DualKey: Miniature Screen Text Entry via Finger Identification

Aakar Gupta
University of Toronto
Toronto, Canada
aakar@dgp.toronto.edu

Ravin Balakrishnan
University of Toronto
Toronto, Canada
ravin@dgp.toronto.edu

ABSTRACT
Fast and accurate access to keys for text entry remains an open question for miniature screens. Existing works typically use a cumbersome two-step selection process, first to zero-in on a particular zone and second to make the key selection. We introduce DualKey, a miniature screen text entry technique with a single selection step that relies on finger identification. We report on the results of a 10 day longitudinal study with 10 participants that evaluated speed, accuracy, and learning. DualKey outperformed the existing techniques on long-term performance with a speed of 19.6 WPM. We then optimized the keyboard layout for reducing finger switching time based on the study data. A second 10 day study with eight participants showed that the new sweqty layout improved upon DualKey even further to 21.59 WPM for long-term speed, was comparable to existing techniques on novice speed and outperformed existing techniques on novice accuracy rate.

Author Keywords
Text entry; wearable; smartwatch; finger identification.

ACM Classification Keywords
H.5.2 Information interfaces and presentation: User Interfaces – Input Devices and Strategies.

INTRODUCTION
As wearable devices such as smartwatches, fitness bands and smart bracelets become ubiquitous, their miniature screens pose a challenge for interface designers for even common tasks like text entry. When the keys are in a full *qwerty* layout, the extremely small keys coupled with the fat-finger problem makes efficient text entry difficult.

Prior work [4,16,23] has tackled this by making the text-entry interaction a two-step process. In the first step, the user typically zooms in or scrolls to a particular zone on the keyboard which reduces the number of keys on the screen,

CHI'16, May 07 - 12, 2016, San Jose, CA, USA

ACM 978-1-4503-3362-7/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2858036.2858052

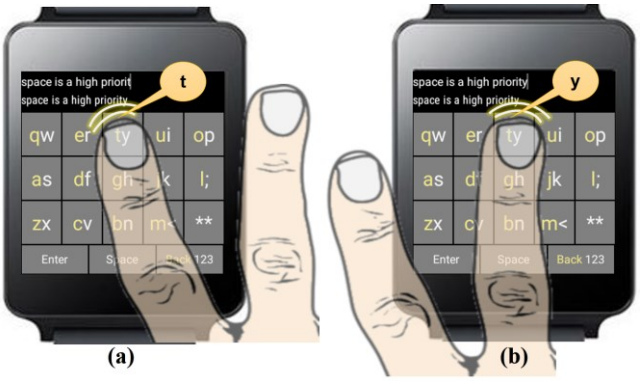


Figure 1: DualKey (a) Index finger types *'ty'* key's left letter *'t'* (b) Middle finger types *'ty'* key's right letter *'y'*

followed by key selection in the second step. While the techniques show impressive improvement over standard *qwerty*, the two-step interaction is cumbersome. Further, the best performing techniques [4,16] use swipe gestures as a way of discerning user intent on the miniature screen. Swipe or slide gestures are also used for shape writing [30] on regular soft keyboards. It is common to see people use both character-by-character entry and shape writing in conjunction, often in the same phrase. However, the use of swipe gestures for character-by-character entry precludes the possibility of shape writing on the proposed keyboards.

The question we explore is how to make the keys effectively bigger without expanding upon the number of steps or gestures beyond finger tapping? One potential solution to both is to make the touchscreen sensitive to finger identity when the tap hits the screen. Although there are no market-ready technical solutions, multiple promising threads of research [2,15,21] have demonstrated the feasibility of finger identification in larger scale systems like the tabletop. It is reasonable to expect that techniques for finger identification will eventually become viable for smaller screen devices, and hence it is worth exploring now if it is worthwhile using finger identification to improve the text input experience on small screens.

We propose DualKey – a solution to efficient text-entry on miniature screen devices using finger identification. DualKey eliminates the need for a two-step selection process, plus it does not require any swipe gestures for character-by-character entry. DualKey has a *qwerty* layout, but with half the keys of a full layout (Figure 1). Every key

corresponds to two characters, each character associated with a different finger. The left character on a key associates with the index finger, and the right associates with the middle finger. Typing *'the'*, for instance, would require the user to tap the keys *'ty'*, *'gh'*, and *'er'* with the index, middle, and index finger respectively. Prior research on single finger tapping [6] has shown that users are more comfortable, fast, and accurate when using their index or middle fingers as compared to the other fingers. We built a finger identification prototype for evaluation that has an accuracy of 99.5% on smartwatch scale touchscreens, implemented the DualKey technique using it, and ran two studies to evaluate its effectiveness.

RELATED WORK

Finger Identification

Prior work on finger identification either tackles the technical problem of identifying fingers or the potential interactions that result from identifying fingers. Sensing methods for finger identification on touchscreens include muscle sensing [2], gloves with fiduciary tags [21], and color markers [11]. Further, user touch authentication technologies use optical fingerprinting [15], capacitive coupling [8] or capacitive fingerprinting [13]. Most of these techniques are demonstrated for tabletops. Some, notably, fingerprinting and muscle sensing, hold promise for future miniaturization. Recent efforts by Apple and others [27,29] on embedding fingerprinting into displays show commercial interest in this space. Other techniques use a fixed orientation of the hand on the touchscreen to initialize a finger mapping [1,28], or use vision [10]. However, these techniques do not actually identify fingers and only work with multitouch gestures where the relative position of the fingers makes it known which finger is which.

Multiple works enrich finger input on smaller touch screens by detecting nuanced attributes like finger pressure [24], shear force [12], thumb touch size [3], nails and knuckles [14], and areas of the finger pad [17]. The exploration of interactions utilizing finger identification, however, is limited [2,7,10,11,21,26]. A popular use case is using different fingers to signal different modes or buttons – cut/copy/paste, menu shortcuts etc. [2,25,26]. Also suggested, is using fingers as containers for URLs [26], copied items [2], and brush colors [21]. Finger-specific chording has been used for multi-step commands [11], mixing brush colors [21], and a virtual mouse [10].

The above described works either explore finger detection, or propose interactions that use finger identification, or in some instances do both. However, there are no end-to-end applications based on robust finger identification that have been evaluated for performance using a working prototype. There is a lack of compelling investigation into how these interactions can be uniquely ingrained in applications, and how their performance can show improvements over basic finger interactions, if any. DualKey fills this gap and hopes to fuel the conversation on finger identification interactions.

Miniature Screen Text-Entry

A lot of work has been done in text-entry for smartphones. These techniques, however, do not translate well for miniature devices like smartwatches whose size makes it impossible to have 26 unambiguously accessible keys. Text-entry for this form factor has recently been addressed in multiple works [4,5,9,16,18,23]. They study typing performance in terms of speed (words-per-minute WPM), uncorrected error rate (UER), and total error rate (TER) which includes corrected and uncorrected errors. GPC refers to gestures-per-character – which indicates how many gestures (or taps) it takes, on average, to type a character. GPC includes space bar and has been estimated for all techniques [16] using the 500 phrase-set [20].

Zoomboard [23] uses a two-step tapping technique, to first zoom into the desired region, followed by selecting the key. With two taps for every character, besides space, Zoomboard had a GPC of 1.85. Zoomboard's speed after ~15 minutes of use on a smartwatch was 9.8 WPM with a TER of 7.1% [16]. As a baseline, standard *qwerty* achieved 13.7 WPM with a whopping 21.2% TER. Zoomboard was the first text-entry technique dedicated to miniature screens, but its performance was surpassed by later techniques.

Splitboard [16] splits the keyboard into left and right sections and shows one of these sections on the display at a time, with the user required to swipe left or right to get the desired section. With two columns of keys common to both sections, Splitboard has a GPC of 1.28. After ~15 minutes of use, Splitboard reported a speed of 15.3 WPM and a TER of 7.35%. Splitboard did not report a significant effect of blocks on speed and was not evaluated long-term beyond initial use. Given the simplicity of the technique, there doesn't seem to be an argument that performance might improve significantly over long term use.

Swipeboard [4] has a GPC of 2 and is based on zooming-in where the first swipe leads to 1 of 9 designated regions and the second swipe selects the key. It relies on a training protocol to maximize learning and reports a text entry speed of 19.5 WPM after ~2 hours of training, with an error rate of 17.48% for the whole experiment (including hard and soft errors [4]). While the keyboard size is smaller than the smartwatch form factor, it does not account for the errors as the technique is not based on precise key selection, but on memorization and performing gesture-pairs on the layout. The reported speed of text-entry accommodates this error rate by allowing for corrections. However, the high error rate is still a significant cause of concern because, in practice, this effectively means that 1 in 5 characters is typed wrong. This changes the dynamic of the text-input task: instead of a user typing relatively fluently with occasional glances at the output text area, here, she has to constantly focus at the output text and intermittently keep correcting the input. Further, Swipeboard did not use the standard experimental protocol of phrase entry. Instead, a single four letter word drawn from a limited word-set made

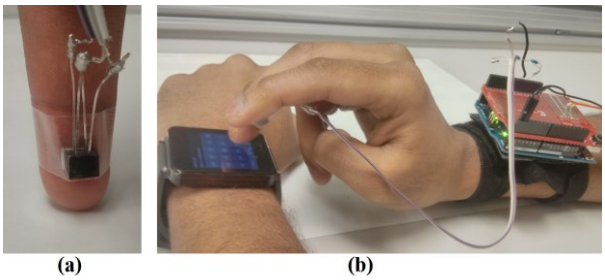
up of only five characters E, T, A, N, and S, and no others was entered. The authors acknowledge that these deviations from standard protocol artificially accelerate novice-expert performance growth. However, this does not provide an accurate representation of the expert performance of the technique. The results could vary on a longitudinal text-entry study with phrases derived from common language. Considering the highly reduced word-set, the initial performance of Swipeboard at the end of ~15 minutes of practice was low at 9.09 WPM. This is due to the emphasis on memorization of two-step gestures for every letter.

Other proposed techniques [5,9,18] are either slower or less accurate than the aforementioned or are not evaluated for performance. Flick input is used in Japanese keyboards to select different characters on the same key based on the flick gesture. Hong et al. [16] evaluated a form of flick input called Slideboard and found that Splitboard outperformed Slideboard. A potential direction of work in this space is related to keyboards that allow ambiguous input and rely on a language model and a miss model for predicting the word [19]. However, our work is focused on per-character entry, which is not supported by such keyboards. No reports have been published on the performance of such keyboards on small screens. While language models are an important part of today's text entry systems, per-character entry is supported by all popular soft keyboards. It allows entering words not in the lexicon, an ability which is routinely used by users not just for English conversations, but also conversations in other languages that are typed in English. For instance, Hindi speakers simply type Hindi words in English which are not present in the dictionaries.

In summary, while existing works have advanced the topic of smartwatch text-entry, there are multiple issues: Firstly, all techniques follow a two-step selection process, with a GPC > 1. Secondly, besides Zoomboard which has a slower performance, all techniques utilize swipes which precludes shape writing on the keyboards. Thirdly, the best performing techniques either have low peak performance or have high entry barriers. Splitboard was not evaluated beyond initial use and the data suggests that its performance plateaus quickly at 15 WPM. Swipeboard has a high error rate, low initial speed, and a very high entry barrier due to the memorization requirements.

DUALKEY

DualKey has a GPC of exactly 1, relying solely on single finger taps. It requires no swipe gestures, only taps, thus allowing for the possibility of simultaneous shape writing on the layout. The current prototype is built for a smartwatch scale touchscreen. DualKey leverages the distinction between the index and middle fingers to enable a single finger tap for a character. Instinctively, simply tapping the two fingers interchangeably on a smartwatch feels comfortable. We developed a prototype that identifies the finger touching the screen using an optical sensor, and sends it to the keyboard app on the watch via Bluetooth.



(a) (b)

Figure 2: (a) Sensor mounted on finger (b) Hardware Setup

Finger Identification Prototype

The finger tap is detected by the touchscreen. The prototype then needs to identify the finger whose tap was registered. We explored multiple techniques to identify index and middle fingers distinctly – tracking individual finger movement with optical markers, color markers, and leap motion; capturing differences in finger motion with ringed inertial motion unit (IMU) rings; and muscle sensing. However, none of these approaches worked proficiently within the constraints of our application requirements – a high precision accuracy for miniature screens, minimum instances of failure, low communication latency to the watch, and unobtrusive instrumentation. After multiple rounds of experimentation, we settled on our final working design that uses a combined miniature photo-transistor and optical detector sensor mounted on the index finger.

As shown in Figure 2, the sensor is connected to an Arduino, which processes the sensor data and sends it to an Android LG G smartwatch via a Bluetooth chip. The sensor detects its distance from the touchscreen. It is mounted and calibrated such that when the index finger touches the screen, the distance value is noticeably lower than when the index finger hovers over the screen. When the touchscreen registers a touch, the system checks if the distance value is low enough for the index finger to be touching. If not, the system determines that the touch was made by the middle finger. The single sensor performed well enough to discard the option of a sensor on the middle finger as well. This minimized instrumentation. An initial pilot with three users showed that upon individual calibration, ~99.5% accuracy was achieved. Each user typed in 25 phrases from 500 phrase-set [20]. However, the high accuracy is achieved at the cost of certain usability constraints. Since the sensor is mounted just below the finger pad, the users can only tap the screen using their fingertips. This somewhat constrains typing as the users cannot tap laterally with their finger pads. Also, even the tiny sensor adds to screen occlusion.

Keyboard

Figure 1 shows the keyboard layout. The *qwerty* layout was used to retain familiarity. Every key is associated with two letters, with the left letter corresponding to the index finger and the right letter corresponding to the middle finger. The *'**'* key is a swap key which enables the user to swap the letter just typed with its same-key counterpart. For example, on the *'er'* key, if a user intends to type *'r'*, but mistakenly uses the index finger and types *'e'*, she can immediately

swap it to *'r'* by using the swap key, instead of using backspace and typing the character again. Swap key was the result of an observation during pilot experimentation wherein almost half the user errors were due to using the incorrect finger on the correct key. The *enter*, *space*, and *backspace* keys are at the bottom of the screen. *Enter* and *Space* keys can be pressed with any finger. A middle finger touch on the *'Back'* key changes the keyboard layout to enable access to numbers and special characters.

PERFORMANCE EVALUATION: DUALKEY QWERTY

A 10-day study was conducted to investigate the long-term performance and learning curve of the technique. Since using finger identification is an entirely new interaction experience for the user, it is expected that initial performance will start slow, and will improve over time based upon how good the interaction technique is and how easy the learning curve is, in the absence of any explicit training. This makes it imperative to study this technique longitudinally. Further, there is no prior work that informs our understanding of input performance for finger identification, which makes it all the more important to investigate the learning curve of such interactions.

Participants

10 participants (9 male, mean age = 25.3) took part in the study. Eight did the study for 10 sessions, one per day, while two did the study for 15 sessions. Only one participant was a native English speaker, while others studied or worked in an English-speaking environment. All participants were regular smartphone users. None of them had experience with a smartwatch. All participants were right-handed and wore the watch on their non-dominant left hand. We did not account for left-handedness in the study.

Design

The study consisted of a series of sessions, one session per day. In each session, participants typed 25 random phrases sourced from Mackenzie et al's phrase sets [20]. No phrases were repeated, even across different sessions. The 25 phrases were divided up into five blocks with five phrases each, with an optional break between blocks. Each session lasted 6-15 minutes, depending on the participants' speed. Since speed improved over sessions, each participant went through ~90 minutes of typing by the end of 10 sessions. To explore if and when the performance plateaus, we extended the run for two randomly selected participants to 15 sessions. The sessions ran on consecutive days, with breaks during weekends. No breaks exceeded 2 days. Text-entry speed in WPM, and error rates UER, TER were recorded.

As is standard, participants were instructed to correct their mistakes as they went. If they did not detect the mistake until several characters later, they were asked to ignore it and continue. They were also asked to ignore system errors caused by occasional incorrect finger detections. The experiment started after a calibration phrase. In total, ((8 participants x 10 sessions) + (2 participants x 15 sessions)) x 5 blocks x 5 phrases resulted in 2750 phrases entered.

Apparatus

The sensor was mounted on the right hand index finger pad at a distance of ~10 mm from the fingertip. The experiment ran on an LG G android watch with a 29.6 x 29.6 mm touchscreen. Size of letter keys was 5.6 x 6.5 mm. The presented phrase and the typed phrase were shown above the keyboard area. Aside from the longitudinal aspect, our study adhered closely to the study design and keyboard lay-out used by Hong et al in their five-smartwatch-keyboard-study [16], in order to aide in cross-study comparisons.

RESULTS

Text-Entry Speed

Figure 3 shows the mean WPM by day (including error correction time). The steep curve over the first six days (sessions) shows a substantial performance improvement, followed by a slower improvement until the 10th day.

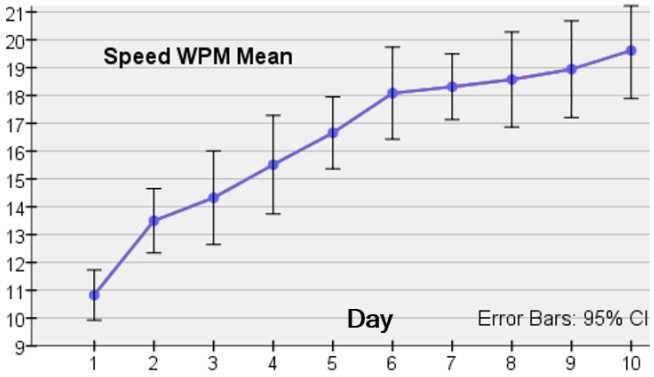


Figure 3: DualKey's Mean Speed WPM by Day

Figure 4 shows the curve for the 2 participants who did 15 sessions. Their performance continued to improve even after 10 days, with an average of 22.42 on the 15th day. This is notable given that DualKey is not designed as an expert technique which requires explicit training. The maximum speed reached was 24.7 WPM at the last session for P1. When considering speed of individual blocks, the maximum speed reached was 26.6 WPM. Interestingly, P2's curve appears to plateau during days 6-11, but then starts climbing again. This might suggest that DualKey's expert performance could potentially be even higher.

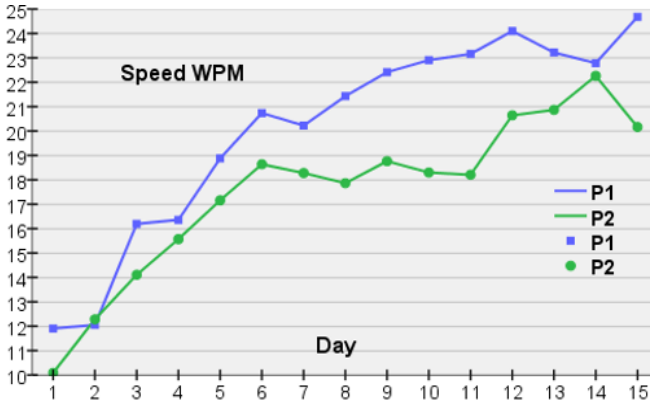


Figure 4: Speed of two participants P1 and P2 over 15 days

Technique	Uses Swipe	GPC	Novice WPM	Long-term WPM	Novice TER	Long-term TER
Zoomboard	**No**	1.85	9.80	*17.08*	~27.5%	19.64%
Splitboard	Yes	1.28	**15.30**	NA	7.35%	NA
Swipeboard	Yes	2	*9.09*	*19.58*	*~19.5%*	*17.48%*
DualKey QWERTY	**No**	**1**	12.31	**19.61**	**6.29%**	**5.25%**
DualKey SWEQTY	**No**	**1**	14.69	**21.59**	**5.67%**	**3.27%**

Table 1: DualKey's comparison with existing techniques. Swipeboard is italicized to indicate its accelerated evaluation design.

Error Rate

Figure 5 shows UER and TER by day. UER was low and remained relatively constant throughout, in the 1-2% range. This includes detection errors made by the system, which are approximately 0.5%. For a perfect finger detection system, UER, and consequently TER would be even lower.

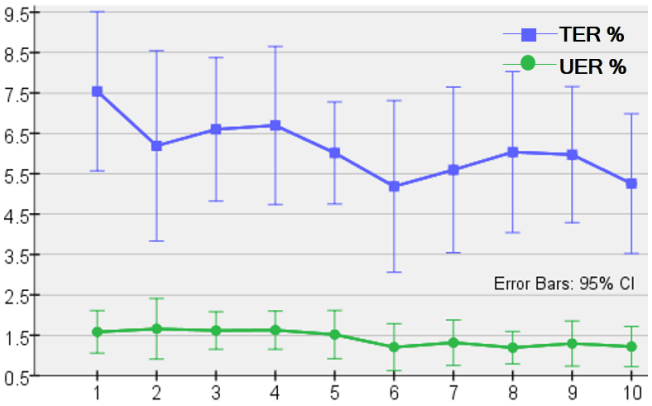


Figure 5: DualKey's TER, UER by Day

TER started out at 7.54%, eventually dropping to 5.26% on the last day. Participants made more errors on the first day and then stabilized for the rest of the days.

Comparison with Existing Techniques

Aside from GPC and swipe gesture use, we look at four performance metrics: novice speed, long-term speed, novice TER%, and long-term TER%. *Novice* performance refers to performance for the last block in the first session which equates to performance after ~15 minutes of practice. *Long-term* performance refers to the performance at the end of ~90 minutes of practice. Even though we have mirrored the smartwatch keyboards [16], it is not viable to compare these metrics across studies with statistical techniques due to the slight implementation differences between studies. However, for comparisons where the difference is visibly large, we can infer which techniques will potentially perform better. We compare DualKey with Splitboard and Swipeboard as they reported the best novice and long-term speeds respectively, and with Zoomboard since it does not require swipe gestures.

DualKey's novice speed is 12.3 WPM. This exceeds the equivalent speeds for Zoomboard (9.8 WPM) and Swipeboard (9.1 WPM). It is lower than Splitboard's 15.3 WPM. However, as mentioned earlier, Splitboard does not suggest a learning effect and reaches its peak performance early. No long-term study was conducted for Splitboard. Swipeboard, on the other hand, recorded a long-term speed of 19.58 WPM. Zoomboard was at 17.08 under the same conditions. However, these speeds were achieved after artificially accelerated learning. DualKey achieved a comparable speed of 19.61 WPM after ~90 minutes of practice under standard evaluation design. DualKey showed even more improvement with 5 more days (~2 hours of practice). Further, TERs of Swipeboard at both novice and long-term stages are prohibitively high at >17%. DualKey's TER was 6.29% at the novice stage. This is comparable to Splitboard's 7.35%. Table 1 lays out these comparisons for easy perusal. We'll discuss the last row in the next section.

DualKey's novice performance exceeds every other technique barring Splitboard. Since finger identification interactions are completely new for the user, an initial inertia is to be expected. As we see in Figure 3, the performance jumps sharply from Day 1 to Day 2, showing that the first day inertia is quickly overcome. However, it is important to understand what cause the initial inertia. We, therefore analyzed the data in more detail to understand the variables that negatively impact accuracy and speed.

FURTHER ANALYSIS OF ACCURACY AND SPEED

Analyzing DualKey Errors

To understand if it was too confusing for users to switch fingers, we analyzed all corrected errors to see how many were caused as a result of using the incorrect finger on the correct key. Figure 6 shows this finger-switching error rate (FER) besides the corrected error rate (CER). FER hovers around 2.5%. Essentially, close to half of the corrected errors were the result of the use of incorrect finger on the correct key. Again, FER is highest on the first day.

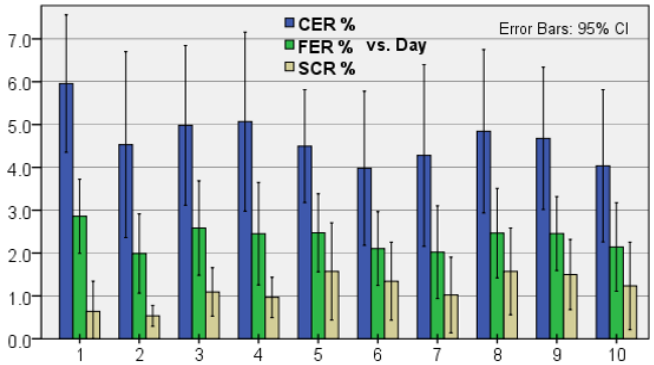


Figure 6: Mean Corrected Error Rate (CER), Finger Error Rate (FER), Swap-Correction Rate (SCR) over 10 days

We further analyze how often the participants used the swap key '**' to correct these finger switching errors. SCR% is the % of swap corrected errors. SCR tells us how helpful the swap key was based on how frequently participants used it to correct finger errors as opposed to backspace and correction. Figure 6 shows that in initial days, the swap key was infrequently used even when finger switching errors were high. Less than a quarter of finger switching errors were corrected using the swap key. This usage considerably improves over time. Participants use the swap key for more than half of the corrected finger errors after the fourth day. We asked participants about this at the end. Their responses indicate that initially they found it difficult to give up their habit of using backspace. As they used the swap key more, they found it easier to use. However, even at the end, the swap key is still not used for all finger switching errors. One participant commented that it depended on the flow of corrections – if the previous error was not a finger switching error, they would use backspace corrections and so even on a finger-switching error they would habitually go for the backspace key.

Analyzing DualKey Speed: Finger-Switching Time

In addition to movement and tapping time, the time taken to type a character relies on the user's swiftness in deciding if the finger needs to be switched for the character, and then in optionally switching the fingers if required:

$$T = T_{deciding} + [T_{switching}] + T_{movement} + T_{tapping}$$

In certain situations, *switching + movement time* could actually be faster than when there is no switching – for instance, when typing '*k*' after '*a*', the middle finger will naturally be positioned close to '*k*' after '*a*' has been typed, and the user simply needs to tap '*k*' with minimal movement. Depending on the *deciding time*, such a situation could be faster than one where there is no switching involved – to type '*x*' after '*i*', the user needs to decide and then move to the other diagonal end of the layout.

Consequently, we analyzed the time duration between two characters with respect to their finger configurations. There are four possible finger configurations for the index (I) and middle (M) fingers – II, IM, MI, and MM. For instance, typing '*k*' after '*a*', requires a switch from the index finger to the middle finger, and therefore is in IM. The analysis ignores the data points where *space* is involved because it can be typed with any finger. It also ignores the first character of the phrases.

Figure 7 shows the time duration between characters in seconds for the four finger configurations. The finger configuration has a significant effect on the time between characters ($F(3,297)=257.725$, $p<.001$). Pairwise comparisons found the difference in means between all finger combinations to be significant ($p<.001$).

The participants were fastest in the II configuration, taking the least amount of time by a large margin. Interestingly, MM takes the maximum time, even without the need to switch fingers. Further, IM takes longer than MI. In summary:

$$T_{II} < T_{MI} < T_{IM} < T_{MM}$$

While $T_{II} < T_{MI}$ is expected since T_{MI} involves switching fingers, the reasons for $T_{MI} < T_{IM}$ and $T_{IM} < T_{MM}$ are not immediately obvious. We examine them in more detail.

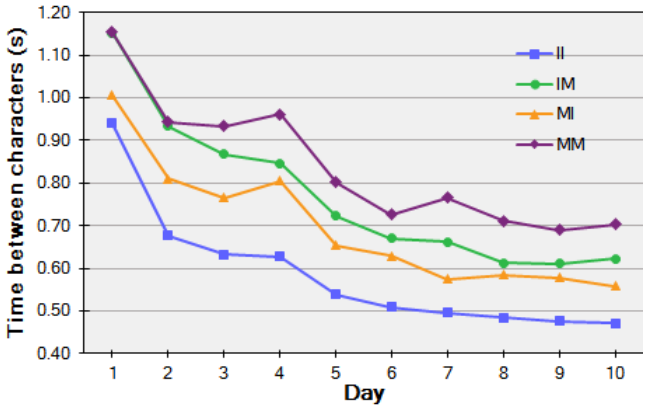


Figure 7: Mean time duration between characters for each of the 4 finger configurations over 10 days. I - Index, M - Middle

$T_{MI} < T_{IM}$: The data effectively says that regardless of the prior finger, it takes longer to type with the middle finger than the index finger. This can be explained by the fact that the middle finger is bigger, used less, and users have less precise control over its movements which causes them to spend more time to correctly position it over the right key.

$T_{IM} < T_{MM}$: Even though switching is not involved, typing with a middle finger when it follows another middle finger takes longer. One explanation is found in the frequency of occurrence of the MM finger configuration data points in the complete data. Among the four configurations, MM's frequency of occurrence is half of the others: II: 30.5%, IM: 27.6%, MI: 27.8%, and MM: 14.1%. Participants encountered the MM configuration less frequently than others and so were less used to MM than IM which causes the MM speed to be low. This hypothesis is confirmed by a closer look at Figure 7. Initially, when the users had no practice, both IM and MM start off at the same time duration. As the users get more experience, they encounter MM less frequently than IM, and the curves eventually diverge after the second session.

While the finger configuration analysis is instructive and promises to be useful for later optimizations, it admittedly does not capture all aspects of user latency. As we stated earlier, positioning of fingers with respect to the next character has a bearing on movement time. We did a coarse position-based analysis where we added another variable in addition to the finger configuration: left and right halves of the screen, assuming that moving to the other half of the screen takes up time for the same finger. However, the analysis did not yield significant results beyond what we already knew from the finger configuration analysis.

The increasing order of time for finger configurations indicates that if we maximize the instances of lower duration finger configurations and minimize higher ones, we can get an overall increment in speed. This will change the keyboard layout from *qwerty*. The optimal layout will assign the 26 letters to the index and middle fingers such that the average time between characters is reduced to a minimum. However, such a new assignment will mean that we deviate from the familiarity of the *qwerty* layout for a novice user. We perform an optimization of the keyboard layout that accounts for both the optimal assignment of letters to fingers and the closeness to *qwerty*, and results in a final finger-optimal near-qwerty layout.

OPTIMIZATION

The optimization is performed in two steps:

I Optimizing the assignment of fingers to letters such that the average time between characters is minimum.

II Optimizing the keyboard layout to be as close to *qwerty* as possible for the optimized assignment of fingers we get in Step I.

Step I: Finger Assignment Optimization

Procedure

Multiple participants remarked on the first day that they found finger switching to be difficult and it required certain mental effort to make the decision. They subsequently reported that it became much easier after the first day, and they were able to type more freely. During the middle sessions, participants mentioned that they no longer had to consciously think and type for several words. One participant commented:

> "You develop a rhythm with the fingers after some time and now it does not feel that different from normal typing."

However, to make DualKey more acceptable to users initially, we need to reduce the decision-making effort, as well as the time taken by finger switching. The analysis reflects this in that the average time for different finger combinations differed starting from the first day to the last day (aside from MI and MM initially). We, therefore, perform optimization such that the overall time between characters is reduced based on these differences in finger configuration time. Our aim is to find an optimal assignment of letters to fingers such that the more frequent letter pairs (bigrams) are associated with finger configurations having lower time duration. To this end, we define the following objective cost function:

$$\boldsymbol{Cost} = \sum_{\boldsymbol{i=a}}^{\boldsymbol{z}} \sum_{\boldsymbol{j=a}}^{\boldsymbol{z}} \boldsymbol{f_{ij}} \times \boldsymbol{t_{ij}}$$

Here f_{ij} is the frequency of a bigram *ij* in the English language corpus [22]; t_{ij} is the average time duration of the finger configuration associated with the bigram *ij*. In effect, t_{ij} has only four possible values, the average time durations: $T_{II}, T_{IM}, T_{MI}, T_{MM}$. For instance, both *'th'* and *'ef'* bigrams have the same $t_{ij} = T_{IM}$ for the *qwerty* layout. The cost function effectively calculates the normalized time it would take for a user to type the entire English corpus if she were to type every pair of letters in the time taken by that finger configuration on average. To minimize the overall time, the cost function needs to be minimized. For different keyboard layouts, the same letter could be assigned to different fingers, and consequently the same bigram will be allocated to different configurations, thus giving different cost outputs. For example, if a new layout is same as *qwerty*, just with key *er* replaced by *re*, then all t_{ij} associated with *e* and *r* will change – for instance, *qwerty*'s $t_{ef} = T_{IM}$ will be changed to $t_{ef} = T_{MM}$ for the new layout. However, two layouts can have equal cost functions if finger assignment is same for every letter.

After optimization, we need the optimal assignment of letters divided into two sets: $\mathbf{S_I}$ & $\mathbf{S_M}$, such that the cost function is minimized. For *qwerty* layout, the assignment is:

$\mathbf{S_I}$	A	B	C	D	E	G	J	L	M	O	Q	T	U	Z
$\mathbf{S_M}$	F	H	I	K	N	P	R	S	V	W	X	Y		

Notice that optimization is not affected by the specifics of the position of the letters on the layout, only in the assignment of a letters to one of the fingers. There will be multiple layouts that will satisfy the final assignments.

In the current DualKey layout, two keys have special characters (*;* <) for their middle finger association. Consequently, there are 14 slots for index finger and 12 slots for middle finger. We keep the positions of the special characters fixed, keeping the number of index and middle finger slots constant at 14 and 12. This will retain *qwerty's* number of letters per row and simplify the analysis.

The four t_{ij} values used for optimization are the individual average values, T_{II}, T_{IM}, T_{MI}, and T_{MM} over all 10 sessions: .581, .765, .691, and .831 respectively. For an assignment of 26 letters into groups of 14 and 12, there are total ${}^{26}C_{14}$ possible combinations that need to be evaluated. We wrote a matlab script that gave the optimal assignment of letters to fingers with the least value of the cost function.

The simplification of time taken between a bigram to the time taken by its finger combination admittedly does not capture the individual key level times. However, finger configuration is a significant contributor to a bigram's time and while the optimization's predicted values may not exactly replicate in practical use, it should certainly impact the speed positively and lessen decision-making effort.

Results

The baseline value of the cost function for the *qwerty* layout is .7006s. The globally optimal assignment is as follows:

$\mathbf{S_I^o}$	A	C	D	E	H	I	L	M	N	O	R	S	T	U
$\mathbf{S_M^o}$	B	F	G	J	K	P	Q	V	W	X	Y	Z		

The corresponding optimal value of the cost function is .6148s. This is a 12.25% improvement over *qwerty*'s .7006s. This implies that theoretically this assignment should increase the speed of DualKey by 12.25%. As a comparison, we ran the optimization for maximizing the cost function to get the worst possible assignment sets. The resultant cost value was .7952. The difference between the best and worst times is in a narrow range and *qwerty* lies somewhere in the middle.

Looking closely at the assignment sets, we see that the letters assigned to the index finger are mostly high frequency letters, E, T, A, etc. whereas the middle finger assignments are low frequency letters X, J, Q, Z etc. In fact, a closer look reveals that S_I^o contains the top 14 English letters by frequency [22], and S_M^o contains the bottom 12. This fits in perfectly with the notion that with the highest frequency letters associated with the index finger, users would need to switch to the middle finger rarely, thus reducing the MM, IM, and also MI configurations and increasing II configurations. Using the frequency table [22], we see that the optimal assignment results in 86.5% of the taps being made by the index finger.

Our optimization algorithm was designed to minimize the time taken between characters. To see how would an optimization based on minimizing the number of *finger switching instances* would perform, we ran another optimization with the same cost function, but with t_{ij} replaced with b_{ij}:

$$\boldsymbol{Cost_{switches}} = \sum_{i=a}^{z}\sum_{j=a}^{z} \boldsymbol{f_{ij}} \times \boldsymbol{b_{ij}}$$

Here, b_{ij} is a binary variable with a value 1 for IM and MI configurations and 0 for II and MM. This reduces finger switching regardless of individual time taken. Not surprisingly, the optimization resulted in the exact same optimal assignment sets. The cost of switching for *qwerty* was .5792, which implied that on average, to type the English corpus, the number of finger switches required would be .5792 per bigram or one switch for every second bigram. The optimal cost value came out to be .2220, which is an improvement of 61.7%. The number of finger switches is reduced to less than one switch for every four bigrams. This is a sizeable reduction in the number of finger switches the user has to deal with, thus lessening the decision-making effort. While theoretically, the speed should improve by 12.25%, the potential reduction in decision-making effort should also impact participants' speed positively, improving the speed even further.

The new assignment sets mean that the layout can no longer be *qwerty*. This could impact the initial performance of users who are habituated to soft *qwerty* keyboards. Our optimization, though, only results in optimal assignment sets and not a fixed keyboard layout. Consequently, we perform a layout optimization that resulted in a keyboard layout which was closest to the *qwerty* layout amongst all the eligible layouts for the optimal assignment set.

Step II: Nearest-qwerty Layout Optimization

Procedure

With 14 letters assigned to the index finger, and 12 to the middle finger, there are 14!x12! = 4.1E+19 eligible layouts for the optimal assignment set. $\mathbf{S_I^o}$ and $\mathbf{S_M^o}$ were optimized individually over their 14 and 12 slots.

A keyboard is considered near to *qwerty* based on the distances of the positions of all letters in the new layout from their positions in *qwerty*. The coordinates were assigned to the 26 slots in the following manner: The first top-left slot is assigned (x, y) = (0, 0) and the x-coordinate increases by 1 for every move to a slot on the right. The y-coordinate similarly increases by 1 for every move to a slot below. For instance, for the *qwerty* layout, *q* has coordinates (0, 0), *p* has (9, 0), and *m* has (6, 2). The distance of a letter in layout L from its position in the *qwerty* layout Q is defined as:

$$\boldsymbol{d_{LQ}^i} = \sqrt{\left((\boldsymbol{x_L^i} - \boldsymbol{x_Q^i})^2 + (\boldsymbol{y_L^i} - \boldsymbol{y_Q^i})^2\right)}$$

Here x_L^i refers to the x-coordinate of the ith letter in a layout L, and x_Q^i refers to the x-coordinate of the ith letter in the *qwerty* layout Q. Based on this distance, we define the distance cost function for a layout L:

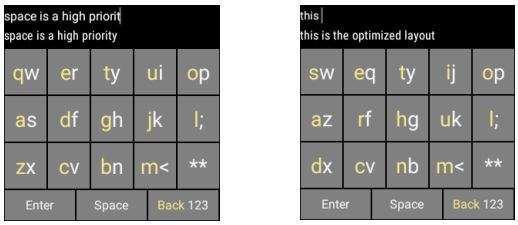
$$\boldsymbol{Distance\ Cost} = \sum_{i=a}^{z} \boldsymbol{d_{LQ}^i} \times \boldsymbol{f_i}$$

Here f_i is the frequency of the ith letter in the English language corpus. This is a weighted distance cost function such that more weight is given to the near-*qwerty* positioning of the most frequent letters.

The globally optimal layout will minimize the weighted distance cost function to give us the layout that is closest to *qwerty*. Since the cost function involves the sum of weighted distances, we can write it as the sum of two independent parts: the weighted distance cost function for letters in $\mathbf{S_I^o}$, and the weighted distance cost function for letters in $\mathbf{S_M^o}$. Because the letters in $\mathbf{S_I^o}$ can never occupy the slots designated for letters in $\mathbf{S_M^o}$, and vice versa, these two cost functions can be independently optimized. Thus, we obtain the globally optimal layout by iterating through only 14!+12! = 8.8E+10 layout permutations.

Results

Figure 8(b) shows the optimal layout that resulted from the optimization. We term it the SWEQTY layout. The optimal weighted distance cost value is .53. The unweighted distance of SWEQTY from QWERTY is 16.9. As the figure shows, the highest frequency letters in both $\mathbf{S_I^o}$ and $\mathbf{S_M^o}$ have retained their *qwerty* positions if their finger assignment was same as *qwerty*. As a comparison, the farthest layout from *qwerty* had a weighted distance cost of 2.95 and an unweighted distance of 93.6.



(a) **(b)**

Figure 8: (a) DualKey QWERTY (b) DualKey SWEQTY

SWEQTY is the optimal layout in terms of *time taken between characters*, *finger switching instances*, and *closeness to qwerty* for any DualKey keyboard with the same slot layout. The same process can be easily applied to any other slot distribution for any DualKey variation.

PERFORMANCE EVALUATION: DUALKEY SWEQTY

With *sweqty*, the problem of initial unfamiliarity with the keyboard is somewhat alleviated due to the closeness with QWERTY. Combined with the potentially reduced decision-making effort and higher speed, we hypothesize that we will see a net improvement in the novice, as well as the long-term speed and error rates. To validate our hypothesis we ran another 10-day study with eight participants with the exact same design as the first QWERTY study. All participants were different from the earlier study. In total, we had 8 participants x 10 sessions x 5 blocks x 5 phrases = 2000 typed phrases.

Results

The *sweqty* speed by days is shown in Figure 9. The effect of days on speed was significant ($F(9, 63) = 34.887$, $p<.001$). The mean novice speed of *sweqty* is 14.69 WPM, and the long-term speed is 21.59, both of which are higher than *qwerty*. SWEQTY's curve is higher than QWERTY. The novice speed shows a 19% improvement over *qwerty*.

Sweqty speeds show a higher variance than *qwerty*. This is because a subset of participants texted heavily on their smartphones every day. Consequently, they were using *qwerty* soft keyboards daily, in addition to *sweqty*. The performance of these users was affected as a result, resulting in higher variance.

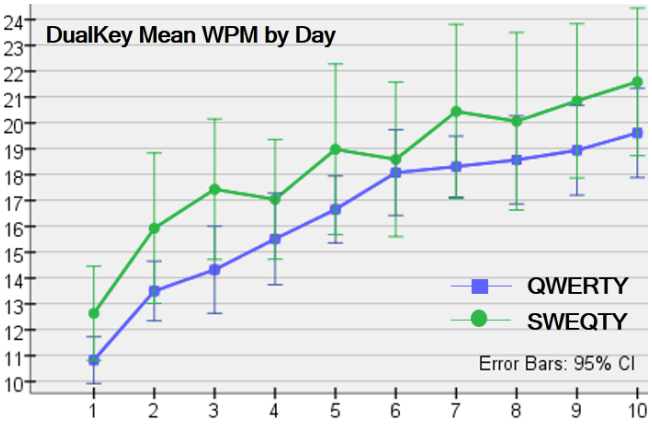


Figure 9: Mean Speeds of DualKey: SWEQTY vs. QWERTY

The novice and expert TERs of sweqty are 5.67% and 3.27% respectively. A mixed ANOVA on error rates for 10 days yielded a significant difference between TERs for qwerty v/s sweqty: $F(1,16 = 6.737, p<.05)$. Figure 10 shows the TER %s. The novice TER improved by 10.93%, and the long-term TER improved by 37.71%, which is a rather large improvement. The sweqty TER% starts off close to *qwerty*, but falls rapidly to values considerably lower than *qwerty*. The relatively high error rates initially in both techniques are because of the techniques being unfamiliar to the users. The fall after first day is pronounced in sweqty because of less finger switching and the low frequency of middle finger taps.

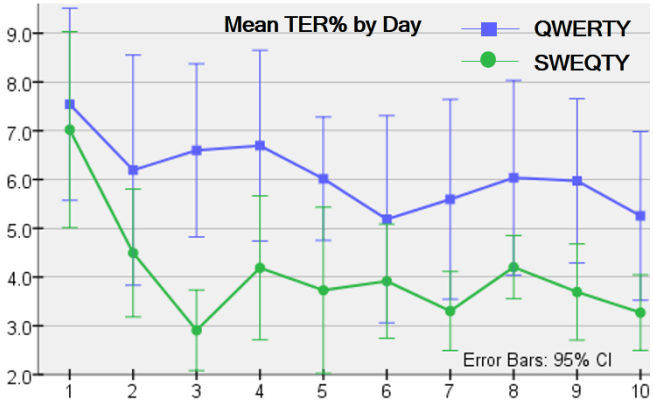


Figure 10: Mean TER% of DualKey: SWEQTY v/s QWERTY

In the Q&A for *sweqty*, participants mentioned that they felt they needed to switch fingers very infrequently and the technique was easy to pick up. When asked about the non-qwerty layout, one of the participants commented:

> "It looked weird. But when I actually started typing, I did not have to actively search for the alphabets a lot."

Looking back at Table 1, we see that all four metrics show improvements over DualKey QWERTY. While *sweqty* v/s *qwerty* speeds do not show significance, there is significant difference between their TERs. At novice stage, the speed improves 19% and TER improves 10.9% despite a non-qwerty layout which shows that the gains of switching to a near-qwerty optimal layout outweigh the familiarity of a *qwerty* layout for DualKey even at the starting point. The huge improvement in error rates boosts the usability of *sweqty* further, especially in the long-term.

SWEQTY further extends the improvement that DualKey showed over existing techniques. Although novice speed does not exceed Splitboard's 15.30, its value of 14.69 can be considered comparable. The novice TER, on the other hand, at 5.67% is lower than Splitboard's 7.35%. Both *qwerty* and *sweqty* perform comparably or better than others on all four metrics. To put it another way, DualKey at least performs comparably to the best performing novice technique while exceeding others, and performs better than all existing techniques on long-term performance.

DISCUSSION

Performance Improvements

Even after optimization using the *sweqty* layout, there is further scope for improvement in multiple areas. First, our custom hardware limited the degree of freedom for the user's fingers and thus constrained the user's ability to do free form typing. Second, even though it was small, the finger sensor occluded the screen by a bit, which further impeded user's performance. An unobtrusive finger identification technology will further improve the performance. In fact, aside from fingerprinting techniques, we can imagine miniature sensor rings that the user can wear that will reduce movement constraints or occlusion.

Third, incorporating auto-correction into the keyboard will allow the user to make more imprecise taps which will increase the overall typing speed. However, auto-correction is not applicable in every situation, as people routinely type words that are not in the dictionary. Further, DualKey can be adapted for other languages that do not work as well with dictionary corrections.

Finger identification leads to huge number of possibilities for simplifying and augmenting interactions. Therefore, as and when the technology gets mature, we will potentially see such applications on the rise and slowly become a part of the natural interaction ecosystem. In an ecosystem where finger identification interactions are a norm, the users' novice performance will certainly be superior.

Limitations and Future Work

Our study does not account for left handedness. There is an added wrinkle when DualKey is implemented for the left hand. Since the index finger is on the right of the middle finger, the characters corresponding to the index finger should be on the right part of a key. This will change the layout from *qwerty/sweqty* to something that is farther from *qwerty*. While the process will remain the same, left handed DualKey needs to be studied independently.

DualKey has been designed and optimized for a smartwatch form factor. However, there are even smaller screens where DualKey's efficacy needs to be evaluated. As an extension, a TriKey model can be studied where a single key corresponds to three letters associated with three fingers (or 2 fingers + thumb), reducing the space requirements even further. The insights we have gained from DualKey can be helpful in designing the optimized versions of such TriKey keyboards. As we have seen, the optimal finger assignment results in allocating the highest frequency letters to the index finger. We can apply the same principle for TriKey, where the highest frequency letters are allocated to the index finger, followed by the middle finger, and finally the ring finger or the thumb.

Designing Finger Identification Interactions

Our analysis provided us some insights on designing interactions that use finger identification. First, reducing finger switching even at the cost of impacting familiarity of an interface is an inquiry worth undertaking for any new application. Second, middle finger taps expectedly take longer and should be assigned low frequency keys. Third, such interactions will be prone to incorrect finger usage mistakes, and providing a quick *Swap* or *Undo* method will be useful. It also reduces the cost of an incorrect-finger tap in the user's mind, thus relaxing the interaction without the user having to worry about making finger errors. Text-entry has a latent implication that it can be undone easily. However, keys like *Send* cannot be undone. It is recommended that buttons that trigger commands that cannot be undone easily should not be associated with multiple fingers.

The space of interactions using finger identifications is starting to grow out of its infancy. What is needed is not just listing a library of interactions, but also end-to-end applications and in-depth analyses that provide a compelling argument for these interactions and how they improve and augment our current interactions. DualKey hopes to be a progressive step in that direction.

CONCLUSION

We presented DualKey, a novel technique for miniature screen text entry via finger identification. We built a custom hardware that performed finger identification with a 99.5% accuracy which involved detecting subtle finger movements. We conducted a comprehensive long-term study of the technique and do an in-depth analysis of the speed and error rates. The error analysis showed the usefulness of having the swap button. Based on the speed analysis, we optimize the assignment of letters to fingers to reduce the finger switching time. Based on the new assignments, we optimize the layout to get a nearest-to-qwerty layout as close to familiar as possible. We conduct another long-term study with eight participants for the new *sweqty* layout to validate the theoretical premise. Despite being a *non-qwerty* layout, *Sweqty* improves upon DualKey in all respects, including improved novice and long-term speeds & error rates, as well as reduced initial decision-making effort.

DualKey (*qwerty/sweqty*) beats the existing techniques in multiple respects: (1) Using single step selection, instead of the prevalent two-step selection, it has a GPC of 1 in contrast to a minimum GPC of 1.28 among existing techniques. (2) In contrast to the best performing techniques, DualKey does not require swipe gestures, thus allowing for parallel shape writing on the layout. (3) DualKey's long-term performance, both in terms of speed and error rate is better than the reported long-term speeds of other techniques, even while not being a typical expert technique. (4) DualKey *qwerty's* novice performance exceeds all others barring Splitboard. DualKey *sweqty* improves upon this novice performance and is comparable to Splitboard in terms of novice speed and betters Splitboard in terms of error rate.

REFERENCES

1. Oscar Kin-Chung Au and Chiew-Lan Tai. 2010. Multitouch finger registration and its applications. *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction - OZCHI '10*, ACM Press, 41. http://doi.org/10.1145/1952222.1952233

2. Hrvoje Benko, T. Scott Saponas, Dan Morris, and Desney Tan. 2009. Enhancing input on and above the interactive surface with muscle sensing. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '09*, ACM Press, 93. http://doi.org/10.1145/1731903.1731924

3. Sebastian Boring, David Ledo, Xiang "Anthony" Chen, Nicolai Marquardt, Anthony Tang, and Saul Greenberg. 2012. The fat thumb. *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services - MobileHCI '12*, ACM Press, 39. http://doi.org/10.1145/2371574.2371582

4. Xiang "Anthony" Chen, Tovi Grossman, and George Fitzmaurice. 2014. Swipeboard. *Proceedings of the 27th annual ACM symposium on User interface software and technology - UIST '14*, ACM Press, 615–620. http://doi.org/10.1145/2642918.2647354

5. Hyeonjoong Cho, Miso Kim, and Kyeongeun Seo. 2014. A text entry technique for wrist-worn watches with tiny touchscreens. *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology - UIST'14 Adjunct*, ACM Press, 79–80. http://doi.org/10.1145/2658779.2658785

6. Ashley Colley and Jonna Häkkilä. 2014. Exploring finger specific touch screen interaction for mobile phone user interfaces. *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures the Future of Design - OzCHI '14*, ACM Press, 539–548. http://doi.org/10.1145/2686612.2686699

7. Ashley Colley, Jani Väyrynen, and Jonna Häkkilä. 2015. In-Car Touch Screen Interaction: Comparing Standard, Finger-Specific and Multi-Finger Interaction. *Proceedings of the 4th International Symposium on Pervasive Displays - PerDis '15*: 131–137.

8. Paul Dietz and Darren Leigh. 2001. DiamondTouch. *Proceedings of the 14th annual ACM symposium on User interface software and technology - UIST '01*, ACM Press, 219. http://doi.org/10.1145/502348.502389

9. Mark D. Dunlop, Andreas Komninos, and Naveen Durga. 2014. Towards high quality text entry on smartwatches. *Proceedings of the extended abstracts of the 32nd annual ACM conference on Human factors in computing systems - CHI EA '14*, ACM Press, 2365–2370. http://doi.org/10.1145/2559206.2581319

10. Philipp Ewerling, Alexander Kulik, and Bernd Froehlich. 2012. Finger and hand detection for multi-touch interfaces based on maximally stable extremal regions. *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces - ITS '12*, ACM Press, 173. http://doi.org/10.1145/2396636.2396663

11. Alix Goguey, Géry Casiez, Daniel Vogel, Fanny Chevalier, Thomas Pietrzak, and Nicolas Roussel. 2014. A three-step interaction pattern for improving discoverability in finger identification techniques. *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology - UIST'14 Adjunct*, ACM Press, 33–34. http://doi.org/10.1145/2658779.2659100

12. Chris Harrison and Scott Hudson. 2012. Using shear as a supplemental two-dimensional input channel for rich touchscreen interaction. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, ACM Press, 3149. http://doi.org/10.1145/2207676.2208730

13. Chris Harrison, Munehiko Sato, and Ivan Poupyrev. 2012. Capacitive fingerprinting. *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, ACM Press, 537. http://doi.org/10.1145/2380116.2380183

14. Chris Harrison, Julia Schwarz, and Scott E. Hudson. 2011. TapSense. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, ACM Press, 627. http://doi.org/10.1145/2047196.2047279

15. Christian Holz and Patrick Baudisch. 2013. Fiberio. *Proceedings of the 26th annual ACM symposium on User interface software and technology - UIST '13*, ACM Press, 41–50. http://doi.org/10.1145/2501988.2502021

16. Jonggi Hong, Seongkook Heo, Poika Isokoski, and Geehyuk Lee. 2015. SplitBoard. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, ACM Press, 1233–1236. http://doi.org/10.1145/2702123.2702273

17. Da-Yuan Huang, Ming-Chang Tsai, Ying-Chao Tung, et al. 2014. TouchSense. *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*, ACM Press, 189–192. http://doi.org/10.1145/2556288.2557258

18. Luis A. Leiva, Alireza Sahami, Alejandro Catala, Niels Henze, and Albrecht Schmidt. 2015. Text Entry on Tiny QWERTY Soft Keyboards. *Proceedings of the 33rd Annual ACM Conference on Human Factors in*

Computing Systems - CHI '15, ACM Press, 669–678. http://doi.org/10.1145/2702123.2702388

19. Frank Chun Yat Li, Richard T. Guy, Koji Yatani, and Khai N. Truong. 2011. The 1line keyboard. *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, ACM Press, 461. http://doi.org/10.1145/2047196.2047257
20. I. Scott MacKenzie and R. William Soukoreff. 2003. Phrase sets for evaluating text entry techniques. *CHI '03 extended abstracts on Human factors in computing systems - CHI '03*, ACM Press, 754. http://doi.org/10.1145/765891.765971
21. Nicolai Marquardt, Johannes Kiemer, David Ledo, Sebastian Boring, and Saul Greenberg. 2011. Designing user-, hand-, and handpart-aware tabletop interactions with the TouchID toolkit. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS '11*, ACM Press, 21. http://doi.org/10.1145/2076354.2076358
22. Peter Norvig. 2013. English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU. *norvig.com*. Retrieved August 24, 2015 from http://norvig.com/mayzner.html
23. Stephen Oney, Chris Harrison, Amy Ogan, and Jason Wiese. 2013. ZoomBoard. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*, ACM Press, 2799. http://doi.org/10.1145/2470654.2481387
24. Graham Alasdair Wilson Phd Thesis. Using Pressure Input and Thermal Feedback to Broaden Haptic Interaction with Mobile Devices. Retrieved September 8, 2015 from http://theses.gla.ac.uk/4363
25. Quentin Roy, Yves Guiard, Gilles Bailly, Éric Lecolinet, and Olivier Rioul. 2015. Glass+Skin: An Empirical Evaluation of the Added Value of Finger Identification to Basic Single-Touch Interaction on Touch Screens. *INTERACT: IFIP International Conference on Human-Computer Interaction*, Springer.
26. Atsushi Sugiura and Yoshiyuki Koseki. 1998. A user interface using fingerprint recognition. *Proceedings of the 11th annual ACM symposium on User interface software and technology - UIST '98*, ACM Press, 71–79. http://doi.org/10.1145/288392.288575
27. Nate Swanner. 2015. Sonavation has bonded 3D fingerprint sensors to Gorilla Glass. *thenextweb.com*. Retrieved September 11, 2015 from http://thenextweb.com/insider/2015/07/21/sonovation-has-bonded-3d-fingerprint-sensors-to-gorilla-glass-kiss-your-home-button-goodbye/
28. Julie Wagner, Eric Lecolinet, and Ted Selker. 2014. Multi-finger chords for hand-held tablets. *Proceedings of the 32nd annual ACM conference on Human factors in computing systems - CHI '14*, ACM Press, 2883–2892. http://doi.org/10.1145/2556288.2556958
29. M Yousefpor and JM Bussat. 2014. Fingerprint Sensor in an Electronic Device. *US Patent App. 14/ ...*. Retrieved September 11, 2015 from https://www.google.com/patents/US20150036065
30. Shumin Zhai, Per Ola Kristensson, Pengjun Gong, et al. 2009. Shapewriter on the iphone. *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems - CHI EA '09*, ACM Press, 2667. http://doi.org/10.1145/1520340.1520380