Usability Guided Key-Target Resizing for Soft Keyboards

Asela Gunawardana
Microsoft Research
Redmond, WA 98052
aselag@microsoft.com

Tim Paek
Microsoft Research
Redmond, WA 98052
timpaek@microsoft.com

Christopher Meek
Microsoft Research
Redmond, WA 98052
meek@microsoft.com

ABSTRACT
Soft keyboards offer touch-capable mobile and tabletop devices many advantages such as multiple language support and space for larger graphical displays. On the other hand, because soft keyboards lack haptic feedback, users often produce more typing errors. In order to make soft keyboards more robust to noisy input, researchers have developed key-target resizing algorithms, where underlying target areas for keys are dynamically resized based on their probabilities. In this paper, we describe how overly aggressive key-target resizing can sometimes prevent users from typing their desired text, violating basic user expectations about keyboard functionality. We propose an anchored key-target method which aims to provide an input method that is robust to errors while respecting usability principles. In an empirical evaluation, we found that using anchored dynamic key-targets significantly reduce keystroke errors as compared to the state-of-the-art.

Author Keywords
source-channel key-target resizing, language model, touch model

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: User Interfaces—*Input devices and strategies*

General Terms
Experimentation, Human Factors, Performance

INTRODUCTION
Mobile and tabletop devices with touchscreens have become increasingly widespread in the commercial market, such as the Apple iPhone [1] and the Microsoft Surface [2]. These devices often utilize a graphically rendered image of a keyboard, or a *soft keyboard* (see [10] for a survey), for text input. Because soft keyboards have no physical manifestation, more space can be devoted to larger displays, and because they are driven by software, soft keyboards can easily support multiple languages, key layouts, and screen orientations [8]. On the other hand, soft keyboards lack tactile feedback, which enable users to know when they have touched, clicked and slipped away from a key. These physical affordances have been shown to be critical for touch-typing [17]. For mobile devices, smaller key sizes exacerbate the situation. Mobile users not only type slower on a soft keyboard than on a miniature physical keyboard [8], but they also generate more errors and fail to notice them [4].

A number of interesting approaches to mitigating errors on soft keyboards involve making the rendered keys larger or smaller depending on their likelihood [3]. Some approaches even visually highlight keys [3, 15, 14], though some studies [7] report that users could find this distracting. A different kind of approach, which we focus on here, is *key-target resizing*, where underlying target areas for keys, rather than the keys themselves, are dynamically resized based on their probabilities. Notably, the soft keyboard of the popular iPhone, which actually markets key-target resizing as one of its key features [16], uses this approach. For example, in the case of a standard QWERTY layout, touches on parts of the rendered "y" key may return "t" if the previous input was "habi" because "habit" is more probable than "habiy." This makes it seem as if the target area for the "t" key has grown while the hit target for the "y" key has shrunk. In order to estimate the probability of keys for key-target resizing, Goodman et al. [6] introduced a *source-channel approach* which combines a *language model* for predicting the likelihood of different intended key sequences with a *touch model* for predicting the likelihood of different finger touch or pen locations on the screen given the user's intended key. When a touch event occurs, the source-channel approach uses both the touch location and the likelihood of various continuations of the key stream in order to determine the most likely key the user intended to type.

While this approach has been shown to improve typing accuracy [6], we argue that it also makes the behavior of the keyboard less predictable to users. Since the key-targets are dynamic and do not correspond to the rendered keys, it is unclear to users where they should touch the keyboard in order to register their desired keys. In extreme cases, overly aggressive key-target resizing may even cause certain key sequences to be impossible to type. For example, it may be impossible to type the "y" key after "habi," even if the user actually intends to type "habiy" (e.g., as a proper name). This violates basic user expectations about keyboard functionality. It is therefore desirable that key-target resizing be guided by the usability principle that every key can reliably

IUI'10, February 7–10, 2010, Hong Kong, China.

be typed in every context, and that the location that returns each key is intuitive. In this paper, we propose an anchored key-target resizing method that accomplishes this within the source-channel approach, so that soft keyboards can remain robust to errors while respecting usability principles.

This paper consists of three sections. In the first section, we provide a summary of the source-channel approach [6] to key-target resizing. In the second, we detail how we modified this approach to yield anchored dynamic key-targets. In the third section, we empirically evaluate our anchored dynamic key-targets in simulation experiments on mobile soft keyboard data, showing improvements in keystroke error rate over the state-of-the-art. After discussing our results, we conclude with opportunities for future research.

BACKGROUND

Related Research

As mentioned in the introduction, a number of researchers have explored making the language model component of key prediction explicit in the interface by visually highlighting the next likely keys. Al Faraj et al. [3] increased the visual size of the keys of a QWERTY soft keyboard on a ultra-mobile PC and found that users were faster and more accurate. Similarly, Magnien et al. [15] found that by bolding the next likely keys for keyboard layouts unfamiliar to users, showing correctly predicted keys increased speed. However, they did not examine performance on a QWERTY layout. MacKenzie and Zhang [14] also highlighted the next likely keys on a QWERTY soft keyboard with different colors for an eye-typing interface. They found that coloring combined with their gaze fixation algorithm significantly reduced error rates. One nice side effect of visually highlighting the next likely keys is that it can sometimes assist users who are unsure of how to spell a word to type correctly.

Unfortunately, in the above studies, none of the researchers investigated the effect of using the key predictions as just a language model weight for noisy input without visual highlighting. As such, it is difficult to tease apart how much the visual aspect contributed to performance above and beyond what would have been gained by leveraging the language model. For key prediction, the above studies employed either character-level bigrams [3] or prefix-based word completions [15][14]. They did not consider ways of incorporating models of touch or eye-gaze input into estimating the overall likelihood of different keys, as is possible with the source-channel approach.

Because different users can and do generate different touch observations for intended keys, Himberg et al. [7] explored personalizing soft keyboards based on individual touch patterns. For a nine-key numeric layout, they visually adjusted the centroids and borders of each key according to different adaptation schedules. Although they did not measure performance, they assessed qualitative feedback from users and concluded that any adaptation of soft keyboard layouts should be sensitive to user expectations so that it is not distracting. We concur with this sentiment and consider personalization an important future direction.

Finally, various researchers have sought to overcome noisy input on soft keyboards by augmenting them with hardware for vibro-tactile feedback [8][4]. Others have developed alternative keyboard layouts based on Fitt's law and character-level bigrams such as the Metropolis [19] and OPTI [13] layouts.

Source-Channel Key-Target Resizing

We now review the source-channel approach to key-target resizing described in [6]. This approach utilizes a language model that predicts the likelihood of intended key sequences, and a touch model that predicts the likelihood of different touch locations for each intended key. The model is fairly general, and subsumes some other probabilistic models such as that of [7].

Suppose that $l_1, \cdots, l_n$ is a sequence of n touch locations, where each $l \in \mathbb{R}^2$ is an x and y coordinate pair. The task is to output a good estimate $k_1^*, \cdots, k_n^*$ of the user's intended sequence of keys, from a key alphabet $\mathcal{K}$. In this paper, we address the case where there are no insertion or deletion errors, where an unintended touch is spuriously registered or an intended touch is not registered. The most likely intended key sequence given the sequence of touch locations is given by

$$k_1^*, \cdots, k_n^* = \underset{k_1,\cdots,k_n}{\arg\max}\, p(k_1, \cdots, k_n | l_1, \cdots, l_n) \quad (1)$$

Using Bayes' rule,

$$p(k_1, \cdots, k_n | l_1, \cdots, l_n) = \frac{p(k_1, \cdots, k_n) p(l_1, \cdots, l_n | k_1, \cdots, k_n)}{p(l_1, \cdots, l_n)} \quad (2)$$

Since the denominator is a positive constant with respect to $k_1, \cdots, k_n$, it can be ignored in the maximization of equation (1) to yield

$$k_1^*, \cdots, k_n^* = \underset{k_1,\cdots,k_n}{\arg\max}\, p(k_1, \cdots, k_n) p(l_1, \cdots, l_n | k_1, \cdots, k_n) \quad (3)$$

The first term is referred to as the *language probability* while the second term is the *touch probability*.

Using the chain rule we can decompose the language probability as

$$p(k_1, \cdots, k_n) = p(k_1) p(k_2|k_1) \cdots p(k_n | k_1, \cdots, k_{n-1}) \quad (4)$$

Following [6], the language probability is modeled using an N-gram language model which makes the approximation

$$p(k_i | k_1, \cdots, k_{i-1}) \approx p_L(k_i | k_{i-N+1}, \cdots, k_{i-1}) \quad (5)$$

That is, we assume that the probability of a key given all the keys so far is approximated by the probability of that key given only the last $N-1$ keys. Since we do not wish key-target resizing to rule out any key at any time, it is important that the language model be "smooth." That is, it is important

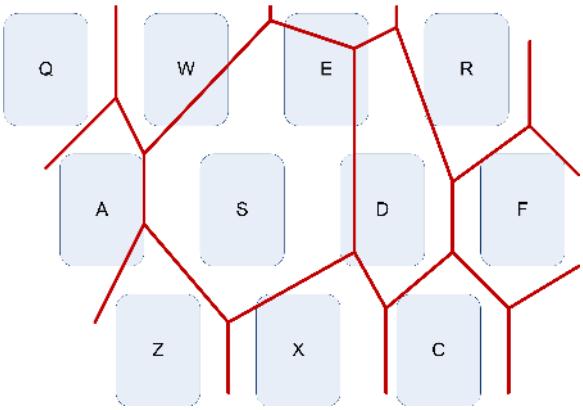


Figure 1. A schematic example where key-target resizing has made it difficult for the user to type the key 'e' because the language model predicts that it is very unlikely compared to the key 's'. The key-target outlines are shown in heavy lines.

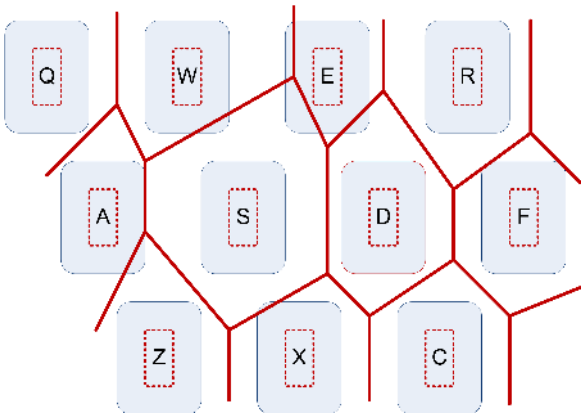


Figure 2. An schematic example where target areas respect each key's anchor. The target area outlines are shown in heavy sold lines, while the anchor outline are shown in broken lines.

that $p_L(k_i = k|k_{i-N+1}, \cdots, k_{i-1}) > 0$ for all k and all histories. As described in [5], we estimate $p_L(k_i|k_{i-N+1}, \cdots, k_{i-1})$ using the interpolated Knesser-Ney technique.

With respect to the touch probability, we again follow [6] and assume that

$$p(l_1, \cdots, l_n|k_1, \cdots, k_n) = p_T(l_1|k_1) \cdots p_T(l_n|k_n) \quad (6)$$

That is, we assume that given the intended key, the touch location for a key press is independent of the intended keys and touch locations for other key presses. The touch probability $p_T(l|k)$ for a single key press will be modeled as a bivariate normal distribution.

For key-target resizing, the keyboard needs to return a key for every keystroke it receives in an online manner, instead of returning a key sequence after receiving an entire sequence of keystrokes, as implied by equation (3). In this case, the most likely intended key is given by

$$k_i^* = \arg\max_{k_i} p_L(k_i|h) p_T(l_i|k_i) \quad (7)$$

where we have denoted the language model history by h.

For each key k, the *key-target* $\mathcal{T}_k(h)$ is the region of the keyboard that returns k, and is given by

$$\mathcal{T}_k(h) = \Big\{ l | p_L(k|h) p_T(l|k) > p_L(k'|h) p_T(l|k'), \forall k' \neq k \Big\} \quad (8)$$

Notice that the key-targets $\mathcal{T}_k(h)$ change with the key history $h = k_{i-N+1}, \cdots, k_{i-1}$.

ANCHORED DYNAMIC KEY-TARGETS

A standard soft keyboard has a predictable user interface–the keys are rendered on the interface and correspond closely to their visual target areas. This guarantees to users that a touch within the rendered boundaries of a key will return that key. Since the target areas on a soft keyboard utilizing key-target resizing change with the key history, the rendered keyboard must either change with the target areas as discussed above, or no longer match the target areas. As was found in [7], users can find dynamic rendering of the keyboard distracting. If the target areas no longer match the rendered keys, users may not always be able to unambiguously predict the response of the interface to different touch locations. In extreme cases, keys which the language model predicts will be very improbable may have target areas that become very small, or even vanish altogether, so that it becomes difficult or impossible for the user to type these keys. For example, Figure 1 shows an instance where the target area of the 's' key has grown at the expense of the 'e' key, leaving the 'e' key's target area small and in an unexpected position.

Such problems will be rare if the language model only made such predictions when the user is truly unlikely to attempt to type these keys. However, even rare occurrences can be quite frustrating to a user. Furthermore, making accurate predictions of the continuations of text is a difficult problem; one that has been the subject of active research for over half a century [18][5]. We therefore present an approach for ensuring that each key has a central *anchor* that is always included in its target area irrespective of the language model history. This provides at least some level of predictability for the user, since touch locations within each key's anchor (typically at the center of the rendered key) are guaranteed to return that key. Figure 2 shows an instance where each key has a central anchor that is always included in its target.

More formally, we wish to associate an anchor $\mathcal{A}_k \subset \mathbb{R}^2$ with each key $k \in \mathcal{K}$ such that $\mathcal{A}_k \subset \mathcal{T}_k(h)$ for all h for any choice of smooth language model or history.

PROPOSITION 1. *$\mathcal{A}_k \subseteq \mathcal{T}_k(h)$ for any choice of smooth language model and history if and only if $p_T(l|k) > 0$ for all $l \in \mathcal{A}_k$ and $p_T(\mathcal{A}_k|k') = 0$ for all $k' \neq k$.*

PROOF. To prove the "if" portion of the result, recall that smoothness means $p_L(k|h) > 0$ for all k and h. Therefore, all $l \in \mathcal{A}_k$ have

$$p_T(l|k) p_L(k|h) > 0$$

and

$$p_T(l|k')p_L(k'|h) = 0$$

for any $k' \neq k$, so that $l \in \mathcal{T}_k(h)$ by the definition of equation (8).

To prove the "only if" portion of the result by contradiction, we consider two cases. In the first case, we suppose there exists an $l \in \mathcal{A}_k$ such that $p_T(l|k) = 0$. In this case, If the user touches location l it cannot be the case that $\mathcal{A}_k \subseteq \mathcal{T}_k(h)$ and we have a contradiction. In the second case, we suppose there exists $k' \in \mathcal{K}$ such that $p_T(\mathcal{A}_k|k') = A > 0$. If $\mathcal{A}_k \subseteq \mathcal{T}_k(h)$ for all histories and smooth language models, we have that

$$p_T(l|k)p_L(k|h) > p_T(l|k')p_L(k'|h)$$

for all $k \in \mathcal{A}_k$ and all choices of language model and history. Integrating both sides over $\mathcal{A}_k$ we get

$$p_T(\mathcal{A}_k|k)p_L(k|h) > p_T(\mathcal{A}_k|k')p_L(k'|h)$$

$$p_T(\mathcal{A}_k|k) > A\frac{p_L(k'|h)}{p_L(k|h)}$$

Since this holds for any choice of language model and history, it must hold when $\frac{p_L(k'|h)}{p_L(k|h)} > \frac{1}{A}$. This implies $p_T(\mathcal{A}_k|k) > 1$, which is the contradiction.

This completes the proof. □

Proposition 1 is an important result because it maintains that the only way to guarantee that an anchor returns its corresponding key irrespective of the language model is to restrict the support of the touch model to disallow the anchor for all other keys. In particular, this means that Gaussian touch models, as described in [6], would need to be restricted in order to guarantee some degree of predictability.

DATA COLLECTION AND EVALUATION

In this section, we describe how we collected mobile soft keyboard data for training and testing our language and touch models. We also discuss the results of simulation experiments we conducted comparing our anchored key-target resizing method against the state-of-the-art approach and a baseline of having no key-target resizing at all.

Data Collection

For both data collection and evaluation, we sought to create a soft keyboard prototype which would allow participants to enter text in a "natural" fashion. If the prototype had no key-target resizing, it might generate too many errors, causing users to change their typing behavior so as to be more deliberate for each keystroke. This would not only upset the naturalness of the data, but we might not generate enough noisy touch input to train a touch model. Therefore, we created a prototype with simulated "ideal" key-target resizing: as long as participants touched keys that were adjacent to their intended key on the QWERTY layout, we registered the correct key. This enabled us to collect data for building a key-target resizing system without having such a system to collect data from. Following traditional text entry tasks [12], we gave users text to copy so that we always knews what keys they were intending to hit.

Procedure

On a touchscreen mobile device, participants were instructed to type in a short phrase that appeared above the soft keyboard as "quickly and as accurately as possible". As participants typed each letter of the phrase, the phrase would subtract that letter. For example, when the user typed "p" for "prevailing winds from the east", the screen would then show "revailing winds from the east". As long as the user touched a key that was adjacent to the one they were supposed to touch, the letter would be subtracted. For example, instead of "p", the user might accidentally hit "o" or "l", both of which constitute adjacent keys on a QWERTY layout. This procedure gave participants a sense that they could type on the soft keyboard in a "natural" fashion, despite its lack of tactile feedback. From follow-up discussions, we found that most participants did not even notice that this occurred and among those who did, they just assumed that the soft keyboard had "awesome intelligence", which is indeed what we were striving to create from their data. Participants finished with the phrase when there were no letters left to subtract.

Note that this procedure allows us to obtain useful distribution data for training touch models because we are effectively learning that when users try to type "p", they sometimes hit "o" with some frequency and "l" with some frequency. Given that we subtract letters only when users touch that letter or an adjacent key, one problem that might occur is that users might type in long sequences of keystrokes that are off by one letter. For example, if the user types "irevailing" instead of "prevailing", we might falsely assume that in addition to hitting "i" for "p", which is not an adjacent key, the user also intended to hit "r" for "p", "e" for "p", and so forth. In order to circumvent this problem, we played a beep sound every time the user touched a key that was neither the expected key nor its adjacent keys.

The touchscreen mobile device we used was a prototype premarket phone with a 4.2 inch resistive screen with 800 x 480 pixels. For comparison, we note that the iPhone sports a 3.5 inch capacitive screen with 480 x 320 pixels.

Stimuli

We presented four types of stimuli phrases. We obtained *standard* phrases from MacKenzie and Soukoreff's phrase set [11], which contains short phrases of English text from 16 to 43 characters whose unigram and bigram frequency correlation with an English corpus is high at $r = 0.954$. Additionally, in order to make sure that participants had a chance to hit every letter on the keyboard, we wrote a script to select the shortest sequences of phrases that covered the entire alphabet from "a" through "z". We used these *standard-training* phrases as training data for our touch models. We treated the rest of the standard phrases as *standard-testing* data for evaluation. Furthermore, we sought to obtain phrases for common mobile tasks. As such, we obtained frequent *urls* sampled from a corpus of web browser logs and search

queries sampled from a corpus of Bing search query logs. We also obtained snippets of *emails* culled from an email corpus, which contained at least one word which was either a proper noun or technical jargon (e.g., "anoo regarding dev budget").

Note that none of the stimuli phrases contained any capitalization, punctuation or other symbols. We did this to avoid having to require users to switch to an alternative keyboard layout (e.g., for selecting symbols), which would also complicate our touch modeling.

Participants received three sets of stimuli with relaxation breaks in between. We used three sets in order to capture three common ways in which users held mobile devices. For each set, participants were asked to type using either two thumbs, one thumb on one hand, or one hand holding the device and another hand for typing. We counter-balanced the order of these three conditions. Each set of stimuli contained 50 phrases consisting of 15 standard-training phrases, 10 standard-testing phrases, 11 email phrases, 7 query phrases, and 7 url phrases. The order of the phrases was randomized.

Participants

We recruited 9 participants (5 males and 4 females) between the ages of 21 and 40 using a professional contracting service. Participants hailed from a wide variety of occupational backgrounds. All participants were compensated for their time. 3 owned touchscreen phones sometime in their life, 3 owned QWERTY phones sometime in their life, and 3 owned 12-key numeric phones only. During recruiting, all participants answered that they were familiar with the QWERTY layout and could type on a normal-size keyboard without frequently looking at the keys.

Training Data and Model Building

We use the *standard-training* phrases, which were specifically chosen to cover all the letter keys of a QWERTY keyboard, to train our touch models. The touch models were full-covariance bivariate Gaussians as described in [6]. They were trained using maximum a posteriori estimation with conjugate priors. The priors for the means were centered at the center of each key. A single shared diagonal covariance matrix was computed for all keys and used to center the prior for the covariance matrix. Models with different equivalent sample sizes were evaluated on the *standard-testing* data and the best model chosen.

The language models used were Kneser-Ney smoothed interpolated key 8-grams (see [6] and [5]). This model was trained using 8.5 million characters of text culled from the USENET. There was no overlap between this text and any of the test phrases. The language model probabilities were exponentially weighted, to balance the dynamic range of the language model and touch model probabilities. In other words, we used $p_L^w(k|h)$ instead of the language model $p_L(k|h)$, as is commonly done in areas such as speech recognition [9]. The weighted language models can be renormalized, but this will have no effect on the maximization of equation (7). The weight w was optimized on the *standard-testing* data.

Key-Target Resizing Evaluation

From our data collection, we obtained logs containing the key participants were supposed to hit as well as their actual touch inputs for that key. Touch inputs were averaged (x, y) pixel coordinates from the touch drivers. With these logs, we are able to conduct simulation experiments examining how we would have performed using different key-target resizing approaches.

Simulation Procedure

The logs contain three types of key outcomes. Using a standard layout in which key boundaries lie halfway between the visible boundaries of each key, the participant could have either 1) typed the expected key, which we denote as a *match*, 2) typed an adjacent key, which we denote as a *fuzzy-match*, or 3) typed a non-adjacent key that was not the expected key, which we denote as a *no-match*. For these outcomes, we used the following procedure for simulating how we would have performed using different key-target resizing approaches.

As we process each keystroke from the first expected key to the last, if the user typed a match, we get one chance to also correctly predict the expected key given the (x, y) touch input. This is because the match typed in by the user gets accepted and the letter is subtracted from the phrase, as described in the data collection procedure. Similarly, for fuzzy-matches, we get only one chance to predict the correct key because adjacent keys get accepted.

On the other hand, if the user has typed a no-match, then the next keystroke will have the same expected key as before since the key would not have been accepted and a beep would have been played. Consecutive sequences of no-matches end only when the user types either a match or a fuzzy-match. For each no-match, we attempt to predict the expected key until we either guess correctly using a key-target resizing approach or we run into a match or fuzzy-match.

For example, suppose the user has typed "iiiprevailing", which includes three nomatches in the beginning. Using the (x, y) coordinates of the first "i", our key-target resizing approach might also incorrectly predict an "i". For the second (x, y) coordinates, however, we might correctly predict a "p", in which case, we do not need to guess on the third character "i" nor the fourth character "p" since we already correctly predicted the expected key.

Evaluation Metrics

At the end of our simulation procedure, we obtain a final input stream, which we can then compare against the expected phrase. For accuracy, we evaluated two metrics. First, we assessed *KeystrokeErrorRate* which measures the degree to which we incorrectly predicted the expected key:

$$\text{KeystrokeErrorRate} = 1 - \frac{|\text{match}|}{|IS|} \tag{9}$$

where $|\text{match}|$ denotes the number of correctly predicted keys, and $|IS|$ denotes the length of the final input stream.

We also assessed *MSDErrorRate*, which measures error rate as a function of the minimum string distance (MSD) between two strings. MSD computes the distance between two strings in terms of the lowest number of edit operations required to convert one string into the other (see [12] for more details). Turned into an error rate measure, MSDErrorRate is calculated as:

$$\text{MSDErrorRate} = \frac{\text{MSD}(T, IS)}{\max(|T|, |IS|)} \quad (10)$$

where T denotes the target expected phrase, and MSD is the minimal edit distance between T and IS.

RESULTS

We first performed a series of experiments to determine the effect of anchor size on KeystrokeErrorRate on the *standard-testing* data set. We then evaluated the best anchor size chosen on the *standard-testing* data set on the *emails*, *queries*, and *urls* data sets to ensure that the improvements we observed generalized to other data sets.

Varying Anchor Size

We experimented with restricting the support of the Gaussian touch models to bigger and bigger rectangles centered at the center of each key. The smallest such rectangles corresponded to the static key-targets, and the rectangles grew in height and width by 2 pixel increments until they reached the center of a neighboring key. Because most keys were 46 pixels wide and 100 pixels high, this occurred when the rectangles were 46 pixels wider than the static targets. Each of these increments corresponded to key targets having anchors that began at 46 pixels by 100 pixels (i.e. that enveloped the entire static target of each key), and then shrank in height and width at 2 pixels per step. Thus, the anchor sizes examined ranged from 0 pixels to 4600 pixels (46×100).

Figure 3 shows the average key error rate on the *standard-testing* data set as a function of anchor size. The best anchor size was 6 pixels by 60 pixels, which corresponded to restricting touch models to rectangles 40 pixels wider and 40 pixels higher than the static key targets.

An error analysis of the static, dynamic, and anchored dynamic systems revealed that anchored key-target resizing corrected 22% of the cases where the static system was correct but the state-of-the-art unanchored dynamic system was incorrect, while maintaining 96% of the corrections the unanchored dynamic system made and making no new errors.

Generalization

Finally, we compared the performance of static key-targets, the state-of-the-art unanchored dynamic key-targets, and the anchored dynamic key-targets on the *standard-testing* set, where we optimized the anchored system as well as the independent *emails*, *queries*, and *urls* sets to ensure that the choices made on the *standard-testing* data set gave good results on these other data sets.

As shown in Figure 4, using the anchored dynamic key-

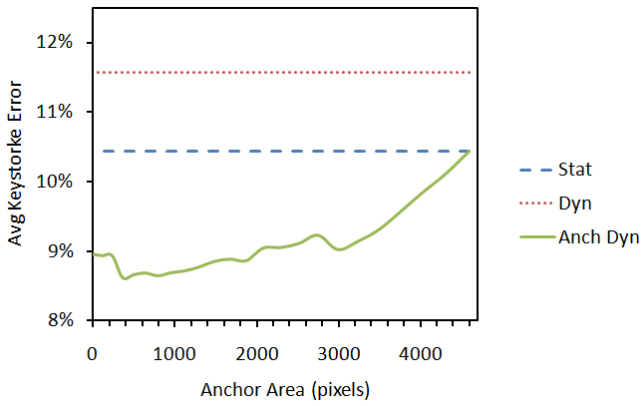


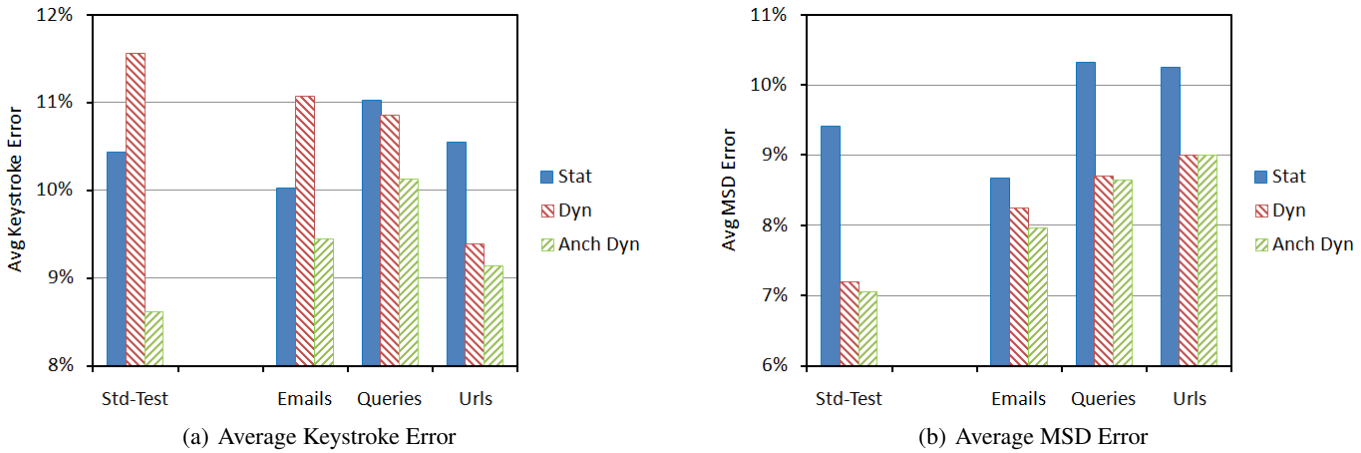
Figure 3. The average key error rate of anchored dynamic key-targets on the *standard-testing* data set as a function of the area of the key anchors (solid line). The performance of the state-of-the-art dynamic key-targets is shown in the dotted line, while the performance of static key-targets is shown in the broken line. At an area of 0, the key-targets are constrained to not extend beyond the center of the neighboring keys. At an anchor area of 4700, the anchors coincide with the static key targets, so that no resizing takes place. The optimal anchor area was 360 pixels.

targets obtains relative improvements of 17%, 6%, 8%, and 13% respectively over the static system on the *standard-testing*, *emails*, *queries*, and *urls* sets with respect to average keystroke error. In contrast, the state-of-the-art unanchored key-target resizing method had an average keystroke error which was 10% *worse* than the static system on the *standard-testing* and *emails* tasks. Note that these difference are all statistically significant at the $p < .01$ level according to McNemar's test.

DISCUSSION

Although the anchored key-target resizing method significantly reduced keystroke errors compared to the static system across the different test sets, the state-of-the-art unanchored system performed worse than the baseline with respect to the *standard-testing* and *emails* tasks. This may be due to a mismatch between the text from these two tasks and the USENET training corpus. Interestingly, the anchored system outperformed the state-of-the-art in al ofl the data sets, but the advantage was not so great with *urls*. Again, this may be explained by the fact that the USENET corpus already contains a fair number of urls. Note that both the anchored and unanchored dynamic systems, which leverage the source-channel approach to key-target resizing, had generally low average keystroke errors.

The anchored and unanchored dynamic systems did not differ much in terms of average MSD error rate. Note that keystroke error rate measures how frequently the keystrokes typed by users do not match what they are intending to type, whereas MSD error rate measures how distant their typed output is from their desired text in terms of required edit operations. Although one could argue that MSD error rate matters most because it represents the amount of work users have to do in order to convert their typed output into the de-



(a) Average Keystroke Error

(b) Average MSD Error

Figure 4. The performance of static key-targets, state-of-the-art dynamic key-targets, and anchored dynamic key-targets on the *standard-testing* set as well as the independent *emails*, *queries*, and *urls* sets.

sired text, this assumes that users type without monitoring their output and only later go back to edit their text. However, previous research [8] suggests that because touchscreen keyboards lack haptic feedback, users are likely to spend more time monitoring their text. Nevertheless, this certainly should be validated in a user study. In fact, we acknowledge that while we have laid the theoretical and empirical grounds for a usability guided key-target resizing method, we still need to verify that real users indeed find the anchors more "usable" in practice. Indeed, we plan to conduct such user studies in the future.

With respect to the data collection we used for our simulation experiments, one problem we encountered was that we obtained far fewer noisy input than we expected. This was due to the fact that we used a mobile device with a large screen and high resolution. We plan to collect more data on a mobile device with a smaller, lower resolution screen (similar to the iPhone) where the need for key-target resizing may be even greater.

CONCLUSION AND FUTURE DIRECTIONS

We have described how state-of-the-art key-target resizing can cause soft keyboards to violate user expectations about keyboard functionality, and how restricting the touch model to yield anchored dynamic key-targets can alleviate this problem. In fact, our theoretical results showed that any source-channel approach that is guaranteed to alleviate this problem must restrict the touch model. We then gave empirical results that showed that anchored dynamic key-targets achieve significant keystroke error reductions as compared to the state-of-the-art.

Our paper described one class of restricted touch model; namely, Gaussians with support restricted to a rectangle around each key. One direction for future research is to explore other restricted touch models, such as models with non-rectangular anchors or non-Gaussian distributions. Distributions could also depend on additional information such as previous keystrokes.

Another area for future work is the adaptation of the touch model, including the anchor sizes, as well as the language model to the user. Furthermore, because users vary in terms of their hand size and consequently their finger touch points, it will be interesting to see if adaptation can improve key-target resizing. In any case, whatever adaptation we pursue will certainly be guided by usability principles, as is the case with our current approach.

REFERENCES

1. Apple iPhone, 2009. http://www.apple.com/iphone/.
2. Microsoft Surface, 2009. http://www.microsoft.com/surface/.
3. K. Al Faraj, M. Mojahid, and N. Vigouroux. Bigkey: A virtual keyboard for mobile devices. In *Proceedings of the 13th International Conference on Human-Computer Interaction. Part III*, pages 3–10, Berlin, Heidelberg, 2009. Springer-Verlag.
4. S. Brewster, F. Chohan, and L. Brown. Tactile feedback for mobile interactions. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 159–162, New York, NY, USA, 2007. ACM.
5. J. Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.
6. J. Goodman, G. Venolia, J. Steury, and C. Parker. Language modeling for soft keyboards. In *AAAI*, 2002.
7. J. Himberg, J. Häkkilä, P. Kangas, and J. Mäntyjärvi. On-line personalization of a touch screen based keyboard. In *IUI*, 2003.
8. E. Hoggan, S. A. Brewster, and J. Johnston. Investigating the effectiveness of tactile feedback for mobile touchscreens. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1573–1582, New York, NY, USA, 2008. ACM.

9. F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1998.

10. M. Klsch and M. Turk. Keyboards without keyboards: A survey of virtual keyboards. Technical report, In: Proceedings of Sensing and Input for Media-centric Systems, 2002.

11. I. S. MacKenzie and R. W. Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 754–755, New York, NY, USA, 2003. ACM.

12. I. S. MacKenzie and K. Tanaka-Ishii. *Text entry systems: Mobility, accessibility, universality*. Morgan Kaufmann, San Francisco, 2007.

13. I. S. MacKenzie and S. X. Zhang. The design and evaluation of a high-performance soft keyboard. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 25–31. ACM, 1999.

14. I. S. MacKenzie and X. Zhang. Eye typing using word and letter prediction and a fixation algorithm. In *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research and applications*, pages 55–58, New York, NY, USA, 2008. ACM.

15. L. Magnien, J. Bouraoui, and N. Vigouroux. Mobile text input with soft keyboards: optimization by means of visual clues. In *Proceedings of Mobile HCI*, pages 337–341, Berlin, Heidelberg, 2004. Springer-Verlag.

16. D. Pogue. iPhone keyboard secrets. *The New York Times*, June 2007.

17. E. Rabin and A. M. Gordon. Tactile feedback contributes to consistency of finger movements during typing. *Experimental Brain Research*, 155:362–369, 2004.

18. C. E. Shannon. Prediction and entropy of printed English. *Bell Sys. Tech. J.*, 30, 1951.

19. S. Zhai, M. Hunter, and B. A. Smith. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 119–128, New York, NY, USA, 2000. ACM.