since it was originally designed for smartphones, it contains 9 x 6 = 54 keys on the screen. This number of keys far exceeds the minimum requirements for simple 26 character English text entry. Thus the size of each button is smaller than expected for an optimized keyboard. In addition, our user study shows that a button designed for 8 different directional swipe gestures and 1 tap gesture would cause problems with usability.

MessagEase

MessagEase keyboard [6], by Exideas, has a layout originally designed for smartphones. It uses swipe or tap gestures to increase the effective size of buttons. It also takes Fitts' law into consideration. The most commonly used characters A, N, I, H, O, R, T, E, and S are placed in the center of the nine buttons. Other less commonly used characters are placed around the main characters. Hence, users can tap on one of the nine buttons to type out the most commonly used characters, or swipe on the buttons to select the surrounding, less commonly used characters. MessagEase has problems similar to FlickKey: the size of each button is smaller than expected for an optimized keyboard because there are too many unused input entries on the screen. In addition, the square layout of MessagEase would waste space on the sides if it were to be put on a smartwatch screen with a square shaped watch face.

DESIGN CONSIDERATIONS

In order to create guidelines for our design, we need to define the qualities of a keyboard optimized for smartwatches. We suggest that an optimal smartwatch keyboard must satisfy 3 requirements:

1. Low input error:
   Many studies have shown the trade-off between text entry speed and accuracy [40, 32]. Slow text entry speeds could result from several factors. For example, users may find a keyboard layout confusing and may need time searching for the characters they want to type. High error rates could rise from users typing out characters by accident due to small button sizes and difficult input methods [30]. A high input error rate not only reduces the usability, but also makes the user slow down in an attempt to avoid mistakes and reach an acceptable error rate. Therefore, we need a layout where the users will not have difficulty searching for specific characters. The keyboard input method should also not contain any complicated motions.

2. Pickup usability:
   The speed of text entry is inversely proportional to the reaction time of typing each character. If the user can react quickly then, the overall typing speed will increase. Quick reaction depends on the ease of finding where each character is located and the fluency of the input method. Fluency comes from minimizing how much time it takes for the users to type a single character and then go on to the next character.

3. Usable design:
   Users will find a layout design easy to use if the layout is simple and consistent. Simple designs also reduce the possibility of making typing mistakes.

We have designed SwipeKey based on these requirements. SwipeKey is a type of keyboard that uses swipe direction to create multiple identifications for a single button. The keyboard design is consistent; every button uses the same pattern. We consider only the swipe direction of each button and adhere to vertical and horizontal symmetry for usability and affordance reasons [38, 1]. SwipeKey satisfies our requirements because:

1. It makes the effective button size N times larger, where N is the number of keys for each button. N could range from 2 to more than 10. With large enough value for N, we can make the effective button size large enough for low error input.

2. SwipeKey requires one single stroke for every character.

3. SwipeKey has an intuitive, consistent and simple design for each button.

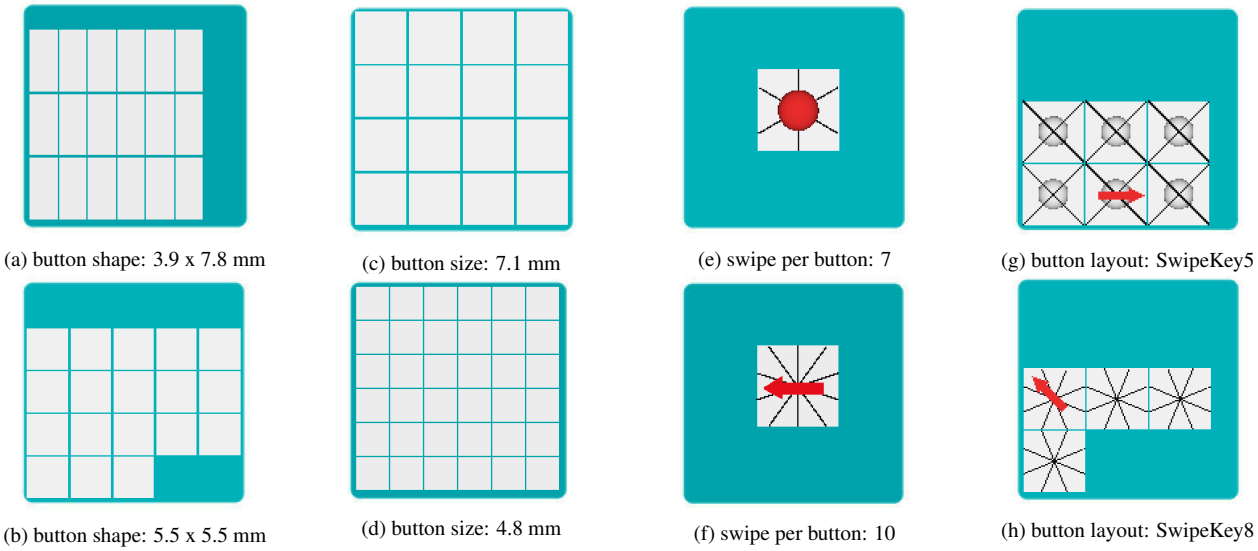
After going through the 3 design considerations, we need to actually start with designing the keyboard layout for SwipeKey. There are 5 parameters we need to address in our design space: button shape, button size, number of swipes per button, button layout, and character arrangement. We then explore the design space through a series of user studies and discussions.

USER STUDY: DESIGNING SwipeKey

We used several tools to accomplish the research. On the software side, we utilized Unity, a game engine framework to create prototypes for our SwipeKey keyboard. On the hardware side, we used an Asus Padfone Infinity mobile phone running Android 4.4. This phone has a 5 inch display. We have decided to use a smartphone for our first prototype instead of an actual smartwatch because prototyping on a smartphone is faster and more convenient.

We attached the phone to the user's non-dominant arm with the screen positioned in landscape orientation, a setup that was also used in Leiva et al.'s study [18]. We rendered a smartwatch screen with a dimension of 29mm x 29mm on the smartphone. Our keyboard measured 25mm x 15mm within this 29mm x 29mm smartwatch screen and it could fit on most commercially available smartwatches.

There are total 4 tasks for each participant in the designing user study. One on shape, one on square size, one on swipe per button, and one on button layout. The experiment order was randomized to counter learning and fatigue artifacts. Figure 2 shows some examples of different keyboard layouts that we tested during our user study. In this test, a red circle or a red arrow was displayed on a single randomly selected button. If a circle on button is highlighted, the user simply needed to



(a) button shape: 3.9 x 7.8 mm

(c) button size: 7.1 mm

(e) swipe per button: 7

(g) button layout: SwipeKey5

(b) button shape: 5.5 x 5.5 mm

(d) button size: 4.8 mm

(f) swipe per button: 10

(h) button layout: SwipeKey8

Figure 2: Sample layouts used while designing SwipeKey. (a),(b) button shape test layout. Each layout contains 18 buttons. (c), (d) button size test layout. (e), (f) swipe per button test layout. Each button is 10 mm per side. (e) is 7 swipe button with a highlighted circle to instruct the user to tap on that button (f) is 10 swipe button with an arrow instructing the user to perform a left swipe. (g), (h) button layout test layout. Each layout is confined in a 15 x 25 mm space.

tap on the corresponding button. On the other hand, if a highlighted arrow appeared, then the user needed to perform a swipe action on that button toward the direction indicted by the arrow. For this swipe action, the finger must press down and start from the button with the highlighted arrow. After initiating the swipe the user could let go of the screen anywhere as long as the finger motion followed the direction of highlighted arrow.

After the user finished tapping or swiping on a button, another button lit up at random. The user gained a point every time he or she taped or swiped on the correct button and towards the correct direction. In case of mistakes, such as missing the highlighted button or not swiping towards the correct direction, the screen would flash red once. Users did not gain or lose any points for misses and they were instructed to try to get as many points as possible within a 60 second time frame. After the timer was up, the screen displayed the number of hits, the number of total taps and swipes, and the percentage of hits over the total number of taps and swipes.

We conducted this user study for designing SwipeKey with 12 users with ages ranging from 19 to 35 (7 male and 5 female).

We analyze the group difference using one-way ANOVA. Then the post hoc test used the Holm-Bonferroni method to adjust for multiple pairwise comparisons.

Button Shape

We decided to make the keyboard buttons square-shaped for 3 reasons, inspired by the consideration of I Scott MacKenzie [24]:

1. Square or rectangular shaped buttons can be tightly packed together in a grid formation without leaving empty gaps between each button (dead space)

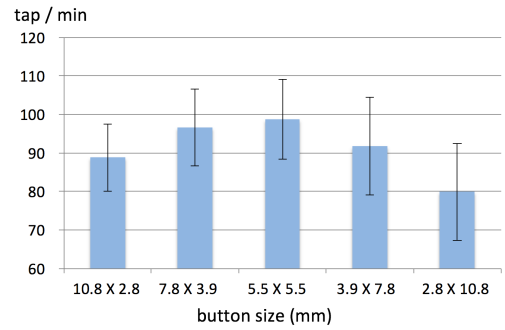
2. Square shaped buttons are better for touch accuracy compared to rectangular shaped buttons

3. Square shaped buttons make all swipe direction angles equal. For example, if the horizontal-to-vertical aspect ratio were 2, then it would be easier to swipe horizontally than vertically. If, on the other hand, the buttons are square, then swiping horizontally or vertically will have equal accuracy.

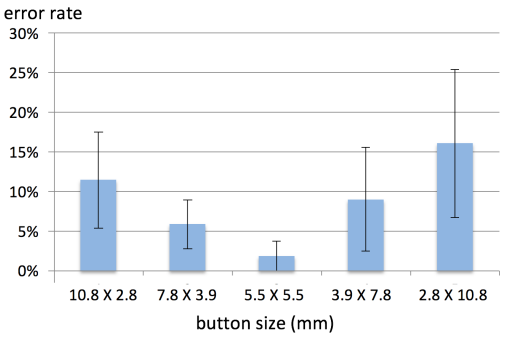
The second statement above is demonstrated in Lee's work [17]. He showed that square-like buttons (wider button) are more preferable than rectangle-like buttons (narrow buttons). To further study the effect of the role of aspect ratio, we have conducted a user study of button shapes on buttons with different aspect ratios but with the same area. Users were asked to tap on a series randomly selected buttons. Figure 2(a) and (b) show the sample test layout.

The shape test consists of 12 users * 5 groups in Figure 3. ANOVA shows significant difference between 5 groups of taps/min ($F(4,55) = 5.51$, $p < .001$, $\eta^2 = 0.29$) and error rate ($F(4,55) = 9.72$, $p < .0001$, $\eta^2 = 0.41$). Post hoc Holm-Bonferroni corrected pairwise comparison shows significant difference in taps/min between 2.8x10.8 button size compared to 5.5x5.5 and 7.8x3.9($p < .01$). Error rate shows significant difference between 5.5x5.5 button size compared to 2.8x10.8, 10.8x2.8 and 3.9x7.8 button sizes.($p < .01$) The result in Figure 3 shows that both error rate and tapping speed will worsen for an aspect ratio different from 1. This suggests that a keyboard design with buttons close to being square shaped is preferred for high speed and low error text entry.

Button Size

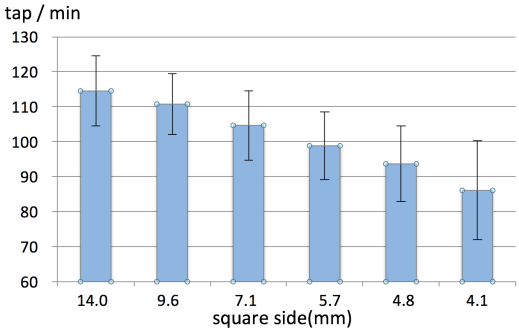


(a)

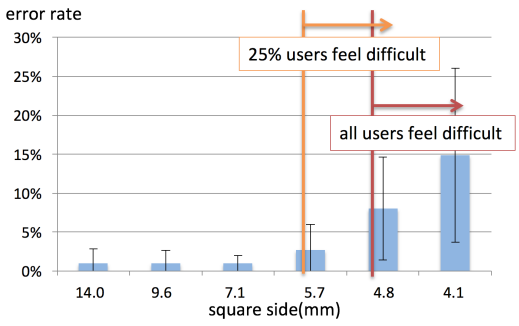


(b)

Figure 3: Button shape test result. (a) Tap speed of 12 users (c) Error rate



(a)



(b)

Figure 4: Result of button size test. (a) Tap speed (b) Error rate

The size of the keyboard and the number of characters are two constrains that connect the button size and swipes per button. Our final goal was to put 26 possible entries (characters in the alphabet) onto the smartwatch keyboard. That means that if we had a keyboard that heavily relied on using swipe motions instead of using taps to select entries on a button, we could make each button bigger. Therefore, we had to simultaneously consider these two correlated parameters: the number of possible swipe directions per button and the button size. Even if they are dependent on each other, we still had to test them individually in separate experiments to further understand the importance of both parameters.

For the experiment involving button sizes, we created keyboards with differently sized buttons. On one hand, having bigger buttons meant that the users had an easier time pressing each of them accurately, but it also meant that we would not be able to fit many buttons on the smartwatch screen. On the other hand, we would be able to place more buttons on the screen if the buttons are small, but it would have it more difficult for users to press each button accurately. During the experiment, users were asked to tap on differently sized square buttons similar to the previous tests. We evaluated them based on how many correct button presses they made. After testing, we asked the user which button size they had trouble tapping on correctly. The results are shown in Figure 4.

The button size test consists of 12 users * 6 groups. ANOVA of Taps/min shows significant difference between groups ($F(5,66) = 12.13$, $p < .0001$,$\eta^2 = 0.48$) and error rate ($F(5,66) = 12.42$, $p < .0001$, $\eta^2 = 0.48$). The pairwise post hoc analysis of error rate separate the results into 2 groups: a group for 14mm to 5.7mm square shaped buttons, and a group with high error rate below 4.8mm. However, the post hoc for the taps/min shows no such 2 distinct groups but a steady drop from larger to smaller buttons. For example, it shows significant difference between 14.0mm square side group and any group square side below 5.7mm($p < .01$). All the users felt that it was difficult to tap on the correct button with a button size smaller than 4.8mm x 4.8mm. About 25% of the users felt that tapping on 5.7mm x 5.7mm buttons was difficult. It suggests making keyboard buttons at least 5.7mm x 5.7mm or larger.

Swipe Directions per Button

For the experiment involving the accuracy of swipe motions, a large button is divided into different angle sections. An arrow lit up at random and the user was asked to swipe towards the direction of the arrow. Different button designs may allow different amounts of swipe directions. For example, if a button design has 4 swipe directions, it means that the arrow can end up pointing at 4 possible directions: up, down, left and right. Figure 2 (f) shows a button with 10 swipe directions, meaning the arrow can point towards any of these 10 directions (mixture of left, right and diagonal swipes). Given a button with a small number of swipe directions, performing the correct swipe is easier compared to buttons with a large number of swipe directions. This is because users need to be more precise with the angle of the swipe if they need to get the direction right within a small angle of tolerance. Clearly,

swipe number each button	2		3			4		5		6			7		8		9	
minimum button for 26 entries	14		9			7		6		5			4		4		3	
possible layout (horizontal x vertical button number)	4x4	5x3	3x3	4x3	5x2	3x3	4x2	3x2	4x2	3x2	4x2	5x1	2x2	3x2	2x2	4x1	2x2	3x1
button size per side (mm)	3.75	5	5	5	5	5	6.25	7.5	6.25	7.5	6.25	5	7.5	7.5	7.5	6.25	7.5	8.33
potential possible entry	32	30	27	36	30	36	32	30	40	36	48	30	28	42	32	32	36	27
selected for user study		o		o			o	o		o				o	o			o
keyboard layout is constrained in 25 x 15 mm space																		

Table 1: Layout decisions for different swipe number variations of SwipeKey. We select the layout with the largest square button size of a specific swipe number. If sizes are the same between many layouts, we select the layout with the maximum number of potential entries.

we do not want to have too many swipe directions per button, as the keyboard would become difficult to use.

Users were asked to swipe towards a randomly chosen direction on a 10mm x 10mm button. Since we consider both vertical and horizontal symmetry, only even swipe numbers are taken into account. There are many works that combine tap with swipe for text entry purposes. Therefore, we also consider every even swipe with an additional tap as a candidate SwipeKey button. After this test, we asked the user to report those buttons with swipe direction that are hard to follow correctly. Note that the odd number swipe represents its lower even number swipe plus one tap. The sample layout is shown in Figure 2 (e), (f). The results of the experiment are shown in Figure 5.

ANOVA of swipes/min shows no significant difference between groups ($F(9,110) = 0.35$, $p = 0.96$, $\eta^2 = 0.08$) but the error rate shows a significant difference ($F(9,110) = 6.02$, $p < .0001$, $\eta^2 = 0.33$). The pairwise post hoc shows a 2 group error: swipe per button 3,9,10,11 as one group and another group with lower error. The average swipe speed is about 90 swipes per minute and almost independent of swipes per button. But the error rate rises when the number of swipe directions per button is larger than 8. Notice that even though the error rate is low for 6 swipes, 17% of the users felt that it was difficult to swipe correctly with this number of swipe directions. Moreover, the layout with tap and swipe has a lower accuracy than the group with swipe only. It suggests that the combination of 2 different types of movements (tap and swipe) might increase the possibility of making errors. This inconsistency between tap and swipe is also reported by 25% of users.

Button Layout

Based on the keyboard space and a minimum of 26 entries, we can arrange many possible layouts for each swipe number variation of SwipeKey. Each swipe number of SwipeKey has one layout with maximum button size. The layout choices of different swipe number variations of SwipeKey are shown in Table 1.

To investigate the characteristics of different swipe number layouts. Users were asked to swipe or tap at randomly selected buttons and directions from 26 possible entries of 8 different types of SwipeKey layouts. After that, users were

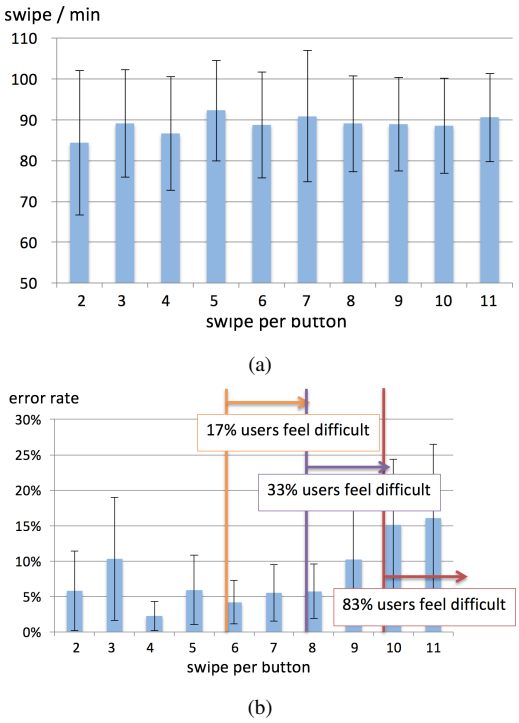


Figure 5: Result of swipe per button test. (a) Swipe speed (b) Error rate

asked to answer a questionnaire about the difficulty rating of each layout and voted for a preferred layout. The results are shown in Figure 6.

Swipes/min shows significant difference ($F(7,88) = 4.8$, $p = .0001$, $\eta^2 = 0.28$) and error rate also shows a significant difference ($F(7,88) = 5.41$, $p < .0001$, $\eta^2 = 0.30$). Post hoc shows two groups: layout 2 and 3 as a group with lower swipes/min and higher error rate compared with another group. The small button size in SwipeKey 2 and SwipeKey 3 layouts were the worst in terms of input speed and error rate. This is consistent with our button size user study: the error rate increases drastically if the button size is smaller than 5.7mm x 5.7mm. The large button size group also suffers a slight speed drop and error rate increase due to many possible swipe directions for each button. Although the SwipeKey 4 and 5 layouts are the best for error rate and speed, these layouts are almost identical to the SwipeKey 6 and 7 layouts in terms of both error rate and speed. However, user ratings paint a clearer picture. Figure 6 (c) shows that SwipeKey 4 and 5 layouts outperform other candidates in difficulty rating and preference. Therefore, we have selected SwipeKey 4 and 5 layouts to represent the official SwipeKey layouts for our smartwatch text entry solution.

Character Arrangement

The last piece of the puzzle in designing SwipeKey is figuring out how to arrange the 26 characters in the alphabet. We considered an alphabetically ordered character layout because it is a common keyboard character layout [22]. Secondly, there

is evidence that experts typing on a keyboard with alphabetical layout perform just as well as experts typing on a QWERTY layout keyboard [22, 27]. Thirdly, SwipeKey is a different type of keyboard. We did not have a proven way of transforming common keyboard layouts, such as QWERTY and Dvorak, and applying them to SwipeKey.

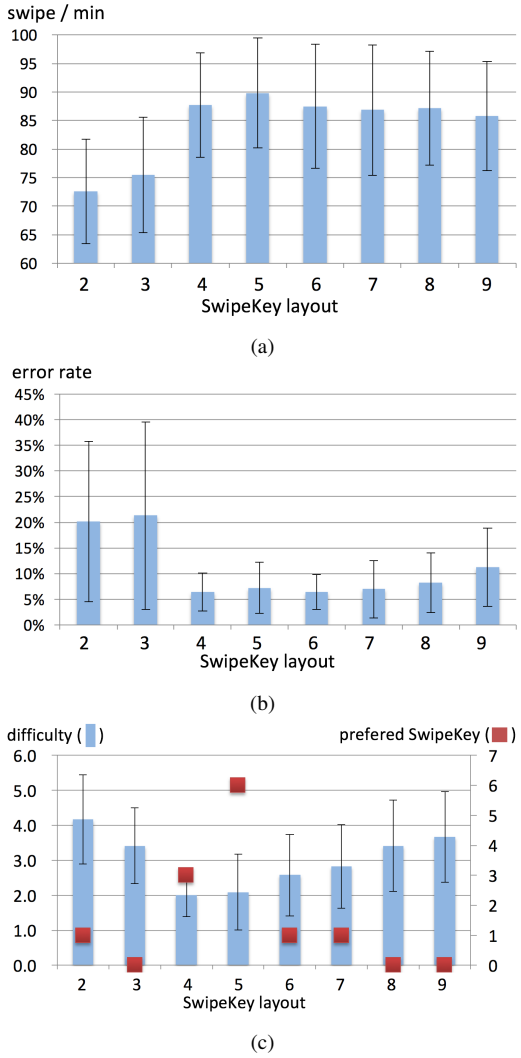


Figure 6: Result of button layout test. (a) Test speed (b) Test error rate (c) Button layout test difficulty rating (blue bar) and participants preferred button layout (red dot)

USER STUDY: SwipeKey PROTOTYPE IMPLEMENTED ON AN ANDROID SMARTWATCH

To test the performance of SwipeKey, we implemented SwipeKey 4 and SwipeKey 5 on a Sony SmartWatch 3 running Android 4.4. We have also implemented Autodesk Research's Swipeboard on our system for the purpose of comparison since it is also an swipe-based keyboard and performs well on smartwatches. Swipeboard's performance is compatible or better than that of Zoomboard [4]. However, Swipeboard has a QWERTY based keyboard layout. In order to separate the factor of keyboard layout from character arrangement, we also implemented a Swipeboard-like keyboard using an alphabetical keyboard layout. To make the keyboard consistent, we removed the button with 4 characters: r, t, y, u and allowed buttons to have only 3 characters each. For all the keyboards we implemented, users could delete characters (backspace) or enter a space (spacebar) in a consistent way. To delete a character, users could swipe left on the text entry space. To enter a space, the users could swipe right on the text entry space. Figure 7 shows all the keyboard designs.

Participants

We have recruited 12 participants (7 male, 5 female) for this test. The average age was 24 years. 9 participants stated that they could perform blind-typing on a PC keyboard. Every participant were ask to wear the smartwatch on their dominate hand throughout tasks and 10 of them were right-handed. All participants own touchscreen smartphones, 5 of them have used a smartwatch before, and 10 of them have used traditional ITU physical keyboards on mobile phones.

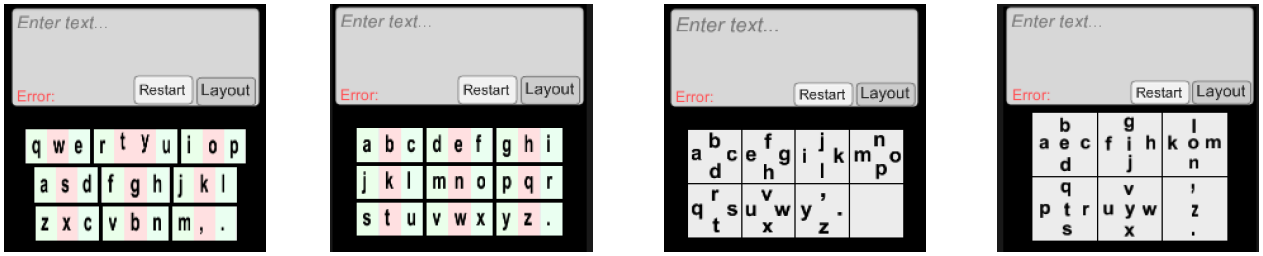
Task

Participants were given a total of 2 minutes to get familiar with each type of keyboard before the test. They were told to use their index finger of their dominant hand to enter text. Each participant had to run 6 task blocks per keyboard, where the users were asked to type out 5 phrases per task block. In other words, they were asked to type out 30 phrases for each keyboard layout in a predetermined counterbalanced order. All in all, each user saw a total of 120 different phrases throughout the test. These test phrases were drawn randomly from the 500-phrase corpus compiled by MacKenzie and Soukoreff [23]. A big screen in front of the user was used to display each phrase and the user then entered the displayed text on the smartwatch keyboard. The screen displayed the current phrase the user was asked to type. The screen switched to showing the next phrase when the participant finished typing out the current phrase. The reason for displaying each phrase on the screen rather than asking users to memorize each phrase was to avoid memorability bias. At the end of the test, participants completed a short questionnaire and a short interview.

Result and Discussion

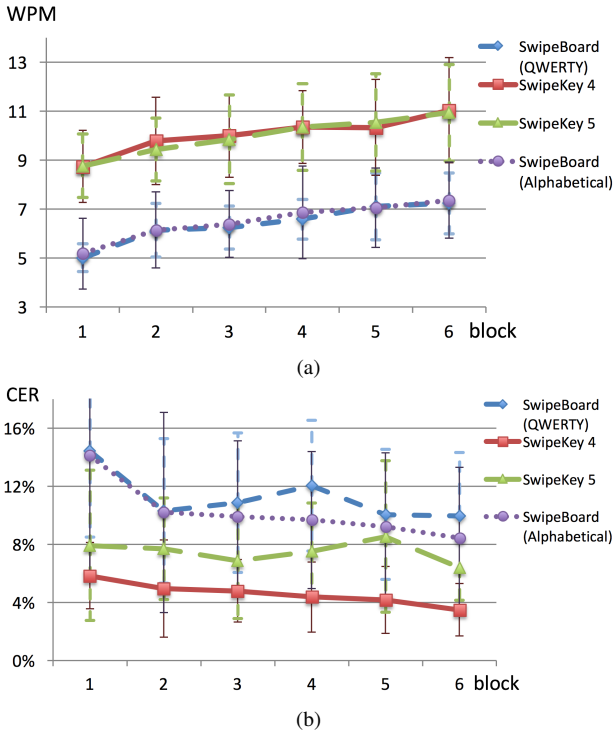
Text Entry Speed

Figure 8 (a) shows the aggregated WPM results of the 12 users (including error correction time). Swipeboard (QWERTY layout) had 5.0 WPM in the first block. After more than half an hour of training, the typing speed increased to 7.2 WPM in the last block. The WPM curve of Swipeboard (QWERTY) is slightly lower and almost overlaps with that of Swipeboard (alphabetical), with a speed of 5.2 WPM in the first block and 7.3 WPM in the end. The large Swipeboard WPM gap between original paper [4] and our experiment comes from different task settings. The users in original paper were asked to fully memorize and type each word at a time. They count only the time between the "start" and "end" of each word. However, we asked users to type a phrase and start counting when user seeing the phrase. The comparison



(a) (b) (c) (d)

Figure 7: Keyboard layout on smartwatch (a) Swipeboard (b) Swipeboard (alphabetical) (c) SwipeKey4 (d) SwipeKey5



(a)

(b)

Figure 8: Quantitative results of user study for SwipeKey (a) WPM (b) CER

of 2 Swipeboard suggest that the QWERTY keyboard layout may not help users type faster than users who use an alphabetical keyboard layout. It also supports our design decision of using alphabetical grouping. Two of our users mentioned that the feeling of typing on Swipeboard (QWERTY) is totally different from typing on a mechanical keyboard, despite the fact that both users are capable of blind-typing. It is hard for them to transfer their muscle memory of using a mechanical QWERTY layout to the Swipeboard (QWERTY) input method. This might be a reason why QWERTY is not particularly helpful for character location memorization.

The WPM of SwipeKey 4 and SwipeKey 5 are 8.7 and 8.8 respectively in the first block, and 11 and 10.9 respectively in the last block. SwipeKey 4 and SwipeKey 5 are comparable in terms of speed even after users had time to practice. The WPM of SwipeKey group is much higher than that of the Swipeboard group. There is a significant performance difference among all the test data ($F(3,44) = 24.7$, $p < .0001$, $\eta^2 = 0.62$). Holm-Bonferroni corrected pairwise post hoc reveals a 2 group WPM.(SwipeKey group and Swipeboard group, all pair between groups $p < .01$). The initial speed of the SwipeKey group is even faster than that of the trained Swipeboard group. It shows that the speed gap between SwipeKey and Swipeboard is at about 4 WPM across all tests. This might imply a non-converging typing speed between these 2 groups even after long-term training. The superiority of SwipeKey in speed is due to the second design requirement. SwipeKey is expected to have a shorter reaction time for each character. It takes 1 swipe for each character while Swipeboard requires 2.

Error Rate

We used the character error rate (CER) to measure error rate. In Figure 8 (b), the CER in the beginning is comparable between Swipeboard (QWERTY) and Swipeboard (alphabetical). Though, after a short training session, Swipeboard (alphabetical) shows a slightly lower CER in the last 3 blocks. Six users reported that they made more mistakes when they tried to type out the 't' and 'y' characters located on the 'rtyu' button of the Swipeboard (QWERTY). One user found the 'rtyu' button inconsistent and confusing because that button consisted of 4 characters while the other 8 buttons in the layout consisted of only 3.

The combined data from each 2 conditions of SwipeKey and Swipeboard shows that the CER for the SwipeKey group (5.8%) is about half of the error rate for the Swipeboard group (10.9%) across all tests ($F(1,46) = 19.0$, $p < .0001$, $\eta^2 = 0.29$). This could be explained by 2 factors. Firstly, Swipeboard users needed to make 2 swipes in order to type a single character, which increased the error rate due to the extra swipe. SwipeKey users needed to make only 1 swipe per character. Secondly, the inconsistent swipe directions in the first (8 directional swipes + 1 tap) and the second layer (2 directional swipes + 1 tap or 4 directional swipes) introduced a higher error rate. Users also reported that SwipeKey is more intuitive to use.

For in-group comparison of SwipeKey, SwipeKey 4 had a significantly lower CER (4.4% on average, 3.4% for the last block) compared to SwipeKey 5 (7.4% on average, 6.3% for the last block) ($F(1,22) = 8.6$, $p < .001$, $\eta^2 = 0.28$). We already noticed this phenomenon in our previous user study

when designing SwipeKey, and we believe that it is caused by inconsistent design and combination of 2 different types of movement: tap and swipe. Some participants mentioned that their tap was accepted as a swipe and that they could not distinguish which is more likely to happen.

User Survey

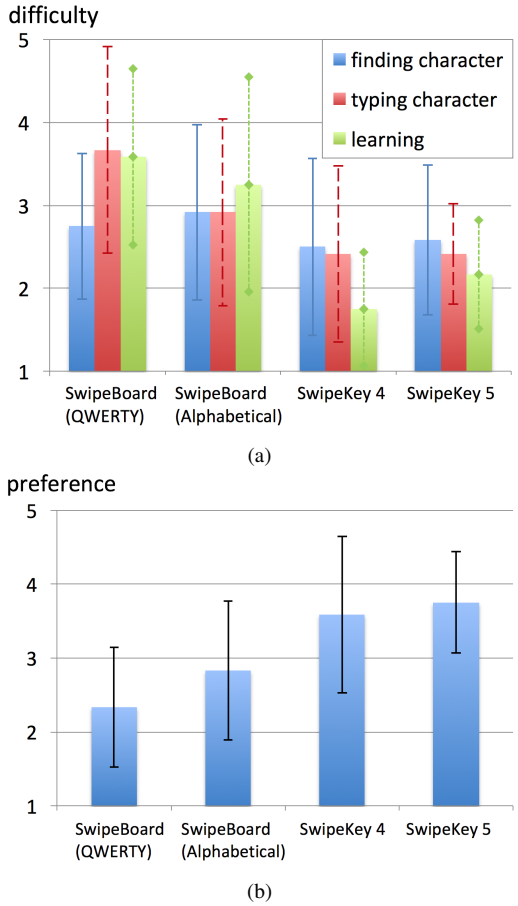


Figure 9: (a) User rating of difficulty on a 5-point Likert scale for finding a character, typing a located character, and learning how to use different keyboards. (b) User preference of different keyboards

We summarize 3 different kinds of difficulty rating of keyboard layouts in Figure 9 (a). There is no significant difference in finding a character between Swipeboard (QWERTY) and Swipeboard (alphabetical). However, there are 3 users who said that they found it more difficult to find characters on the alphabetically ordered keyboard. Their difficulty rating for finding characters on the alphabetically ordered keyboard is much higher than that of other users (average 3.9 point of the 3 users compared with 2.7 point of 12 users). The rest of the users did not find the QWERTY layout more helpful.

Typing characters on Swipeboard (QWERTY) is the most difficult. This is revealed in our interview with users. Our users mentioned that "It is hard to swipe 'r' and 't', and it continuously enters the wrong character", "Sometimes, my swipe on the first layer keeps entering the wrong group of characters", or "The 't' and 'y' are hard to type". Considering the difficulty of learning for each keyboard, we see that SwipeKey is easier to learn compared to Swipeboard ($F(1,46) = 19.7$, $p < .0001$, $\eta^2 = 0.30$). One of the participants said that it took him a while to get accustomed to Swipeboard, but he could use SwipeKey immediately. However, those who had difficulty in finding characters do feel that there is a high learning curve for SwipeKey(average 3.5 compared with total average 2.0).

In Figure 9 (b), the preference for SwipeKey in a 5-point Likert scale is higher than that of Swipeboard ($F(1,46) = 15.0$, $p < .001$, $\eta^2 = 0.25$). Most of the users said that they prefer SwipeKey because it is more intuitive and easier to use in comparison with Swipeboard. However, there is one participant that prefers Swipeboard (QWERTY) due to the QWERTY layout, despite the fact that both speed and error rate of that particular participant are better with SwipeKey.

LIMITATION AND CONCLUSION

In this work, we have focused on creating a 1 swipe or 1 tap keyboard for smartwatches. We have first discussed the 3 qualities of optimized keyboards. These requirements outline the design for SwipeKey, and allowed us to conduct a series of user studies to narrow down the number of possible design options. We have implemented SwipeKey 4 and SwipeKey 5, both optimized for WPM, error rate, rating of difficulty and user preference. The result of SwipeKey shows a 55% improvement in WPM and a 44% decrease in error rate compared to Swipeboard. We have discovered that the QWERTY keyboard layout does not provide significant advantages for small swipe-based keyboards according our experiment. This knowledge could be helpful for future keyboard designs on small devices since keyboard designers are not forced to distort their keyboard designs to accommodate the QWERTY layout.

There is an important design principle that emerges from our user study. A consistent design is crucial for intuitive, easy to use, and low error rate keyboard layouts.

We have focused our work on keyboard design for smartwatches. As suggested in the work of Leiva et al. [18], designing a suitable keyboard depends on the screen size. For a smaller keyboard size, we may find that an even higher swipe direction number SwipeKey such as SwipeKey 6 or even SwipeKey 9 may outperform SwipeKey 4 or 5. This is because as the keyboard space becomes smaller, the button size will shrink accordingly, and eventually the size of a button would become smaller than the acceptable minimum size, leading to a drastic increase in input errors. Note that the error rate increases more slowly when increasing the number of swipe directions compared to shrinking the button size below a certain size limit.

Modern soft keyboards use word prediction and autocorrection to improve performance. These are indeed beneficial to any kind of input method. However, in this paper we focus on increased WPM and reduced error rates through keyboard design, and automatic error correction could very well obscure deficiencies in our keyboard design. With a proven

keyboard design as a baseline, we can now look into word prediction and auto-correction in future works.

We believe that our work provides a well-designed keyboard for smartwatches that enables intuitive, fast, and low error text entry. Smartwatch manufacturers can implement SwipeKey on smartwatches, along with other input methods, such as voice input, to provide more choices for the users. Furthermore, this work provides a new understanding of swipe resolution and character arrangement on small devices. This information could be helpful for future researchers.

REFERENCES

1. 2001. In *Interactive Systems: Design, Specification, and Verification*, Chris Johnson (Ed.). Lecture Notes in Computer Science, Vol. 2220.
2. Christoph Amma, Dirk Gehrig, and Tanja Schultz. 2010. Airwriting Recognition Using Wearable Motion Sensors. In *Proceedings of the 1st Augmented Human International Conference (AH '10)*. Article 10, 8 pages.
3. Apple. 2015. Apple Watch. (2015). Official Site http://www.apple.com/watch/.
4. Xiang 'Anthony' Chen, Tovi Grossman, and George Fitzmaurice. 2014. Swipeboard: A Text Entry Technique for Ultra-small Interfaces That Supports Novice to Expert Transitions. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. 615–620.
5. Mark D Dunlop, Andreas Komninos, and Naveen Durga. 2014. Towards high quality text entry on smartwatches. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2365–2370.
6. Exideas. 2015. MessageEase. (2015). Official Site https://www.exideas.com/ME/index.php.
7. FlickKey. 2010-2014. FlickKey Keyboard. (2010-2014). Official Site http://www.flickkey.com.
8. Markus Funk, Alireza Sahami, Niels Henze, and Albrecht Schmidt. 2014. Using a Touch-sensitive Wristband for Text Entry on Smart Watches. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems (CHI EA '14)*. 2305–2310.
9. Mikael Goldstein, Robert Book, Gunilla Alsiö, and Silvia Tessa. 1999. Non-keyboard QWERTY touch typing: a portable input interface for the mobile user. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 32–39.
10. Nathan Green, Jan Kruger, Chirag Faldu, and Robert St Amant. 2004. A reduced QWERTY keyboard for mobile text entry. In *CHI'04 extended abstracts on Human factors in computing systems*. ACM, 1429–1432.
11. Jonggi Hong, Seongkook Heo, Poika Isokoski, and Geehyuk Lee. 2015. SplitBoard: A Simple Split Soft Keyboard for Wristwatch-sized Touch Screens. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. 1233–1236.
12. Huawei. 2015. Huawei Watch. (2015). Official Site http://consumer.huawei.com/minisite/worldwide/huawei-watch/.
13. Christina L. James and Kelly M. Reischel. 2001. Text Input for Mobile Devices: Comparing Model Prediction to Actual Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01)*. 365–371.
14. Jiepu Jiang, Wei Jeng, and Daqing He. 2013. How do users respond to voice input errors?: lexical and phonetic query reformulation in voice search. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 143–152.
15. Jeong Ho Kim, Lovenoor Aulck, Ornwipa Thamsuwan, Michael C Bartha, and Peter W Johnson. 2014. The effect of key size of touch screen virtual keyboards on productivity, usability, and typing biomechanics. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 56, 7 (2014), 1235–1248.
16. Per-Ola Kristensson and Shumin Zhai. 2004. SHARK 2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*. ACM, 43–52.
17. Seungyon Lee and Shumin Zhai. 2009. The Performance of Touch Screen Soft Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. 309–318.
18. Luis A. Leiva, Alireza Sahami, Alejandro Catala, Niels Henze, and Albrecht Schmidt. 2015. Text Entry on Tiny QWERTY Soft Keyboards. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. 669–678.
19. LG. 2015. LG WATCH Urban. (2015). Official Site http://www.lg.com/us/smartwatch/urbane.
20. Frank Chun Yat Li, Richard T. Guy, Koji Yatani, and Khai N. Truong. 2011. The 1Line Keyboard: A QWERTY Layout in a Single Line. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. 461–470.
21. Kent Lyons, Thad Starner, Daniel Plaisted, James Fusia, Amanda Lyons, Aaron Drew, and E. W. Looney. 2004. Twiddler Typing: One-handed Chording Text Entry for Mobile Phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. 671–678.
22. I. Scott MacKenzie and R. William Soukoreff. 2002. Text Entry for Mobile Computing: Models and Methods,Theory and Practice. *HumanComputer Interaction* 17, 2-3 (2002), 147–198. DOI: http://dx.doi.org/10.1080/07370024.2002.9667313

23. I. Scott MacKenzie and R. William Soukoreff. 2003. Phrase Sets for Evaluating Text Entry Techniques. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03)*. 754–755.

24. I. Scott MacKenzie and Shawn X. Zhang. 1999. The Design and Evaluation of a High-performance Soft Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*. 25–31.

25. Anders Markussen, Mikkel Rønne Jakobsen, and Kasper Hornbæk. Vulture: A Mid-air Word-gesture Keyboard.

26. Motorola. 2015. Moto 360. (2015). Official Site https://www.motorola.com/us/products/moto-360.

27. Donald A Norman and Diane Fisher. 1982. Why alphabetic keyboards are not easy to use: Keyboard layout doesn't much matter. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 24, 5 (1982), 509–519.

28. Nuance. 2016. T9. (2016). http://www.nuance.com/for-business/by-product/t9/index.htm

29. Stephen Oney, Chris Harrison, Amy Ogan, and Jason Wiese. 2013. ZoomBoard: A Diminutive Qwerty Soft Keyboard Using Iterative Zooming for Ultra-small Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. 2799–2802.

30. Pekka Parhi, Amy K. Karlson, and Benjamin B. Bederson. 2006. Target Size Study for One-handed Thumb Use on Small Touchscreen Devices. In *Proceedings of the 8th Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '06)*. 203–210.

31. Kurt Partridge, Saurav Chatterjee, Vibha Sazawal, Gaetano Borriello, and Roy Want. 2002. TiltType: Accelerometer-supported Text Entry for Very Small Devices. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST '02)*. ACM, New York, NY, USA, 201–204. DOI:http://dx.doi.org/10.1145/571985.572013

32. Rjean Plamondon and Adel M. Alimi. 1997. Speed/accuracy trade-offs in target-directed movements. *Behavioral and Brain Sciences* 20 (6 1997), 279–303. Issue 02. DOI:http://dx.doi.org/null

33. S. Primorac and M. Russo. 2012. Android application for sending SMS messages with speech recognition interface. In *MIPRO, 2012 Proceedings of the 35th International Convention*. 1763–1767.

34. Reza Rawassizadeh, Blaine A Price, and Marian Petre. 2015. Wearables: Has the age of smartwatches finally arrived? *Commun. ACM* 58, 1 (2015), 45–47.

35. Samsung. 2015. Gear S. (2015). Official Site http://www.samsung.com/tw/consumer/mobile-phones/galaxy-gear/galaxy-gear/SM-R7500ZWABRI.

36. Andrew Sears, Doreen Revis, Janet Swatski, Rob Crittenden, and Ben Shneiderman. 1993. Investigating touchscreen typing: the effect of keyboard size on typing speed. *Behaviour & Information Technology* 12, 1 (1993), 17–22.

37. Sony. 2015. SmartWatch 3 SWR50. (2015). Official Site http://www.sonymobile.com/global-en/products/smartwear/smartwatch-3-swr50/.

38. Harold Thimbleby. 2002. Symmetry for Successful Interactive Systems. In *Proceedings of the SIGCHI-NZ Symposium on Computer-Human Interaction (CHINZ '02)*. 1–9.

39. Keith Vertanen and Per Ola Kristensson. 2009. Parakeet: A Continuous Speech Recognition System for Mobile Touch-screen Devices. In *Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI '09)*. 237–246.

40. Shumin Zhai, Jing Kong, and Xiangshi Ren. 2004. Speed–accuracy tradeoff in Fitts law taskson the equivalency of actual and nominal pointing precision. *International journal of human-computer studies* 61, 6 (2004), 823–856.

41. Shumin Zhai and Per Ola Kristensson. 2012. The word-gesture keyboard: reimagining keyboard interaction. *Commun. ACM* 55, 9 (2012), 91–101.

42. Shumin Zhai, Per Ola Kristensson, Pengjun Gong, Michael Greiner, Shilei Allen Peng, Liang Mico Liu, and Anthony Dunnigan. 2009. Shapewriter on the iphone: from the laboratory to the real world. In *CHI'09 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2667–2670.