i'sFree: Eyes-Free Gesture Typing via a Touch-Enabled Remote Control

Suwen Zhu
Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
suwzhu@cs.stonybrook.edu

Jingjie Zheng
Google
Kitchener, Ontario, Canada
jingjie@acm.org

Shumin Zhai
Google
Mountain View, California, USA
zhai@acm.org

Xiaojun Bi
Department of Computer Science
Stony Brook University
Stony Brook, New York, USA
xiaojun@cs.stonybrook.edu

ABSTRACT

Entering text without having to pay attention to the keyboard is compelling but challenging due to the lack of visual guidance. We propose *i'sFree* to enable eyes-free gesture typing on a distant display from a touch-enabled remote control. i'sFree does not display the keyboard or gesture trace but decodes gestures drawn on the remote control into text according to an invisible and shifting Qwerty layout. i'sFree decodes gestures similar to a general gesture typing decoder, but learns from the instantaneous and historical input gestures to dynamically adjust the keyboard location. We designed it based on the understanding of how users perform eyes-free gesture typing. Our evaluation shows eyes-free gesture typing is feasible: reducing visual guidance on the distant display hardly affects the typing speed. Results also show that the i'sFree gesture decoding algorithm is effective, enabling an input speed of 23 WPM, 46% faster than the baseline eyes-free condition built on a general gesture decoder. Finally, i'sFree is easy to learn: participants reached 22 WPM in the first ten minutes, even though 40% of them were first-time gesture typing users.

CHI 2019, May 4–9, 2019, Glasgow, Scotland UK

ACM ISBN 978-1-4503-5970-2/19/05…$15.00
https://doi.org/10.1145/3290605.3300678

CCS CONCEPTS

• **Human-centered computing** → **Interaction devices**; **Interaction techniques**.

KEYWORDS

Text entry; Touchscreen; Gesture typing; Eyes-free text entry

ACM Reference Format:
Suwen Zhu, Jingjie Zheng, Shumin Zhai, and Xiaojun Bi. 2019. i'sFree: Eyes-Free Gesture Typing via a Touch-Enabled Remote Control. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland UK*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300678

1 INTRODUCTION

Eyes-free text entry – entering text without having to pay attention to the keyboard – is appealing for many reasons. On

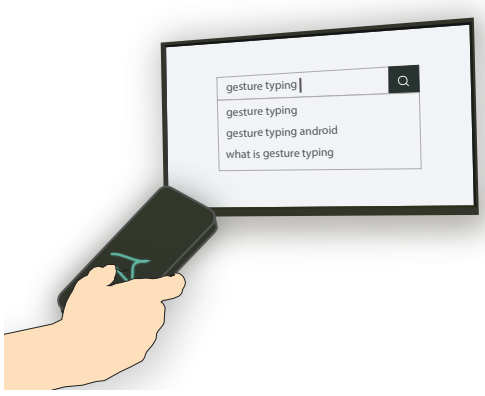


Figure 1: i'sFree enables eyes-free gesture typing on devices like a smart TV using a touch-enabled remote control. The output display does not visualize the keyboard layout, nor the gesture trace. The blue stroke on the remote control is for illustrating the finger motion and is invisible in real use.

touchscreen *direct* input devices, it frees a user's attention from the keyboard, saves screen real estate, and with an optimized decoder, it may even lead to better performance than the typical "eyes-on" input [13, 28, 34].

Eyes-free input can be more valuable when interacting with a distant display (e.g., a TV) via a remote control. In this setting, the input and output devices are decoupled, meaning that users need to regularly switch attention between where they type and where the output is. The keyboard present on the distant display also discourages users from transitioning to eyes-free input, where less attention switch may lead to more focused interaction and better performance.

Taking advantage of the popular gesture typing input method [10], we investigated how to support eyes-free gesture typing on a touch-enabled remote control. Why gesture typing? First, it has certain advantages that are well-suited for eyes-free input. It allows users to express intended words by approximating shape finger strokes, rather than precise taps on the corresponding keys which may be difficult without key visuals. Second, the pictorial effect [18] shows that humans are better at memorizing shapes than sequences of letters. Expert users may memorize common word shapes and execute them efficiently in an eyes-free manner. Third, gesture typing is favored by some users over the traditional tap typing on touch-enabled devices. It would benefit those users greatly if it can be supported on a remote control.

Although eyes-free gesture typing seems appealing, supporting it is not easy. It is challenging for the system to decode the gesture input without knowing the keyboard location. The input signals could be very noisy without the visual guidance of keys.

To address these challenges, we designed and implemented i'sFree, a text entry system that supports eyes-free gesture typing on a touch-enabled remote control. The main difference from a general gesture typing system [10] is that it dynamically learns the keyboard position based on the geometry of current and historical input gestures, and uses a Monte Carlo method to account for the keyboard location uncertainty in decoding. It is designed based on the understanding of how users perform eyes-free gesture typing.

Our user study showed promising results. First, eyes-free gesture typing was feasible: reducing the visual guidance on the distant display hardly affected the input speed. Second, the i'sFree gesture decoding algorithm was very effective, enabling an input speed of 23 WPM, 46% faster than the baseline eyes-free condition built on a general gesture decoder. Third, i'sFree is also easy to learn: participants reached 22 WPM in the first ten minutes, even though 40% of them were first-time gesture typing users.

Eyes-free typing on a physical keyboard (a.k.a., touch typing) has proven feasible and efficient. Likewise, we expect i'sFree will serve as an efficient text entry method on a remote touchpad, broadening the communication channel between users and many computing devices.

2 RELATED WORK

Our work relates to eyes-free text entry, gesture typing technique, and imaginary UI research.

Eyes-Free Text Entry

Eyes-free text entry has long been desired since the invention of the keyboard/typewriter. Touch typing – typing without looking at the keyboard – is considered superior to "hunt-and-peck", because the typists can focus the visual attention on the composition of the text. It has been the standard typing method taught since at least the 1920's [9] and was used for typing on a Twiddler one-handed keyboard and a mini-Qwerty physical keyboard [4].

As touchscreen devices are becoming increasingly popular, a considerable amount of research has been conducted to support eyes-free text entry on touchscreens, including tap typing on an invisible virtual keyboard with one or two fingers [24, 34], tap typing on a touchpad with a thumb [13], ten-finger typing on a flat touchscreen [5], and ten-finger typing on the back of a tablet [20]. These research showed that eyes-free text entry was feasible because many users could well-remember the Qwerty layout.

In addition to tap typing, some research investigated gesture-based text entry on touchscreen, such as an eyes-free Graffiti text input system [23, 25] and a swipe-based invisible text entry method with numberpad-like layout on smartwatches [17].

Another option for eyes-free text entry is speech input, which has advanced greatly since the rise of deep learning technique. However, speech input suffers from some inherent limitations: it may raise privacy concerns; it is socially inappropriate in quiet environments; the recognition accuracy may suffer in noisy environments (e.g., parties, playing video games); it is also unfeasible for speech-impaired users. Thus, speech input is usually complemented by other methods.

Complementary to the existing eyes-free input methods, we focus on inputting on a full-size Qwerty keyboard via a touch-enabled remote control. It is well suited for scenarios where the displays are decoupled from the input region, e.g., TVs, game consoles, head-mounted displays in virtual or augmented reality.

Gesture Typing

First proposed by Zhai and Kristensson [10, 31–33], gesture typing has gained broad adoption across the world, and has been extended to a variety of input modalities. Bi et al. [3] created a bimanual gesture keyboard, which allowed one word to be entered by multiple strokes using both hands.

Markussen et al. [16] investigated gesture typing in mid-air. Their text entry system, Vulture, projects users' hand movement onto the display, and uses pinch as a word delimiter. Their studies showed that users could achieve 20.6 WPM after a 10-session study, and could reach 28.1 WPM if trained on a few phrases. Additionally, Yu et al. [30] explored using head movement to perform gesture typing, while Yeo et al. [27] investigated using device tilt angle for gesture typing. Building on the existing gesture typing technique, our research explores how to support it in the eyes-free mode via a remote control, which had not been explored.

Imaginary UIs

Research on imaginary interfaces showed that users can perform spatial interaction without visual feedback, providing empirical evidence on the feasibility of eyes-free text input.

Virtual Shelves [11] and Yan et al. [26] showed that users could effectively select targets in a body-centric virtual space or a VR environment with their kinesthetic and spatial memory. Gustafson et al. [6] discovered that users were able to draw single stroke characters and simple sketches in a user-defined imaginary space with a recognition rate of 94.5%. Imaginary Phone [7] showed that users could build up spatial memory automatically by interacting with a physical device, and transfer the spatial knowledge from physical to imaginary interfaces. Lin et al. [12] also showed that users can accurately distinguish targets on their forearms while no visual references were provided.

As shown in literature, interacting with computers with little or no visual feedback is always appealing. However, no research has investigated how to support it for gesture typing on a touchpad, which is the main research question we aimed to address in this work.

3 EXPERIMENT 1: UNDERSTANDING EYES-FREE GESTURE TYPING

To support eyes-free gesture typing on a touch-enabled remote control, we first conducted a user study to investigate users' typing behavior.

Experiment Setup

Design and Tasks. We designed a Wizard-of-Oz experiment to collect users' unbiased eyes-free gesture typing behaviors. Participants were instructed to transcribe phrases displayed on a 46-inch smart TV via a touchscreen phone that functioned as a touchpad. To reflect the eyes-free typing condition, no keyboard or gesture trace was displayed on the TV or the touchpad. Participants were asked to gesture type as naturally as possible, and assume that the underlying text entry system could correctly decode the input.

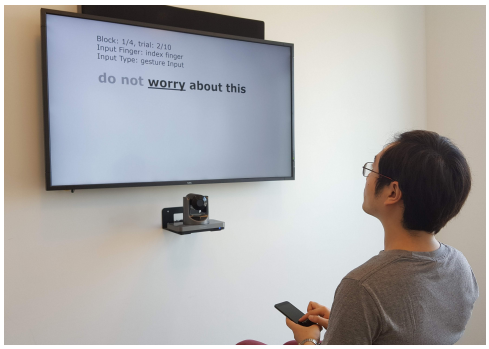


Figure 2: The setup of Experiment 1. A user was seated in front of a TV and performing eyes-free gesture typing.

Participants and Apparatus. We recruited 12 participants (5 female, all right-handed), aged from 24 to 34 ($M = 27.8$). Their median familiarity with Qwerty layout was 4.5 (1: very unfamiliar; 5: very familiar). 1 user had no prior experience with gesture typing. The median familiarity with gesture typing was 3.5. Participants were asked to use their preferred posture throughout the study: 6 participants used one thumb, and 6 used one index finger. The phrases were selected from a subset of the MacKenzie and Soukoreff phrase set [15, 29]. The same set were used for all participants.

Similar to Lu et al.'s work [13], we used a Nexus 5X device as a remote touchpad. The screen on the device was completely black and provided no visual feedback. Participants were seated on a chair 2 meters away from a smart TV. The phrase to be transcribed was shown on the TV, and the current target word was underlined. The task consisted of 4 blocks and each block had 10 phrases. Participants were allowed to take a short break after completing each block.

In total, the study included: 12 participants × 4 blocks × 10 phrases = 480 trials.

Results

To understand how a user locates the imaginary keys in eyes-free gesturing, we first inferred imaginary key positions from the observed gesture traces and analyzed their distributions.

Inferring Key Position from Gestures.
Unlike tap typing in which it is easy to map touch points to key positions, it is challenging to obtain the imaginary key positions from a continuous gesture trace. It is reasonable to assume that the starting and ending points on a gesture trace correspond to the starting and ending letters in a word, but much less obvious to assign key positions for the letters in between.

We located the imaginary key position based on the gestural shape, using the dynamic time warping (DTW) algorithm [19], a widely used algorithm for matching sequential

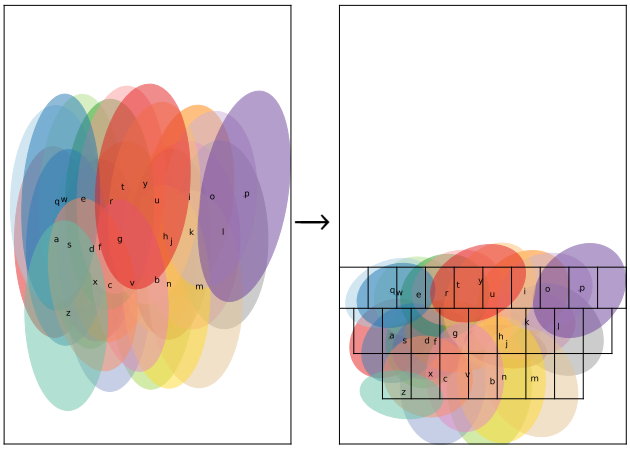
Figure 3: An example of inferring key positions of the word *nice* from its gesture. The solid line shows the gesture trace, while the dotted line shows the gesture template.

data. Assuming u was an input gesture for word w, we first created the template pattern v of w by connecting the centers of corresponding keys on a regular Qwerty layout. We used the Android AOSP keyboard on Nexus 5X to create the template. We then translated the centroids of u and v to the origin, sampled u and v into N ($N = 100$) equidistant points, and applied DTW to generate an optimal match between u and v. For a given letter c in the word w, it had a corresponding key center in the template pattern v, denoted by v_c. Its matching point in u, denoted by u_c, was the imaginary key center. If there were multiple points in u that matched v_c, we calculated the centroid of these points as the imagined key position. Figure 3 shows an example. As shown, this matching method satisfied our expectation that the starting and ending points on a gesture trace corresponded to the starting and ending letters in a word.

Distributions of Imaginary Key Positions.
Figure 4a shows the distributions of the imaginary key positions extracted from gestures. The imaginary positions of each key approximately followed a Gaussian distribution, with greater variance in y-direction. The mean standard deviation of imaginary key positions was $10.64mm$ in y, more than twice as great as the mean standard deviation in x ($4.88mm$). It was probably because the left and right edges of a touchpad served as references to bound the touch point distributions in x-direction.

Normalizing Imaginary Keyboard Vertical Position.
The large variance in the imagined key positions will likely cause ambiguity in decoding. However, we discovered that *normalizing the keyboard vertical position* can substantially reduce the variance.

Specifically, we first assumed there existed a keyboard k on the bottom of the remote control whose dimensions were identical to an Android AOSP keyboard. Given a gesture u, we vertically translated u so that the average distance between the imaginary key positions in u and corresponding key centers on the keyboard k was minimized.



(a) (b)

Figure 4: Touch point distribution on the imaginary keyboard (95% confidence ellipses) before (left) and after (right) normalizing the keyboard vertical position. The borders illustrate boundaries of the touchpad.

This translation essentially normalized the imaginary keyboard vertical position to the same location. Figure 4b shows the distribution of imaginary key positions after this process. The variances in y-direction were substantially reduced. The mean standard deviation in y-direction across letters was reduced to $4.60mm$, 57% smaller than the value before normalization. The drastic reduction showed that the great y-variance in imaginary key positions was largely caused by the unknown vertical location of the imaginary keyboard. In other words, normalizing the keyboard vertical position greatly reduced the y-variance of key distribution.

Figure 4b also showed that after normalization, the height and width of the keyboard were close to those of an Android keyboard across users and trials. It was probably because users often imagined the keyboard according to the virtual keyboards they used, and the common virtual keyboards shared the similar shape (3×10 Qwerty layout) and aspect ratio. For example, on a Nexus 5X device, the Android keyboard was $64.85 \times 27.62mm$; the Microsoft Swiftkey keyboard was $64.85 \times 26.42mm$; the Swype keyboard was $64.85 \times 25.82mm$. All were similar in dimensions. Participants were largely influenced by their daily typing experience on virtual keyboards.

Discussion

Our analysis led to the following findings.

Finding 1. Large key position variances caused by the unknown keyboard vertical position. Figure 4 shows that normalizing the keyboard vertical position reduced the key variance in y-direction by almost 50%, indicating a large

amount of variance was caused by the unknown keyboard vertical location. The reason, according to participants' comments, was the lack of visual or physical bounds to vertically locate the keyboard. In contrast, the horizontal position of the keyboard was easy to locate, because the left and right physical edges of the touchpad naturally defined the left and right boundaries of the keyboard.

Figure 4 also shows that after normalization, the width and height of the imaginary keyboard were close to that of a regular virtual keyboard on Android devices. That is, these two parameters did not deviate too much from a regular virtual keyboard in eyes-free input.

Finding 2. Small keyboard position offset from previous gesture. Although there was a large variance in keyboard vertical locations, the difference in the keyboard vertical position between the current gesture (denoted by Y_t) and the immediate previous gesture (denoted by Y_{t-1}) was small.

In the previous normalization procedure, assuming we needed to vertically translate Y_t by d_t for normalization, and translate Y_{t-1} by d_{t-1} for normalization. The mean (SD) of $\mid d_t - d_{t-1} \mid$, which reflected the absolute difference between Y_t and Y_{t-1}, was $3.09mm$ ($2.99mm$), accounting for only 10.4% of the keyboard height.

It showed that within a phrase, Y_{t-1} often served as a reference (or anchor) for the upcoming Y_t; a user tended to keep keyboard position relatively stable from word to word within a trial.

4 I'SFREE: SUPPORTING EYES-FREE GESTURE TYPING

As any text entry system, the key component is the decoding algorithm. Our eyes-free gesture decoding algorithm was designed based on the findings from Experiment 1 and the general gesture decoding principle [10, 32]. In this section, we first briefly review the principle of general gesture decoding, explain how eyes-free gesture decoding work, and introduce the implementation of i'sFree.

General Gesture Typing Decoding Algorithm

As outlined by Kristensson and Zhai [10, 32], a general gesture decoding algorithm takes a gesture trace (u) on a keyboard as input, and outputs N-best words with probabilities. In principle, it consists of the following two components:

- A language model that provides a prior probability of a word w based on the language context, denoted by $l(w)$.
- A spatial model that provides the likelihood of the gesture (feature) distribution for a given word w based on the geometry and location of gesture trace u, denoted by $c(w)$.

The final probability (score) of a word w given a gesture u is obtained by combing the language score $l(w)$ and spatial score $c(w)$ as:

$$s(w) = \frac{l(w)c(w)}{\sum_{i \in W} l(i)c(i)}, \tag{1}$$

where W is the set of words in the dictionary. The detailed description of such a decoder can be found in [10, 32].

i'sFree Gesture Decoding Algorithm

Decoding Principle.

The problem of applying the general decoding algorithm for eyes-free input is that it requires knowledge of the keyboard location and dimensions to calculate the spatial score $c(w)$. However, as shown in Experiment 1, although the keyboard width (W), height (H), and its horizontal position (i.e., X of top-left corner) remained largely unchanged, the vertical location of the keyboard (i.e., Y of top-left corner) was unknown in eyes-free input.

To address this problem, we introduce a *keyboard vertical location learner* in the decoding algorithm (Figure 5), which takes a gesture u as input, and outputs a range of possible keyboard vertical location: $Y \in [Y_{highest}, Y_{lowest}]$, where $Y_{highest}/Y_{lowest}$ is the highest/lowest possible keyboard vertical location. Note that $Y_{lowest} \geqslant Y_{highest}$, because the positive y direction is down in Android OS and iOS.

The algorithm then uses a Monte Carlo-based method to calculate the spatial score $c(w)$. It samples n keyboard vertical locations from this range, denoted by $Y_1, Y_2, Y_3, ..., Y_n$:

$$\begin{aligned} Y_1 &= Y_{highest}, \\ Y_i &= Y_{i-1} + \Delta, \end{aligned} \tag{2}$$

where $\Delta = (Y_{lowest} - Y_{highest})/n$.

For each Y_i, we use a regular gesture typing decoding algorithm (e.g., [10, 32]) to calculate the spatial score $c_i(w, Y_i)$. As informed by Experiment 1, we hypothesize that the keyboard width W is the same as the remote control width, the keyboard height H is the same as the height of an Android AOSP keyboard, and X is the left edge of the remote control (i.e., 0).

Assuming these n keyboard vertical positions are equally probable, the final spatial score of w is the average of these n scores:

$$c(w) = \frac{\sum_{i=1}^{n} c(w, Y_i)}{n}, \tag{3}$$

where $c(w, Y_i)$ is the spatial score of w given the keyboard's vertical location is at Y_i.

With both Equation (3) and Equation (1), we can calculate the final probability (score) of a word w given the input gesture u. The word w^* with the highest probability (score) is the intended word for u. The decoding task ends here.

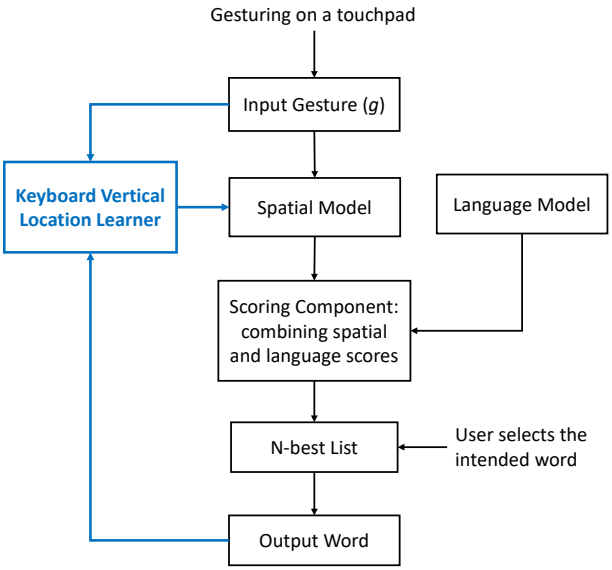


Figure 5: Architecture of the eyes-free gesture decoder. The only difference from a regular gesture typing decoder is the introduction of the Keyboard Vertical Location Learner highlighted in blue.

Figure 5 shows the architecture of the decoder. As shown, the only difference between the i'sFree and the general gesture typing decoding algorithm is the introduction of the keyboard vertical location learner. The following section explains in details how it works.

Keyboard Vertical Location Learner.
This module takes a gesture trace u as input, and outputs a range of possible keyboard vertical positions (i.e., $[Y_{highest}, Y_{lowest}]$). It includes the following two components working in parallel: a *geometric component* which makes estimation from the shape and location of input gesture u, and a *historical component* which learns from the historical typing data.

Geometric Component: Estimating $Y_{highest}$ and Y_{lowest} based on shape and location of a gesture trace. It is developed based on the intuition that a gesture trace should fall within the imaginary keyboard area when it is being drawn.

As a user is drawing the gesture u, the algorithm first locates P_h and P_l in u. P_h is the highest touch point in u, while P_l is the lowest touch point in u. To include both P_h and P_l in the keyboard area, the keyboard upper bound Y should be within the following range:

$$Y \in [P_{h_y}, \quad P_{l_y} - H], \tag{4}$$

where P_{h_y} is the y coordinate of P_h, P_{l_y} is the y coordinate of P_l, and H is the height of the imaginary keyboard, which

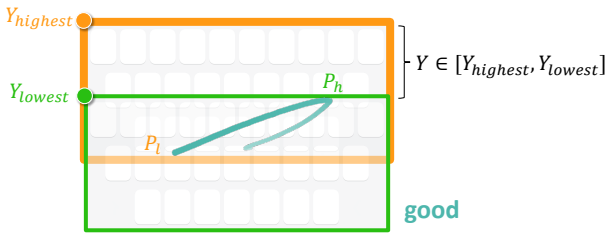


Figure 6: An example of learning the keyboard vertical location from the shape and location of the gesture trace. P_h and P_l are the highest and lowest points in the input gesture (green trace). Both of them should be included in the keyboard area. The green and orange boxes show the lower, and upper bound of the keyboard areas.

we assume is identical to the height of an Android AOSP keyboard. Figure 6 shows an example of how it works.

Historical Component: Estimating $Y_{highest}$ and Y_{lowest} based on previous input gesture. This method is developed based on Finding #2 from Experiment 1: the imaginary keyboard vertical location of the current gesture had only a small offset ($\sim$ 100 pixels, $6mm$) from that of the previous gesture.

After a user successfully entered the previous word w_{t-1} with gesture u_{t-1}, we estimated the keyboard vertical location for u_{t-1}, denoted by Y_{t-1}. This value is then used to estimate Y_t of the current gesture u_t as:

$$Y_t \in [Y_{t-1} - \epsilon/2, \quad Y_{t-1} + \epsilon/2], \tag{5}$$

where ϵ reflects how much the keyboard top boundary Y_t may deviate from the previous location Y_{t-1}. In our implementation, we chose $\epsilon = 6mm$, a value observed from Experiment 1.

Y_{t-1}, the vertical location of the previous keyboard, was calculated as follows. We located the corresponding imagined key positions for all the letters in w_{t-1}, following the procedure described in Section 3 using the DTW algorithm. For a letter c in w_{t-1}, if it was a top-row key in Qwerty (e.g., "Q", "W", "E"), the corresponding keyboard upper bound estimation Y_c would be:

$$Y_c = c_y - H/6, \tag{6}$$

where c_y is the y coordinate of the imagery key position of letter c, and H is the imaginary keyboard height.

Equation (6) was derived from the geometric property of a 3×10 Qwerty layout: each row had a height of $H/3$ and the center of a top-row key was half a key height away from the keyboard upper boundary. If c is a middle-row key (e.g., "A", "S", "D"), the upper bound estimation would be $c_y - H/2$; if c is a bottom-row key (e.g., "Z", "X", "C"), the upper bound estimation would be $c_y - 5H/6$.

Assuming there are n letters c_i ($i \in [1, n]$) in the word w_{t-1}, each letter c_i generates an estimation of the keyboard upper boundary Y_{c_i}. The final estimation Y_{t-1} is the average of across them:

$$Y_{t-1} = \frac{\sum_{i=1}^{n} Y_{c_i}}{n}. \quad (7)$$

Using Equation (7) with Equation (5), we can estimate the keyboard vertical location range of the current input gesture.

Combining Geometric and Historical Components. These two components work in parallel. Each generates an independent estimation of $[Y_{highest}, Y_{lowest}]$, denoted by R_g (from geometric component) and R_h (from historical component), respectively. By default, we use $R_g \cap R_h$ as the final estimation of Y range.

We revert to using the geometric component estimation R_g only in any of the following situations: (1) the user is entering the first word in an input session, which means no previous input gesture is available; (2) $R_g \cap R_h$ is empty, which means the current keyboard position deviates with a great offset from its previous location.

After obtaining $[Y_{highest}, Y_{lowest}]$, the keyboard vertical location learner passes it to the spatial model (Figure 5), which follows Equation (2) and Equation (3) to calculate the spatial score for a word candidate $c(w)$. We chose $n = 5$ in our implementation. It is then passed to the Scoring component in Figure 5 to generate the final score of a word candidate w.

Calculating the spatial score of a word w given a keyboard vertical location Y_i has been published by Kristensson and Zhai [10]. Note that the i'sFree gesture decoding algorithm is orthogonal to the actual implementation of a general decoder. In addition to the the algorithm described in [10], general gesture decoding algorithms in other keyboards such as Microsoft SwiftKey, SlideIt, Touchpal and Google Gboard can in principle be used as building blocks in the i'sFree gesture decoding algorithm as well, although they have not been published in the scientific literature. We implemented a general gesture typing decoder following the principle introduced in [10]: it estimates the spatial scores based on shape and location geometry feature distribution given w is the intended word.

Note that for each keyboard location (Y_i), we used the means of key center distributions in a normalized keyboard (Figure 4) as the actual key positions: it reflects where the user thought the keys were within an imaginary keyboard. Though the relative key positions largely followed their Qwerty orders, there are slight differences from key to key.

We consider our eyes-free decoding algorithm an extension of the regular gesture typing decoding algorithm [10, 32]. It is simplistic, and based on observations and hypotheses informed from the findings in Experiment 1. The only difference from a regular gesture typing decoding algorithm is the introduction of the keyboard vertical location learner, which learns the keyboard vertical location from both the instantaneous and historical input signals. The algorithm was developed based on the hypotheses that the keyboard width, height, and horizontal location remain unchanged.

Implementation of i'sFree

We built *i'sFree*, a text entry system that implemented the i'sFree gesture decoding algorithm. It was implemented on top of the open sourced Android keyboard, using a trigram language model whose lexicon size was 60K. Since it inherited the input method service in Android, it can serve any Android applications that need text input service, such as Messenger, Search Bar, and Intelligent Assistant (e.g., Google assistant).

i'sFree is an eyes-free gesture text entry method in which the keyboard is completely invisible. Some users, especially those who are not very familiar with Qwerty layout, might need to occasionally check the Qwerty location. To accommodate the need of these users, we implemented a pop-up keyboard reference as the *training wheel*. If a user cannot recall a certain key position during the eyes-free gesture typing, she may pause the finger movement (i.e., finger stays still) on the touchpad for 300 milliseconds and a Qwerty keyboard will appear on the output display. This keyboard reference will remain on the display till the input finger finishes the current gesture stroke. If the user can well recall key positions, she may gesture ahead and no keyboard reference will pop up.

The algorithm, models, and keyboard presented here were developed only for the purpose of testing the feasibility of eyes-free indirect word-gesture keyboards as a proof-of-concept input method. We think it is robust enough for empirical and live studies. We believe the concepts, principles, and empirical findings leading to the algorithm design are very useful for future more advanced, more accurate, and more efficient algorithms, if we can prove a sufficient level of usability of this algorithm in the next section.

5 EXPERIMENT 2: EVALUATING I'SFREE

We conducted a user study to evaluate the performance of i'sFree. We were especially interested in understanding whether the i'sFree gesture decoding algorithm was effective, and how users would perform eyes-free gesture typing with i'sFree.

Experiment Setup

Tasks. The study was a phrase transcription task. The experiment setup was similar to Experiment 1 (Figure 2). Participants used a Nexus 5X device running Android 6.0.1 as the

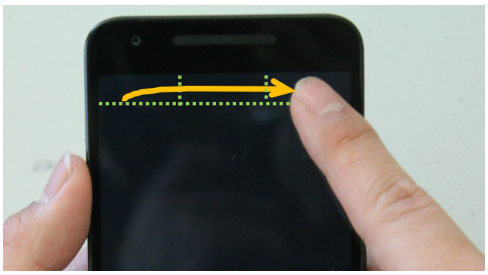


Figure 7: The control region on the phone screen. The green dotted lines indicated the regions corresponding to the first, second, and the third candidate word. The yellow arrow showed a right swipe, corresponding to the submit action.

remote touchpad. The phone screen remained black during the study session, and no gesture trail feedback was provided on the device. The gesture sequence was received on the input device and the decoding results were forwarded to a computer over Wi-Fi. A 46-inch smart TV was used to display the phrases and task information. Suggestions were shown on the TV screen.

To support basic text input, we enabled basic functions including choosing suggestions and backspacing on the remote touchpad. The top portion (1080 × 300 pixels) of the phone screen was used as the control region, as shown in Figure 7. The top suggestion was always automatically committed when users finished drawing a gesture, i.e., lifted the finger from the touchpad. The next three suggestions were displayed on the TV. Users could double tap on the left, middle or right of the control region to commit the 1st, 2nd or 3rd candidate word. A left swipe gesture in the control region was used as the backspace key. Since gesture typing keyboard performs word-level corrections, the backspace key would delete an entire word instead of a single letter. To reflect the natural typing behavior, participants were encouraged to use backspace and suggestions freely. After entering a phrase, users performed a right swipe gesture in the control region to submit the current phrase. Participants were instructed to enter the phrase as fast and accurately as possible.

The phrases were randomly selected from the MacKenzie and Soukoreff phrase set [15, 29], and different from the phrase set used in Experiment 1. The same set of phrases was used for all the participants but randomized in order for each participant.

Design. The study was a within-subject design. The independent variable was the keyboard condition with 3 levels:

- *i'sFree.* The keyboard we implemented with previously described i'sFree gesture decoding algorithm.
- *Eyes-free baseline.* It was identical to the i'sFree condition, except that it used a regular gesture typing decoder ([10, 32]). It allowed us to evaluate whether the proposed i'sFree gesture decoding algorithm was effective. The keyboard receiving input signals was located at the bottom of the touchpad and was invisible. The keyboard and gesture traces were hidden on the TV screen. Similar to i'sFree, if a user paused the finger over 300 ms on the touchpad, a reference keyboard would be displayed on the TV to help participants searching for keys.
- *Eyes-on baseline.* It represented the basic approach of adopting gesture typing on a remote control: a keyboard receiving input signals was located at the bottom of the touchpad and was invisible (because the touchpad had no display function). A Qwerty layout and the gesture traces were displayed on TV. It also used a regular gesture typing decoder following the principles in [10, 32].
  The purpose of including eyes-on baseline was twofold: (1) comparing it with eyes-free baseline allowed us to evaluate how reducing visual guidance would affect gesture typing performance; (2) comparing it with i'sFree would assess how i'sFree would improve input performance from the basic approach of performing gesture typing via a remote control.

The order of the three conditions was counterbalanced across participants. Before the formal study, participants performed a practice session with four phrases. Each task consisted of four blocks, and each block contained eight phrases. Participants were allowed to take a short break after the completion of each block.

Participants. 18 subjects (7 female, all right-handed) aged between 18 and 36 ($M = 27.7$) participated in the experiment. The participants were asked to rate their familiarity with the Qwerty layout on a 1-5 scale (1: very unfamiliar; 5: very familiar). The median familiarity with Qwerty layout was 4.5. Participants were not required to have experience with gesture typing: 7 users never used gesture typing before the study; 3 users performed gesture typing on a daily basis. Participants used their preferred posture throughout the study: 11 used index finger, and 7 used thumb.

In total, the study included: 18 participants × 3 conditions × 4 blocks × 8 phrases = 1728 trials.

Results

Overall Input Speed. This measure showed the overall speed of entering phrases. The calculation followed MacKenzie's equation [14]:

$$WPM = \frac{|S-1|}{T} \times \frac{1}{5}, \tag{8}$$

where S is the length of the transcribed string in character including spaces, and T is the elapsed time in minutes from the first gesture stroke to the last word selection.

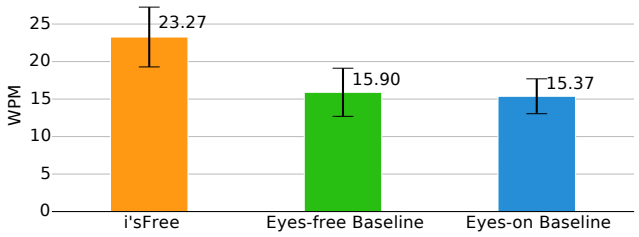


Figure 8: Means (95% confidence interval) of input speed.

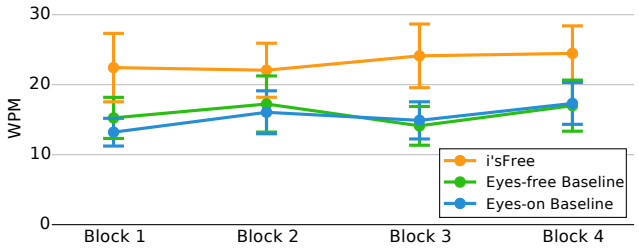


Figure 9: Means (95% confidence interval) of overall input speed by block for each keyboard condition.

Figure 8 shows the average (95% confidence interval) input speed across all participants. The average (SD) input speed in WPM were 23.27 (8.02) for i'sFree, 15.90 (6.45) for the eyes-free baseline, and 15.37 (4.68) for the eyes-on baseline. There was a significant main effect of keyboard condition on input speed ($F_{2,34} = 41.47, p < .001$). Pairwise t-tests with Bonferroni adjustment showed the difference was significant between i'sFree and the two baselines (both $p < .001$), but not significant between the eyes-free baseline and the eyes-on baseline ($p = 1$).

We also compared the overall input speed by block for each keyboard condition, as shown in Figure 9. The average speed within each block for i'sFree was higher than the other two conditions. There was a significant main effect of the block number on the input speed ($F_{3,51} = 7.212, p < .001$). A significant keyboard × block interaction effect was also observed ($F_{6,102} = 2.347, p = .037$). Pairwise comparison with Bonferroni adjustment showed the differences between Block 1 vs. Block 4 and Block 3 vs. Block 4 were significant ($p = .005$, $p = .017$ respectively).

Gesturing Speed. To understand how fast users drew gestures, we calculated *gesturing speed*:

$$S = \frac{\|L\|}{t}, \tag{9}$$

where $\|L\|$ is the length of a gesture and t is the elapsed time between touch down and touch up events for drawing the gesture. The unit is millimeters per second (mm/s).

The average (SD) gesturing speed was 104.26 (33.50) mm/s for i'sFree, 93.88 (37.16) mm/s for the eyes-free baseline, while it dropped to 74.00 (36.31) mm/s for the eyes-on baseline. The keyboard condition had a significant main effect on the gesturing speed ($F_{2,34} = 17.16, p < .001$). Pairwise t-tests with Bonferroni adjustment showed the difference was significant between the eyes-on baseline and the other two eyes-free gesture typing methods ($p < .005$), but not for i'sFree vs. the eyes-free baseline ($p = .13$).

Error Rate. Since gesture typing keyboard performed word-level corrections, we measured error rate with word error rate. The word error rate is based on minimum word distance, which is the smallest number of word deletions, insertions, or replacements needed to transform the transcribed string into the expected string. The word error rate [2, 34] is defined as:

$$r = \frac{MWD(S,P)}{|P|} \times 100\%, \tag{10}$$

where $MWD(S,P)$ is the minimum word distance between the transcribed phrase S and the target phrase P, and $|P|$ denotes the number of words in P.

The means (SD) of error rates were 2.14% (2.00%) for i'sFree, 5.04% ($SD = 5.57\%$) for the eyes-free baseline, and 2.95% (2.90%) for the eyes-on baseline. There was a significant main effect of the keyboard condition on the word error rate ($F_{2,34} = 6.803, p = .003$). Pairwise comparison with Bonferroni adjustment showed the differences between i'sFree and the eyes-free baseline was significant ($p = .039$), but not between other pairs.

Backspace Usage. To understand how often users corrected their mistakes to achieve such a low error rate, we also measured the backspace key usage. We define backspace to words ratio as:

$$d = \frac{N_d}{WordsCount(P)}, \tag{11}$$

where N_d is the number of backspace key presses in one trial, P is the target phrase in this trial, and $WordsCount(P)$ is the total number of words in P.

The mean (SD) backspace to word ratio was 0.26 (0.11) for i'sFree, 0.56 (0.21) for the eyes-free baseline, and 0.44 (0.15) for the eyes-on baseline. It showed users used backspace approximately once every 2 words in eyes-on baseline and eyes-free baseline, but once every 4 words in i'sFree. There was a significant main effect of the keyboard condition on the backspace to word ratio ($F_{2,34} = 32.68, p < .001$). Pairwise comparisons with Bonferroni adjustment showed the differences were significant for all pairs ($p < .05$).

Keyboard Pop-up Reference Usage. For i'sFree and the eyes-free baseline, we analyzed the average duration the keyboard pop-up was invoked in each trial. The average (SD) duration

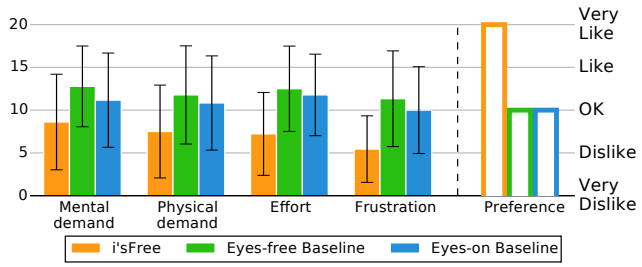


Figure 10: Left: Mean (SD) of the subjective ratings, all in 1-20 continuous scales where 20 is the most negative rating. Right: Median preferences on a 5-level scale.

was 2.54 (2.69) seconds for i'sFree, and 3.83 (2.71) seconds for the eyes-free baseline.

We compared the duration the keyboard reference was displayed against the trial duration, the average (SD) keyboard reference usage was 15.34% (12.38%) for i'sFree, and 15.48% (9.47%) for the eyes-free baseline. We did not observe a significant difference between the two conditions ($F_{1,17} = 0.009, p = .924$).

Subjective Measures. Subjective measures were collected at the end of the study. We used a subset of the questions in NASA-TLX [8] to measure participants' perceived workload. Users were asked to rate the mental demand, physical demand, effort, and frustration of the task on a continuous numeric scale of 1-20. The results are shown in Figure 10.

i'sFree was rated more positively than the other two conditions across all measures. We performed ANOVAs and there was a significant effect of the keyboard condition on all measures ($F_{2,34} = 6.562, p = .004$ for mental demand, $F_{2,34} = 8.678, p < .001$ for physical demand, $F_{2,34} = 12.81, p < .001$ for effort, and $F_{2,34} = 18.39, p < .001$ for frustration).

Each participant was asked to give an overall preference rating for each keyboard condition on a 5-level scale: 1 (very dislike) - 5 (very like). The median rating was 5 for i'sFree, 3 for the eyes-free baseline and the eyes-on baseline.

Discussion

Our design, development, and evaluation of i'sFree led to the following findings.

First, eyes-free gesturing is feasible. The similar performance between eyes-on baseline and eyes-free baseline indicated that reducing the visual guidance on a distant display hardly affected the gesture typing performance. Participants had faster gesture motion speed in eyes-free baseline (probably because they were freed from tracing letters), but were more error-prone which resulted in more backspace delete. The overall performance between these two conditions were similar.

Second, the i'sFree gesture decoding algorithm was effective. The input speed of i'sFree was 23.27 WPM, 46% faster than eyes-free baseline which used a regular gesture typing decoder. i'sFree reduced the backspace usage by 50% over eyes-free baseline. i'sFree was also 40.9% faster the eyes-on baseline.

Compared with reported input speeds from literature, i'sFree was much faster than the default text entry method on TV: controlling a cursor to select keys (7.66 WPM [13]). It was also faster than tap typing on a remote touchpad with keyboard displayed on TV (20.97 WPM [13]).

Third, i'sFree was an easy-to-learn input method. The average input speed was already 22.44 WPM in the first block, and around 40% of the subjects had never used gesture typing before. This promising result was partially due to users' familiarity with Qwerty layout which allowed them to locate key positions with limited visual cues. The pop-up keyboard reference was also helpful for novice too, although it was not used very often (around 15% of time). Users commented that they occasionally used it for entering long words.

Note that there are other approaches that enable eyes-free text entry on a fixed keyboard location, such as using a physical overlay or providing haptic feedback when the finger touches the boundaries of the keyboard. i'sFree is different in that it enables typing anywhere on a regular touchpad without any augmentation.

6 CONCLUSIONS

We present i'sFree, an eyes-free gesture typing method on a touch-enabled remote control. The i'sFree gesture decoding algorithm decodes gestures similar to a general gesture typing decoder, but learns from the instantaneous and historical input gestures to dynamically adjust the keyboard location. We have designed it based on the understanding of how users perform eyes-free gesture typing. Overall, i'sFree effectively supports eyes-free text entry, demonstrating that eyes-free gesture typing via a touch-enabled remote control is feasible and promising.

In addition to remote displays, i'sFree could be adopted in other scenarios such as VR/AR using a touch-enabled controller. Virtual and augmented Reality has been demonstrated in a variety of broad use cases. Motivated by recent advances [1, 21, 22], we envision that mapping the physical gesture to the virtual text can potentially improve the text entry efficiency in VR/AR.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. We thank our user study participants. This work was supported in part by NSF CHS-1815514, and a Google Faculty Research Award (2018).

REFERENCES

[1] Mahdi Azmandian, Mark Hancock, Hrvoje Benko, Eyal Ofek, and Andrew D. Wilson. 2016. Haptic Retargeting: Dynamic Repurposing of Passive Haptics for Enhanced Virtual Reality Experiences. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 1968–1979. https://doi.org/10.1145/2858036.2858226

[2] Xiaojun Bi, Shiri Azenkot, Kurt Partridge, and Shumin Zhai. 2013. Octopus: Evaluating Touchscreen Keyboard Correction and Recognition Algorithms via. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 543–552. https://doi.org/10.1145/2470654.2470732

[3] Xiaojun Bi, Ciprian Chelba, Tom Ouyang, Kurt Partridge, and Shumin Zhai. 2012. Bimanual Gesture Keyboard. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 137–146. https://doi.org/10.1145/2380116.2380136

[4] James Clawson, Kent Lyons, Thad Starner, and Edward Clarkson. 2005. The Impacts of Limited Visual Feedback on Mobile Text Entry for the Twiddler and Mini-QWERTY Keyboards. In *Proceedings of the Ninth IEEE International Symposium on Wearable Computers (ISWC '05)*. IEEE Computer Society, Washington, DC, USA, 170–177. https://doi.org/10.1109/ISWC.2005.49

[5] Leah Findlater, Jacob O. Wobbrock, and Daniel Wigdor. 2011. Typing on Flat Glass: Examining Ten-finger Expert Typing Patterns on Touch Surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2453–2462. https://doi.org/10.1145/1978942.1979301

[6] Sean Gustafson, Daniel Bierwirth, and Patrick Baudisch. 2010. Imaginary Interfaces: Spatial Interaction with Empty Hands and Without Visual Feedback. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 3–12. https://doi.org/10.1145/1866029.1866033

[7] Sean Gustafson, Christian Holz, and Patrick Baudisch. 2011. Imaginary Phone: Learning Imaginary Interfaces by Transferring Spatial Memory from a Familiar Device. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 283–292. https://doi.org/10.1145/2047196.2047233

[8] Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. In *Human Mental Workload*. Advances in psychology, Vol. 52. North-Holland, Oxford, England, 139–183.

[9] Yamada Hisao. 1980. A Historical Study of Typewriters and Typing Methods: from the Position of Planning Japanese Parallels. *Journal of Information Processing* 2, 4 (feb 1980), 175–202.

[10] Per-Ola Kristensson and Shumin Zhai. 2004. SHARK2: A Large Vocabulary Shorthand Writing System for Pen-based Computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM, New York, NY, USA, 43–52. https://doi.org/10.1145/1029632.1029640

[11] Frank Chun Yat Li, David Dearman, and Khai N. Truong. 2009. Virtual Shelves: Interactions with Orientation Aware Devices. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. ACM, New York, NY, USA, 125–128. https://doi.org/10.1145/1622176.1622200

[12] Shu-Yang Lin, Chao-Huai Su, Kai-Yin Cheng, Rong-Hao Liang, Tzu-Hao Kuo, and Bing-Yu Chen. 2011. Pub - Point Upon Body: Exploring Eyes-free Interaction and Methods on an Arm. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 481–488. https://doi.org/10.1145/2047196.2047259

[13] Yiqin Lu, Chun Yu, Xin Yi, Yuanchun Shi, and Shengdong Zhao. 2017. BlindType: Eyes-Free Text Entry on Handheld Touchpad by Leveraging Thumb's Muscle Memory. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 2, Article 18 (June 2017), 24 pages. https://doi.org/10.1145/3090083

[14] I. Scott MacKenzie. 2015. A Note on Calculating Text Entry Speed. http://www.yorku.ca/mack/RN-TextEntrySpeed.html.

[15] 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)*. ACM, New York, NY, USA, 754–755. https://doi.org/10.1145/765891.765971

[16] Anders Markussen, Mikkel Rønne Jakobsen, and Kasper Hornbæk. 2014. Vulture: A Mid-air Word-gesture Keyboard. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1073–1082. https://doi.org/10.1145/2556288.2556964

[17] Aske Mottelson, Christoffer Larsen, Mikkel Lyderik, Paul Strohmeier, and Jarrod Knibbe. 2016. Invisiboard: Maximizing Display and Input Space with a Full Screen Text Entry Method for Smartwatches. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*. ACM, New York, NY, USA, 53–59. https://doi.org/10.1145/2935334.2935360

[18] Douglas L Nelson, Valerie S Reed, and John R Walling. 1976. Pictorial superiority effect. *Journal of Experimental Psychology: Human Learning and Memory* 2, 5 (1976), 523.

[19] Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing* 26, 1 (February 1978), 43–49. https://doi.org/10.1109/TASSP.1978.1163055

[20] Oliver Schoenleben and Antti Oulasvirta. 2013. Sandwich Keyboard: Fast Ten-finger Typing on a Mobile Device with Adaptive Touch Sensing on the Back Side. In *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI '13)*. ACM, New York, NY, USA, 175–178. https://doi.org/10.1145/2493190.2493233

[21] Qi Sun, Anjul Patney, Li-Yi Wei, Omer Shapira, Jingwan Lu, Paul Asente, Suwen Zhu, Morgan Mcguire, David Luebke, and Arie Kaufman. 2018. Towards Virtual Reality Infinite Walking: Dynamic Saccadic Redirection. *ACM Trans. Graph.* 37, 4, Article 67 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201294

[22] Qi Sun, Li-Yi Wei, and Arie Kaufman. 2016. Mapping Virtual and Physical Reality. *ACM Trans. Graph.* 35, 4, Article 64 (July 2016), 12 pages. https://doi.org/10.1145/2897824.2925883

[23] Hussain Tinwala and I. Scott MacKenzie. 2010. Eyes-free Text Entry with Error Correction on Touchscreen Mobile Devices. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries (NordiCHI '10)*. ACM, New York, NY, USA, 511–520. https://doi.org/10.1145/1868914.1868972

[24] Keith Vertanen, Haythem Memmi, and Per Ola Kristensson. 2013. The Feasibility of Eyes-free Touchscreen Keyboard Typing. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*. ACM, New York, NY, USA, Article 69, 2 pages. https://doi.org/10.1145/2513383.2513399

[25] Cheng-Yao Wang, Min-Chieh Hsiu, Po-Tsung Chiu, Chiao-Hui Chang, Liwei Chan, Bing-Yu Chen, and Mike Y. Chen. 2015. PalmGesture: Using Palms As Gesture Interfaces for Eyes-free Input. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '15)*. ACM, New York, NY, USA, 217–226. https://doi.org/10.1145/2785830.2785885

[26] Yukang Yan, Chun Yu, Xiaojuan Ma, Shuai Huang, Hasan Iqbal, and Yuanchun Shi. 2018. Eyes-Free Target Acquisition in Interaction Space Around the Body for Virtual Reality. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM,

New York, NY, USA, Article 42, 13 pages. https://doi.org/10.1145/3173574.3173616

[27] Hui-Shyong Yeo, Xiao-Shen Phang, Steven J. Castellucci, Per Ola Kristensson, and Aaron Quigley. 2017. Investigating Tilt-based Gesture Keyboard Entry for Single-Handed Text Entry on Large Devices. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4194–4202. https://doi.org/10.1145/3025453.3025520

[28] Bo Yi, Xiang Cao, Morten Fjeld, and Shengdong Zhao. 2012. Exploring User Motivations for Eyes-free Interaction on Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2789–2792. https://doi.org/10.1145/2207676.2208678

[29] Xin Yi, Chun Yu, Weinan Shi, Xiaojun Bi, and Yuanchun Shi. 2017. Word Clarity As a Metric in Sampling Keyboard Test Sets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4216–4228. https://doi.org/10.1145/3025453.3025701

[30] Chun Yu, Yizheng Gu, Zhican Yang, Xin Yi, Hengliang Luo, and Yuanchun Shi. 2017. Tap, Dwell or Gesture?: Exploring Head-Based Text Entry Techniques for HMDs. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 4479–4488. https://doi.org/10.1145/3025453.3025964

[31] Shumin Zhai and Per-Ola Kristensson. 2003. Shorthand Writing on Stylus Keyboard. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '03)*. ACM, New York, NY, USA, 97–104. https://doi.org/10.1145/642611.642630

[32] Shumin Zhai and Per Ola Kristensson. 2012. The Word-gesture Keyboard: Reimagining Keyboard Interaction. *Commun. ACM* 55, 9 (Sept. 2012), 91–101. https://doi.org/10.1145/2330667.2330689

[33] 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 (CHI EA '09)*. ACM, New York, NY, USA, 2667–2670. https://doi.org/10.1145/1520340.1520380

[34] Suwen Zhu, Tianyao Luo, Xiaojun Bi, and Shumin Zhai. 2018. Typing on an Invisible Keyboard. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 439, 13 pages. https://doi.org/10.1145/3173574.3174013