Octopus: Evaluating Touchscreen Keyboard Correction and Recognition Algorithms via "Remulation"

Xiaojun Bi[1] **Shiri Azenkot**[2] **Kurt Partridge**[1] **Shumin Zhai**[1]
[1]Google Inc.
Mountain View, CA, USA
{bxj, kep, zhai}@google.com
[2]University of Washington
Seattle, WA, USA
shiri@cs.washington.edu

ABSTRACT
The time and labor demanded by a typical laboratory-based keyboard evaluation are limiting resources for algorithmic adjustment and optimization. We propose *Remulation*, a complementary method for evaluating touchscreen keyboard correction and recognition algorithms. It replicates prior user study data through real-time, on-device simulation. To demonstrate remulation, we have developed Octopus, an evaluation tool that enables keyboard developers to efficiently measure and inspect the impact of algorithmic changes without conducting resource-intensive user studies. It can also be used to evaluate third-party keyboards in a "black box" fashion, without access to their algorithms or source code. Octopus can evaluate both touch keyboards and word-gesture keyboards. Two empirical examples show that Remulation can efficiently and effectively measure many aspects of touch screen keyboards at both macro and micro levels. Additionally, we contribute two new metrics to measure keyboard accuracy at the word level: the *Ratio of Error Reduction (RER)* and the *Word Score*.

Author Keywords
Simulation; Text Entry; Touch Screen Interaction

ACM Classification
H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms
Design; Human Factors

INTRODUCTION
Today, touchscreen keyboards are used by hundreds of millions of people around the world as their default text entry method. To reduce typing errors, most of these "Smart Touch Keyboards" (STKs) correct errors automatically as users type. However, occasionally the error correction system itself makes a mistake, with undesirable and sometimes humorous consequences [6,20].

An increasingly popular alternative to the STK is the smart gesture keyboard (SGK). An SGK recognizes words based on the user's finger gestures (Figure 1, right). SGK's are also known as Shape Writing keyboards [21, 28], gesture keyboards [5] or word-gesture keyboards [29] in the literature. Different forms of SGKs have been commercially distributed in many products such as ShapeWriter, SlideIT, Swype, Flex T9, TouchPal, and the Android 4.2 stock keyboard (Gesture Typing) [2]. SGKs face the same correction challenges as STKs because they must map ambiguous finger gestures to words.

Figure 1. Illustrations of Smart Touch Keyboard (Left) and Smart Gesture Keyboard (Right)

HCI research has explored various techniques for error prevention, including adapting decoding algorithms to hand posture [4], and personalizing ten-finger typing for large touchscreens [7].

As with other advanced UI technologies such as speech recognition, effective and efficient evaluation is critical to the improvement of smart keyboards. An evaluation generally consists of (1) data collection and (2) data analysis. Our goal is to facilitate both stages of the process. Collecting keyboard output data typically involves a laboratory experiment with a dozen or more participants. The time and labor required by these experiments make frequent evaluation of small algorithmic changes infeasible.

Moreover, current data analysis techniques do not provide important information about keyboard algorithms. For example, they do not explicitly and quantitatively measure an STK's ability to correct user errors, and the typical accuracy metrics (e.g., the MSD error rate [16]) examine output at the character-level, and do not reflect today's keyboards' word-level behaviors.

To expand the repertoire of tools and methods for evaluating STKs and SGKs, we propose *Remulation,* a novel keyboard evaluation approach that measures correction and recognition algorithms of keyboards by *re*playing previously collected user data through real-time on-device si*mulation*. It takes touch screen events recorded

CHI 2013, April 27–May 2, 2013, Paris, France.

during prior user studies as input, and injects them into a live mobile keyboard, at the same rate as they were collected. The original experiment is thereby replicated on a working keyboard. Remulation can efficiently evaluate many (not all) aspects of both STKs and SGKs without conducting laboratory experiments, and can be done repeatedly as if the same group of participants were employed to type the same input tirelessly over and over on different keyboards.

Typical use cases for Remulation include: (1) a developer seeks to evaluate the impact of an algorithmic change to her keyboard, (2) a researcher seeks to evaluate different versions of the same third-party keyboard, and (3) a device manufacturer wants to select a third-party keyboard to embed in its devices without inspecting the keyboards' proprietary source code or algorithms.

To effectively measure the accuracy of a keyboard algorithm, we introduce the metrics of *Word Score*, which reflects the number of correct words out of every 100 input words from a given test dataset, and *Ratio of Error Reduction* (RER), which quantifies an STK's error prevention and correction ability. Both metrics measure keyboard accuracy at the word level.

In the rest of this paper we demonstrate the Remulation approach and its corresponding data analysis methods by designing and implementing Octopus, a Remulation-based keyboard evaluation tool. We put Remulation and Octopus into practice by applying them to two STKs and two SGKs, revealing various insights at both the macro and micro levels.

In summary, this work contributes a novel approach to evaluating touchscreen keyboards, which includes:

- *Remulation*, a new approach for evaluating keyboard correction and recognition algorithms by replicating prior user study data with real-time simulation
- The design and implementation of Octopus, a Remulation-based keyboard evaluation tool and system
- A demonstration of Remulation and Octopus in real use, with evaluations of two STKs and two SGKs
- *RER* and *Word Score*, new metrics for evaluating keyboard accuracy.

RELATED WORK

Collecting natural use data and applying them to train recognition algorithms has been widely adopted as a research methodology in AI (e.g., speech and handwriting recognition). Recently, it has also been employed to design keyboard algorithms. Gunawardana et al. [9] used pre-recorded data on a keyboard to train and evaluate their own error correction algorithm. Remulation uses some of the same techniques for the different problem of comparing two or more third-party keyboards when the source code or algorithm is unavailable. Our work shows that it is possible to send "fake" touch events to today's mainstream devices (Android in particular but other OSs in principle) and evaluate keyboards in a "black box" fashion.

In what follows we discuss relevant prior work on (1) simulating human text entry for evaluating keyboard performance without traditional laboratory studies, and (2) current analysis techniques used for assessing a text entry method's accuracy.

Simulating Text Entry

In 1982, Rumelhart and Norman [23] described a model for simulating skilled typists on physical typewriters, with the goal of understanding human typing behavior. Their model focused on predicting keystroke timing and simulated key transposition and doubling errors. In our work, we rely on direct data replication rather than complex modeling of human performance, and seek to evaluate keyboard algorithms rather than theorize user behavior.

Since the early 1990's, research interest has shifted from studying physical typewriters to soft keyboards. The Fitts-digraph model, first proposed by Lewis [12], was used to estimate average text entry speeds based on movement time between pairs of keys and digraph frequencies. This model has been a popular performance prediction tool for keyboard evaluation with different layouts using a single finger or a stylus [12,13,14,19], and has been used as an objective function for keyboard optimization [26]. A two-thumb physical keyboard predictive model has also been proposed [15]. Unlike these models, which predict an upper bound for average text entry speed assuming a certain error rate implied by Fitts' law (4% per target), our Remulation and analysis method assesses keyboard error rates and accuracies using the same speed that had naturally occurred in the data collection experiment. Also, we focus on evaluating keyboards with similar appearances but different algorithms, instead of different layouts.

Measuring Accuracy in Text Entry

Standard text entry accuracy metrics compare the transcribed and presented strings. The Minimum String Distance (MSD) [16] is often used to measure the "distance" between the two strings, based on the number of character insertions, deletions, and replacements needed to turn one string into another. While this has been effective in measuring traditional physical keyboards that literally output every letter typed, it is insufficient for measuring STKs. STKs embed a dictionary or language model and do not necessarily map touch points to text on an individual letter basis. Similarly, SGKs usually work at the word level: they recognize a continuous finger gesture to output a single word [21, 28, 29]. Since both SGKs and STKs operate at word-level, it is more meaningful to adopt the word-level metrics. Furthermore, although word-level metrics (e.g., WER) are common in other fields such as speech recognition [1], they are rarely used to measure keyboard performance.

Wobbrock and Myers [25] analyzed the character input stream in addition to the presented and transcribed strings. They assume that text flows serially character by character.

This assumption does not hold for STKs, in which whole words may be algorithmically modified after they are entered. The character-level metric also does not suit SGKs well.

Building upon this prior work, we develop new word-level metrics to measure keyboard accuracy.

OCTOPUS: A REMULATION-BASED KEYBOARD EVALUATION SYSTEM

Based on the Remulation concepts described earlier, we designed and implemented Octopus (named after the conference room "Dr. Octopus" where the concept was first discussed). Octopus is a Remulation-based touchscreen keyboard evaluation tool. Figure 2 shows its architecture, which consists of (1) the Simulator, (2) Dataset, and (3) the Keyboard Output Receiver.

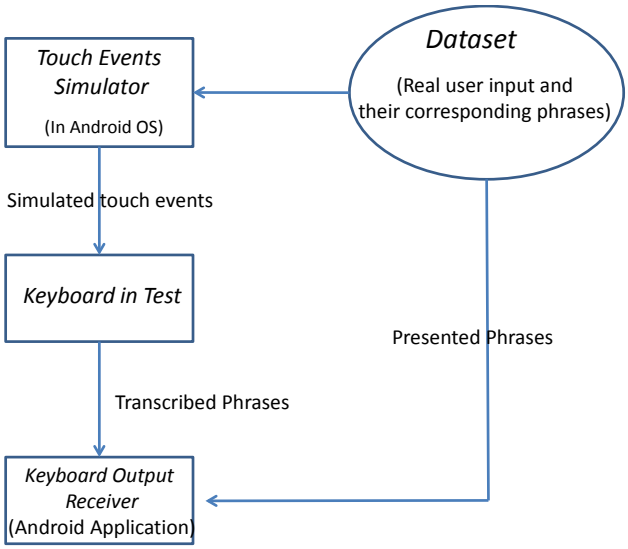


Figure 2. The architecture of the Octopus System.

(1) Touch events simulator. This component simulates touch actions on a mobile device in real time according to the dataset. It can simulate TOUCH_DOWN, TOUCH_MOVE, and TOUCH_UP events, which are the three basic touch operations on a mobile device. Using these events as building blocks, Octopus can simulate all the typical touch interactions. For example, a quick tap usually consists of a TOUCH_DOWN event immediately followed by a TOUCH_UP event, and a gesture usually consists of a TOUCH_DOWN event followed by several TOUCH_MOVE events and a TOUCH_UP event.

The Simulator accurately specifies the interval between every two touch events. The precision of such intervals between two events is less than ±10 ms. It can also simulate multi-finger interaction by specifying the finger ID of a touch event. These features allow Octopus to keep fidelity high in simulation, which is critical for evaluating modern soft keyboards. For example, a keyboard might adjust its algorithm according to the typing speed of a user. Simulating touch actions in real time is critical to measure such algorithms. When a user quickly types with two thumbs, she might land the second thumb before lifting the first one, generating multi-touch events. Octopus enables us to investigate how a keyboard handles these situations.

(2) Dataset. The dataset is fed into Octopus to simulate real users' text entry actions. The datasets consist of touch events and their corresponding phrases in *presented phrases*. Each touch event includes the event type (i.e., TOUCH_DOWN, TOUCH_MOVE, or TOUCH_UP), the (x,y) screen coordinates, the timestamp, and the finger identifier.

The collected data aims to reflect fundamental human performance, independent of particular visual design elements, keyboard features, or algorithms. Also, to challenge keyboards' algorithms and better discriminate different keyboards, the dataset strives to capture users' relaxed, natural and uncorrected typing behaviors.

In the current implementation of Octopus, the dataset is collected through lab studies in which participants type or gesture the presented phrases on a mobile device as naturally and as quickly as possible, using a collector keyboard. The collector keyboard provides users with only asterisks as feedback when they enter text, to prevent them from adjusting their input behaviors to take advantage of certain keyboard algorithms and features (such as deleting a whole word at a time).

Ideally, the layout and dimensions of the collector keyboard are identical to the test keyboard used during Remulaiton. If the test keyboard has slightly different dimensions, touch points can be scaled and translated according to the target keyboard's height, width, and top-left corner location. The validity of such transformations should be further empirically verified in future research, particularly when the transformation is large.

(3) Keyboard Output Receiver. This is an application (Figure 3) that runs on the touch screen device at the same time as the Simulator that generates the touch events.

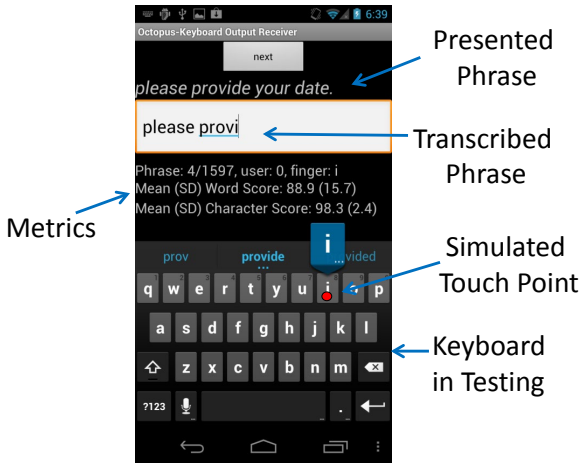


Figure 3. The UI of Keyboard Output Receiver as Octopus *remulates* the phrase "please provide your date." The red dot marks a simulated touch point.

The Keyboard Output Receiver includes a standard text view widget that receives the *transcribed phrases* from the keyboard. The communication between the keyboard and the Keyboard Receiver is through the mobile OS input method framework, so Remulation can be performed on any keyboard that is installed on the device. The Keyboard Output Receiver logs the *transcribed phrases* generated by the keyboard, compares them with the *presented phrases* from original Dataset, and then calculates and displays the quality measures of the keyboard being tested (Figure 3).

STRENGTHS AND LIMITATIONS

The Octopus *remulation* method is but one method of keyboard evaluation. Text entry is a complex process and no single method, such as a laboratory experiment, instrumented field deployment, or modeling and prediction, can provide a complete understanding of a keyboard. In our view, the Remulation-based approach offers another method that has both strengths and limitations.

Strengths. The advantages of this method include:

1. *Efficiency*. Empirical data can be used and reused multiple times to evaluate different keyboards. Efficiency is important for rapid algorithm iteration, since a developer can determine the impact of algorithm changes without new human participants.
2. *Fidelity*. This approach faithfully replicates participants' behavior in real time during the user study. It can precisely specify the time intervals between touch events, and also can support multi-touch input.
3. *Sensitivity*. Running Octopus is like employing the same group of participants typing over and over on multiple keyboards. There are no confounding variables like time of day, fatigue, or learning effects, so the impact of small algorithm tweaks can be measured reliably.
4. *No source code required*. This approach offers a "black box" evaluation: one must only install a keyboard on a mobile device to evaluate it. This enables evaluation of third-party keyboards without accessing their proprietary algorithms or source code.

Limitations. Octopus focuses on correction and recognition algorithms of keyboards. It does not evaluate the entire user experience of a touchscreen keyboard. In particular, it does not evaluate UI-related interaction behaviors, such as selecting the target via the suggestion bar or using the backspace key. Also, it is limited to keyboards with the same layout as the one used in data collection (e.g., Qwerty).

Because the user data is collected a priori and the keyboard for data collection does not provide the user with feedback for correct or erroneous input, Octopus does not account for changes in user behavior in response to keyboard output. Octopus focuses on "open loop" typing aspects in which the user types ahead and trusts the keyboard to correct their imprecise input (be it touch or gesture). This type of data reflects the most basic and natural input behaviors, unencumbered by the UI and algorithms, but does not capture feedback-driven behavior adjustments.

IMPLEMENTATION

We implemented Octopus on the Android operating system. The initial simulator was implemented on a desktop machine. It constructed touch events according to the Android protocol and sent them to the device using Android Debug Bridge (ADB) [3] via a USB cable. The limitations of this approach were 1) it was difficult to accurately control time intervals between simulated events because of the latency variance between the device and the machine; 2) simulated events occasionally went missing due to disruptions in the USB connection; and 3) the protocol for constructing touch events varied across devices, requiring extra engineering per device.

The current Simulator was implemented on top of Android's Monkey event simulation tool. It runs on a device without direct connection to a host computer. Since it simulates touch events at the OS level, it works on any Android device. From the perspective of a keyboard application, events from the simulator are indistinguishable from events that are generated by the device's touch screen.

The Keyboard Output Receiver is a standard android application developed in Java.

METRICS FOR MEASURING KEYBOARD ACCURACY

Octopus is designed to evaluate the effectiveness of keyboard algorithms. This section introduces the measures used: *Character Score*, *Word Score*, and *Ratio of Error Reduction.*

Character Score

Character Score is based on Minimum String Distance, (MSD) [16], which is the smallest number of character insertions, deletions, and replacements needed to transform one string into another:

$$MSD\ error\ rate = \frac{MSD(P,T)}{MAX(|P|,|T|)} \quad (1)$$

where P is the presented phrase and T is the transcribed phrase. To more intuitively and directly reflect accuracy, we convert the error rate to *character score*:

$$Character\ Score = (1 - MSD\ error\ rate) \times 100 \quad (2)$$

The character score is between 0 and 100, and approximately indicates the percentage of correct characters. The higher the score, the more accurate the keyboard. A keyboard with a score of 100 is error-free. However, note that this score depends on the test dataset. A dataset could include fundamental errors (e.g. input based on misread words from *presented text*), so a score of 100 may not be achievable.

Word Score

Word Score is based on Minimum Word Distance (MWD), which is the smallest number of word deletions, insertions,

or replacements needed to transform one string into another.

$$MWD\ error\ rate = \frac{MWD(P,T)}{MAX(|P|,|T|)} \quad (3)$$

A word is defined as a string of characters entered between one or more continuous spaces. $|P|$ and $|T|$ are the lengths of the presented and transcribed phrases, measured in number of words.

Similar to Character Score, Word Score is defined as:

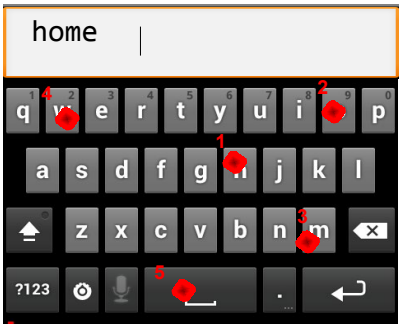
$$Word\ Accuracy\ Score = (1 - MWD\ error\ rate) \times 100$$

It approximately represents the percentage of correct words for a given dataset. Like Character Score, it is also dataset-dependent.

Ratio of Error Reduction (RER)

Character and *Word Scores* measure the overall accuracy of a given dataset, but they do not specifically measure a keyboard's correction and recognition capabilities. Comparing transcribed to presented text does not provide information about how many of a user's errors were corrected by the touch keyboard.

Figure 4 shows an example of user touch input on a soft keyboard. The presented text is "home," but the user touches the keys "h", "o", "m", and "w." A naïve keyboard that does not attempt to correct user imprecision may output "homw," but the keyboard in the figure has successfully corrected the user's input and output the text "home." However, if we compare the presented text to the transcribed text, we do not learn about the successful error correction. If the user's input had been precise (hitting the "h," "o," "m," and "e" keys), the naïve keyboard would have produced the same transcribed text. A Word Score comparing presented text with transcribed text will give the same result for the two scenarios.



presented: home

transcribed: home

Figure 4. A user touches the screen when entering the word "home." The touches are labeled in the order entered. The user imprecisely entered the "e," touching above the "w" key instead.

We thus propose to compare the presented and transcribed text to *baseline* text, which reflects a user's uncorrected keyboard output. The baseline text is generated from the closest key labels on the keyboard to the users' actual touch points. This is a naïve key-detection algorithm that offers no error correction, and literally transcribes the user's touch points. Comparing the *baseline* text to the presented text provides information about a *user's* accuracy; comparing the *baseline* text to the transcribed text provides information about the *keyboard's* ability to correct user errors.

To quantify a touch keyboard's ability to correct user error into correct words, we propose another metric, the *Ratio of Error Reduction* (RER). This metric is defined as the proportion of errors in the baseline string that were fixed by the keyboard. The RER is calculated as follows:

$$RER = \frac{E_{baseline} - E_{transcribed}}{E_{baseline}} \times 100\% \quad (4)$$

where $E_{baseline}$ is the error rate of the baseline text and $E_{transcribed}$ is the error rate of the transcribed text. *RER* is applicable only when $E_{baseline} > 0$. We can use either the MSD or the MWD to express the error rates.

EVALUATING STKS

We used Octopus to evaluate two touch keyboards on Android phones. Since the purpose of the current work was to research and demonstrate an evaluation method and a tool, not to report the relative merit and quality of different commercial products, we anonymously refer to the two keyboards as STK-A and STK-B. The dimensions of the two are identical, but the algorithms that determine keyboard output for given touch input are different. Since STK-B was developed a bit later than STK-A, it was expected that it would perform slightly better than STK-A.

Method

Collecting Typing Data

To evaluate the keyboards with Octopus, we collected text input data from users in a laboratory study we called "Salt." The study was similar to the dataset used in Azenkot and Zhai [4], but with different conditions for different hand postures, and different instructions to encourage the users to type more quickly.

A wizard of Oz keyboard was used in the study (Figure 5), which provided users with only asterisks as feedback when they entered text. After a user finished a phrase, she pressed the "next" button to proceed to the next phrase.

We recruited 40 participants. The average age was 32 (the youngest was 18 and the oldest was 59). Five were left-handed. All had experience with text entry on smartphones. The average level of self-rated proficiency with STKs was 5.5 (SD = 1.2) according to a pre-study questionnaire (1 – 7, 1 = no experience, 7 = expert).

Unlike the study in Azenkot and Zhai [4], which specified a hand posture for every participant, this study allowed the participants to text with two thumbs, one finger, or one thumb on the dominant hand according to their preference. A Galaxy Nexus phone was used throughout the study.

Twenty-four users entered text with two thumbs, 4 with one thumb, and 12 with the index finger.

Figure 5. The WOZ keyboard we used to collect data for evaluating STKs. The keyboard design aims to capture fundamental text entry behavior.

Each participant entered the same set of 50 phrases randomly chosen from the MacKenzie and Soukoreff phrase set [17, 25]. All touch events were logged. Participants were asked to enter text "as naturally and as fast as possible." The first 10 phrases for each user were considered a warm-up and excluded in the dataset. Overall, we collected input for 1,597 phrases with 7,106 words.

Running Octopus

We ran Octopus using the Salt dataset on two Galaxy Nexus phones, the same type of phone used to collect the data. A complete run of Octopus with the Salt data took about 212.8 minutes (about 3.5 hours) on each keyboard.

Results and Discussion

Character and Word Scores

The mean *Word Scores* across the 40 participants were 83.7 (*SD* = 11.0), and 84.7 (*SD* = 10.2). The *character scores* were 94.6 (*SD* = 4.4) and 94.5 (*SD* = 4.6), for STK-A and STK-B, respectively (Figure 6). A paired t-test did not show a significant difference between these two keyboards on either of the two measures, indicating that the overall accuracies of these keyboards were similar.

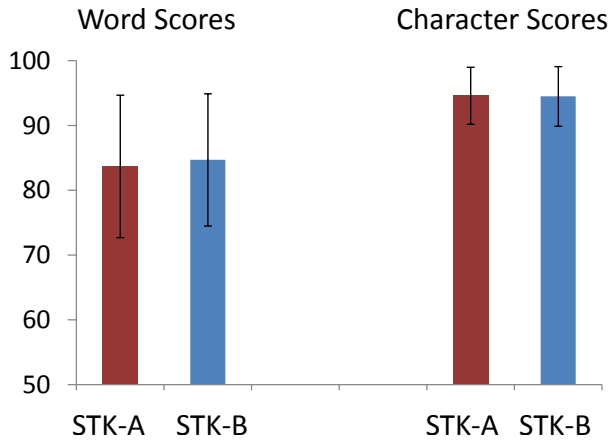


Figure 6. Mean (SD) of Word and Character Scores.

A detailed analysis, however, shows important differences between the two keyboards. The same correct output were generated from the two keyboards on 5,401 (of 7,106) words from the Salt dataset. There were 1,000 cases of touch input in Salt in which both STKs failed to generate correct target words. There were 438 word cases in which STK-B succeeded but STK-A failed. By visually inspecting the output, we discovered that STK-B seemed better at handling missing spaces than STK-A. The top three rows in Table 1 show examples. Conversely, there were 267 cases where STK-B failed but STK-A succeeded, as shown by the bottom three rows of Table 1. STK-B seemed to be more conservative than STK-A in auto-correcting spatial proximity errors.

Output on STK-A	Output on STK-B
pleasevorovife your date	please provide your date
my gagoritevsibjevy	my favorite subject
an offervyoy cannot refus	an offer you cannot refuse
three two one zero	three twp one zero
are you talking to me	are yiy talking to me
hair gel is very greasy	hair gel is very greadu

Table 1. Examples of Salt output from STK-A and STK-B. The phrases on the same row were generated on the same set of touch points.

Here we can see the Octopus *remulation* approach not only generated global metrics, but also afforded inspection of specific error cases. The latter capability enables the kind of micro-level analysis critical for more detailed insights into a keyboard's characteristics.

RER Rates

The word and character scores of baseline text of the Salt dataset were 38.9 (*SD* = 15.3) and 80.6 (*SD* = 8.6) respectively, indicating that users' input was very sloppy. More than 60% of words would have been incorrect if the touch input were decoded with a naïve closest-label keyboard algorithm.

Both STK-A and STK-B keyboards markedly reduced the errors due to imprecise input. As shown in Table 2, the *Ratio of Error Reduction* (RER) was 72.6% and 75.1% for MWD, and were 72.2% and 71.6% for MSD, for STK-A and STK-B, respectively. Approximately 7 out of 10 word errors in the Salt dataset were corrected by each keyboard.

	MWD	*MSD*
STK-A	16.3%	5.4%
STK-B	15.2%	5.5%
Baseline	61.1%	19.4%
RER of STK-A	**72.6%**	**72.2%**
RER of STK-B	**75.1%**	**71.6%**

Table 2. Ratio of Error Reduction (RER) for both keyboards, using the MSD and MWD to measure error.

When a keyboard attempts to correct user errors, a word undergoes a transition between two possible states: *correct* (no errors) and *incorrect* (contains some errors). There are four such transitions: *incorrect* to *correct*, *incorrect* to *incorrect*, *correct* to *incorrect*, and *correct* to *correct*. Table 3 shows examples of the transitions for the presented text "home."

Ideally, all transitions are either *correct* to *correct*, where the keyboard algorithms recognize the input is correct and do not modify it, or *incorrect* to *correct,* where the keyboard algorithms identify "sloppy" input and modify it to the target word. *Correct* to *incorrect* transitions are likely to occur when a user enters a string that is OOV (out of vocabulary). Transitions that move from *Correct* to *Incorrect* have been found in prior work to be disruptive and frustrating to users [9].

	Baseline	*Transcribed*
Incorrect → Correct	homw	Home
Incorrect → Incorrect	hone	Gone
Correct → Incorrect	home	homw
Correct → Correct	home	home

Table 3. Examples of transitions from the baseline to the transcribed text for the presented text "home." A "correct" state indicates the transcribed text is the same as the presented text, and an "in-correct" state indicates they are different.

To deeply understand the keyboard's error correction behavior, we investigate the composition of these four types of transitions (Figure 7). 3,297 words (46.4%) on STK-A and 3,425 words (48.2%) on STK-B keyboards underwent an *incorrect* to *correct* transition, demonstrating strong auto-correction performance for both keyboards. STK-A and STK-B falsely changed 1.5% and 0.8% of words from correct to incorrect. Although small in number, in this very critical category of errors, STK-B made only about half the errors that STK-A did. Table 4 shows examples of successful corrections.

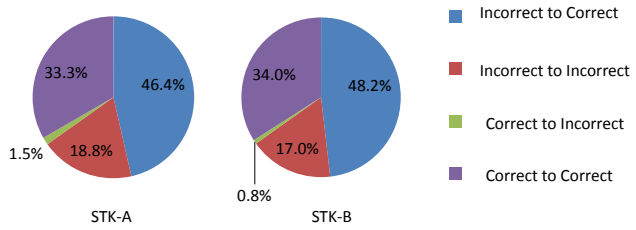


Figure 7. Percentages of Four Types of Transitions on STK-A (left) and STK-B (right) keyboards

Baseline	Corrected output on STK-A and STK-B
he s jst kr everone	he is just like everyone
nofer youcannot refuse	an offer you cannot refuse
pleae orovife our aye	please provide your date

Table 4. Examples of Successful Corrections. Words in red are erroneous words.

EVALUATING WORD-GESTURE KEYBOARDS

In addition to evaluating STKs, we used Octopus to evaluate two SGKs, SGK-A and SGK-B. SGK-B was developed later than SGK-A, so we expected it to perform better.

Method

Collecting Gesture Data

The Salt study described earlier also collected gestures. The experiment balanced the order of gesture and touch data collection. The WOZ keyboard in the study was the same as the one used in touch data collection (720 × 414 pixels). It was identical to SGK-B in dimensions, but slightly different from SGK-A in height (720 × 398 pixels).

The gesture data collected were therefore slightly scaled to match SGK-A when running Octopus.

The same 40 subjects participated in the study. The levels of proficiency of these participants reflected the partial adoption of SGKs: half of the participants were proficient users who used SGKs at least 5 days a week; the other half had never used an SGK.

Unlike STK data collection in which participants freely chose the preferred input finger, hand posture was a two-level, within-subject factor (i.e., thumb or index finger). Each participant gestured a set of 50 phrases by index finger and the same set of phrases by the thumb. The device used in this study was the same as the one used in the typing data collection study. A user's finger gesture trace was shown using a blue stroke, and only minimal output feedback was provided. The current target word was underlined and the previously gestured words were dimmed (Figure 8). Our purpose was again to capture the most natural input behaviors, and to avoid interference from any particular recognition algorithm.

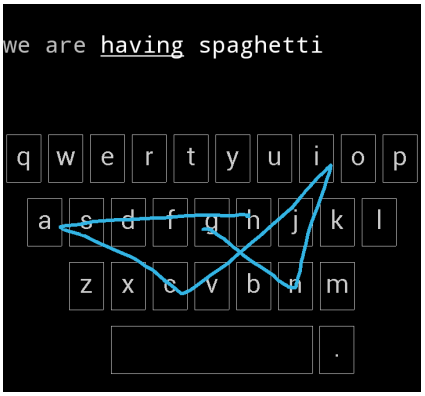


Figure 8. Keyboard Layout for Gesture Data Collection. The participant was gesturing the word *having*.

The first 10 phrases for each posture and each user were treated as a warm-up and excluded from the dataset. In the entire study, we collected data for 40 × 2 (finger postures) × 40 (participants) = 3,200 phrases with 14,235 words.

Running Octopus

We ran Octopus using the Salt gesture data described above on a Galaxy Nexus phone to evaluate SGK-A and SGK-B. The study had two within-subject factors: input finger (thumb and index finger) and keyboard (SGK-A, and SGK-B). A complete run of Octopus took approximately 400 minutes (around 7 hours).

Results and Discussion

Since gesture input does not have *baseline* text, the RER measure is not applicable for evaluating SGKs. We therefore focus on the word and character scores.

An ANOVA showed that both word ($F(1,39) = 155.5$, $p < .001$) and character ($F(1,39) = 134.6$, $p < .001$) scores of SGK-B were significantly higher than those for SGK-A. The mean word scores were 73.0 (SD = 12.2) and 81.9 (SD = 10.0), and the mean character scores were 83.8 (SD = 8.8) and 89.7 (SD = 7.0) for SGK-A and SGK-B respectively (Figure 9). SGK-B generated about 10 more correct words in every 100 gestures than SGK-A.

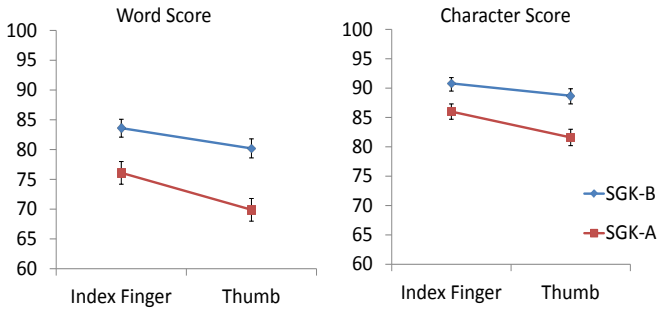


Figure 9. Mean (Std. Error) Word and Character scores by Keyboard and Posture.

The analysis also showed a significant main effect of input finger on word ($F(1, 39) = 14.6$, $p < 0.01$) and character ($F(1, 39) = 11.0$, $p < 0.05$) scores. The participants were more accurate gesturing with the index finger than with the thumb.

ANOVA showed significant interactions for keyboard × input finger on both word ($F(1,39) = 10.0$, $p < .005$), and character scores ($F(1,39) = 12.1$, $p < .005$). As illustrated in Figure 9, SGK-B was especially more accurate than SGK-A in the thumb condition, in which the input was more "sloppy" than in the index-finger condition.

Figure 10 shows the mean word score for each participant. As illustrated, SGK-B is more accurate then SGK-A for 39 out of 40 participants (and the remaining participant had the same score on both). It confirms the finding from the ANOVA analysis: SGK-B is significantly superior to SGK-A.

By comparing the output of both keyboards with presented phrases, we discovered some limitations of each keyboard. These findings could help developers to further improve the algorithms.

One interesting observation of SGK-A's performance was that it tended to mistakenly include unintended letters adjacent to the target letter in the recognition results. As shown in the first three rows of Table 5, SGK-A misrecognized *provide* as *provides, are* as *ate*, and *smart* as *smarty*. Unintended letters *s* (close to *e*), *t* (close to *r*), and *y* (close to *t*) were mistakenly included.

SGK-A Output	SGK-B Output
please provide your date	please provides your date
you are not a jedi yet	you ate not a jedi yet
yes you are very smart	yes you are very smarty
a great dissonance	a great disturbance
you must be hefting old	you must be getting old
a quezon to answer	a question to answer

Table 5. Examples of Output Phrases on SGK-A and SGK-B. Erroneous words are in red.

Unlike SGK-A, one potential problem for SGK-B was its large vocabulary. It contained obscure words that distracted a sloppy gestures from their intended word. For example, it falsely recognized the target words *disturbance* as *dissonance, getting* as *hefting*, and *question* as *quezon* (see the bottom three rows of Table 5).

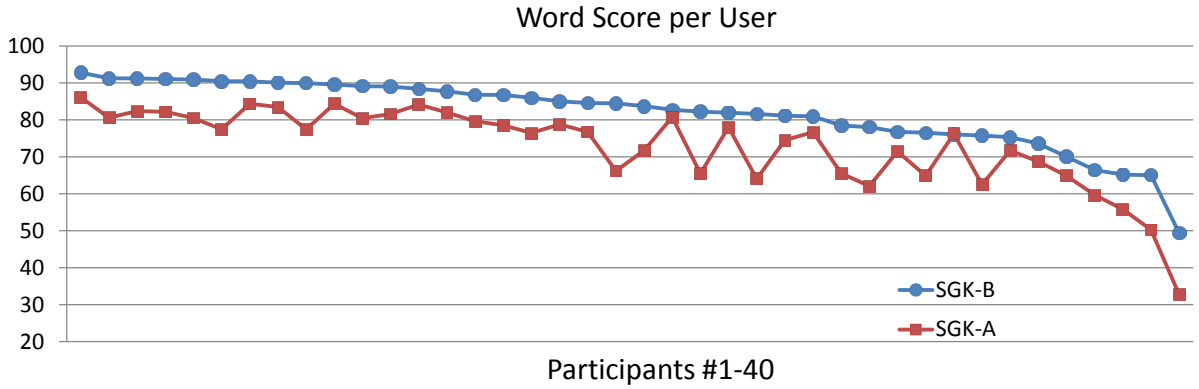


Figure 10. Mean Word Score for Each Participant (Sorted in Descending Order on SGK-B).

LIMITATIONS AND FUTURE WORK

We note again that the performance measures from Octopus Remulation depend on the test dataset. A dataset can be either too easy (nearly perfect input) so all keyboards can give high word scores (a ceiling effect), or too difficult (all input hopelessly sloppy and erroneous) so no keyboard can do well on it (a floor effect). We have collected a dataset, Salt, that is natural and sloppy, so it can discriminate among different keyboards. Indeed our word scores were in 70's and 80's, close neither to the ceiling (100) nor the floor (0). However whether the Salt data closely resembles users' "natural" behavior on real keyboards is debatable. We suspect there may not be a "perfectly natural" behavior when it comes to interacting with real UI technologies. The better the recognition technology gets, the sloppier the user may behave to take advantage of the technology. It will therefore be necessary to collect different datasets that reflect different ranges of user expertise and behavior.

Octopus can be used and enhanced in several ways. We described only two applications of Octopus for evaluating touch and gesture keyboards. With a new set of data, Octopus can be used to evaluate keyboards with different layouts on various devices, including tablets. With appropriate operating system support, the Remulation approach can also be implemented on platforms other than Android, such as Apple's iPhone and Microsoft's Windows Phone, enabling comparison across platforms.

CONCLUSION

We have presented *Remulation*, a novel approach to evaluating keyboard correction and recognition algorithms by replicating prior user study data via real-time simulation. It contributes to the wide spectrum of user interface evaluation methods, ranging from A/B testing in laboratory experiments to model-based prediction. We have also contributed two new metrics, *Word Score* and *Ratio of Error Reduction* (RER), to measure keyboard accuracy at the word level, and to quantify STK error-correction capability.

Based on the *Remulation* approach and new data analysis methods, we have designed and implemented Octopus, a keyboard evaluation tool. Powered by the Salt dataset we collected, we used Octopus to evaluate two smart touch keyboards and two smart gesture keyboards. The results clearly demonstrated the value of the *Remulation* approach, the Octopus tool, and the metrics of *Word Score* and *RER*. For example, Octopus showed that today's STKs can correct over 70% of the word errors that a naïve touchscreen keyboard would produce on the Salt dataset. Octopus also exposed different types of errors made by the tested STKs, even though standard metrics showed that the keyboards performed identically. We have shown that Octopus remulation can also be applied to continuous word gesture-based keyboards, and found that one SGK is much stronger than another, which in turn suggests that this novel input paradigm [29] may progress even further with future research.

ACKNOWLEDGEMENTS

We thank participants in our studies, and the CHI paper reviewers for the insightful comments.

REFERENCES

1. Andrew, C. W., Maier, V., Green, P. (2004). From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. *Proc. of INTERSPEECH.* 4 pages.
2. Android 4.2 http://www.android. com/whatsnew/
3. Android Developers. Tools: Andorid Debug Bridge. Accessed April 13, 2012. http://developer.android.com/guide/developing/tools/adb.html.
4. Azenkot, S. and Zhai, S. (2012). Touch Behavior with Different Postures on Soft Smart Phone Keyboards. *Proc. of MobileHCI'12.* 251-260.
5. Bi, X., Chelba, C., Ouyang, T., Partridge, K., and Zhai, S. (2012). Bimanual gesture keyboard. *Proc. of UIST'12.* 137-146.
6. Damn You, Auto-Correct! http://www.damnyouauto correct.com/. Accessed April 13, 2012.
7. Findlater**,** L**.**, and Wobbrock, J.O. (2012). Personalized input: improving ten-finger touchscreen typing through automatic adaptation. *Proc. of CHI '12*. 815-824.
8. Goodman, J., Venolia, G., Steury, K., and Parker, C. (2002). Language modeling for soft keyboards. *Proc. AAAI '02*, Menlo Park, CA, USA, 419-424.
9. Gunawardana, A., Paek, T., and Meek, C. (2010). Usability guided key-target resizing for soft keyboards. *Proc. IUI '10*. ACM, New York, NY, USA, 111-118.
10. Kristensson, P.O. and Zhai, S. 2004. SHARK: a large vocabulary shorthand writing system for pen-based computers. *Proc. of UIST '04*. ACM Press: 43-52.
11. Lewis, J. R. (1992). *Typing-key layouts for single-finger or stylus input: initial user preference and performance* (Technical Report No. 54729). Boca Raton, FL: International Business Machines Corporation.
12. Lewis, J. R., Kennedy, P. J., & LaLomia, M. J. (1999). *Development of a Digram-Based Typing Key Layout for Single-Finger/Stylus Input.* Proceedings of The Human Factors and Ergonomics Society 43rd Annual Meeting.
13. Lewis, J. R., LaLomia, M. J., & Kennedy, P. J. (1999). *Evaluation of Typing Key Layouts for Stylus Input.* Proceedings of The Human Factors and Ergonomics Society 43rd Annual Meeting.

14. Lewis, J. R., Potosnak, K. M., & Magyar, R. L. (1997). Keys and Keyboards. In M. G. Helander, T. K. Landauer & P. V. Prabhu (Eds.), *Handbook of human-computer interaction* (2nd ed., pp. 1285-1315). Amsterdam: Elsevier Science.

15. MacKenzie, I. S. and Soukoreff, R. W. (2002). A Model of Two–Thumb Text Entry. In Proceedings of Graphics Interface '02, pp. 117–124.

16. MacKenzie, I. S. and Soukoreff, R. W. (2002). A chracter-level error analysis technique for evaluating text entry methods. In Proceedings of the 2nd Nordic Conference on Human-Computer Interaction (NordiCHI). Arhus, Denmark (Oct. 19–23). 243–24.

17. MacKenzie, I. S., and Soukoreff, R. W. (2003). Phrase sets for evaluating text entry techniques. *Proc. of CHI EA '03*, pp. 754-755. New York: ACM.

18. MacKenzie, I. S., and Tanaka-Ishii, K. (Eds.). (2007). *Text Entry Systems: Mobility, Accessibility, Universality*: Morgan Kaufmann Publishers.

19. MacKenzie, I. S., and Zhang, S. X. (1999) The design and evaluation of a high-performance soft keyboard. Proceedings of the ACM Conference on Human Factors in Computing Systems CHI '99, pp. 25-31. New York: ACM.

20. The New York Times, iPhone Keyboard Secrets. http://pogue.blogs.nytimes.com/2007/06/27/iphone-keyboard-secrets/

21. Rick, J. (2010). Performance optimizations of virtual keyboards for stroke-based text entry on a touch-based tabletop. *ACM UIST,* 77-86.

22. Rudchenko, D., Paek, T., and Badger, E. (2011). Text Text Revolution: A game that improves text entry on mobile touchscreen keyboards. *Proc. Pervasive '2011*.

23. Rumelhart, D. E. and Norman, D. A. (1982), Simulating a Skilled Typist: A Study of Skilled Cognitive-Motor Performance. Cognitive Science, 6: 1–36.

24. Wobbrock, J.O. (2007). Measures of text entry performance. Chapter 3 in I.S. MacKenzie and K. Tanaka-Ishii (eds.), Text Entry Systems: Mobility, Accessibility, Universality. San Francisco: Morgan Kaufmann, pp. 47-74.

25. Wobbrock, J.O. and Myers, B.A. (2006). Analyzing the input stream for character-level errors in unconstrained text entry evaluations. ACM Transactions on Computer-Human Interaction 13 (4), pp. 458-489.

26. Zhai, S., Hunter, M., & Smith, B. A. (2002). Performance optimization of virtual keyboards. *Human-Computer Interaction, 17*(2,3), 89-129.

27. Zhai, S. and Kristensson, P.O. 2003. Shorthand writing on stylus keyboard. *Proc. of CHI '03*. ACM Press: 97-104.

28. Zhai, S. and Kristensson, P.O. Introduction to Shape Writing (2006) *IBM Research Report* RJ10393 (A0611-006), November.

29. Zhai, S. and Kristensson, P.O. (2012). The word-gesture keyboard: reimagining keyboard interaction. *Commun. ACM* 55, 9 (September 2012), 91-101.