Error Auto-Correction Mechanisms on Tiny QWERTY Soft Keyboards

Luis A. Leiva[*]
PRHLT, Universitat Politècnica de València
llt@acm.org

Alireza Sahami
hciLab, Universität Stuttgart
{name.surname}@vis.uni-stuttgart.de

Alejandro Catalá
ISSI-DSIC, Universitat Politècnica de València
acatala@dsic.upv.es

Niels Henze
hciLab, Universität Stuttgart
{name.surname}@vis.uni-stuttgart.de

Albrecht Schmidt
hciLab, Universität Stuttgart
{name.surname}@vis.uni-stuttgart.de

[*] Work done while visiting the hciLab in Stuttgart.

This work is part of the Valorization and I+D+i Resources program of VLC/CAMPUS and has been funded by the Spanish MECD as part of the International Excellence Campus program. This work is also supported by the Spanish MINECO (TIN2014-37475 and TIN2010-20488) and the GVA VALi+d program (APOSTD/2013/013).

CHI 2015, April 18–23, 2015, Seoul, Republic of Korea.
Workshop on Text entry on the edge.

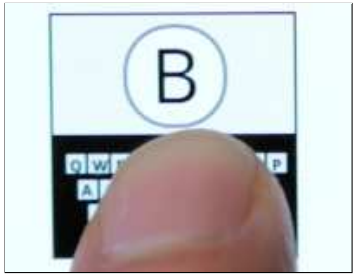
Abstract
Wearable devices (e.g., smartwatches, smartglasses, and digital jewelry) featuring a touchscreen are becoming widely available to consumers. On these devices, entering text with a qwerty soft keyboard is troublesome mainly because of the very small screen space. We investigate 3 simple mechanisms to auto-correct typing errors as they go, at the word level: spell checker with n-best lists, language models, and a combination of both. These mechanisms have potential for wearable devices, which have limited autonomy and limited computing capabilities, even when paired to a smartphone.

Author Keywords
Text Entry; Small Screens; Small Devices; QWERTY

ACM Classification Keywords
H.5.2 [**User Interfaces**]: Prototyping; Screen design

Introduction and Related Work
With the ongoing breakthrough of wearables, such as smartwatches or digital jewelry, text entry on devices with tiny touchscreens (1" wide or less) becomes a relevant and timely topic for HCI. While many approaches have been proposed to enter text on very small devices, every technique or keyboard layout based on a touchscreen has to compete with qwerty [6].

(a) ZoomBoard



(b) Callout

(c) ZShift

Figure 1: Our qwerty soft keyboard prototypes on a 1" screen.

For a long time, spell checkers [4] and character probabilities [2, 7] have been used to correct typing errors on mobile devices. Other approaches have considered typing databases [1] and language models [3], even combined with models of pen/touch placement. More recently, researchers have explored optimization techniques to predict ambiguous text entry on smartwatches [5] and tablets [9]. Among these, we wondered which ones would be suitable for use as error auto-correction mechanisms on very small devices.

Until very recently, ZoomBoard [8] was the only qwerty soft keyboard tailored to diminutive touchscreens. Then, we showed that there are other feasible (qwerty-based) alternatives [6]. In the following we describe the keyboard prototypes that we have studied in previous work.

ZoomBoard Keyboard: To increase the accuracy with which a key can be acquired, instead of immediate selection, the keyboard zooms in (Figure 1a). Then, the user can enter a character with an additional tap. Afterward, the keyboard goes back to the initial zoom level.

Callout Keyboard: Inspired by the soft keyboards used on current smartphones, when the user touches a key, a callout showing the character is created in a non-occluded location (Figure 1b). The user can refine the key to be entered by slightly moving the finger on the keyboard, and then enter the character by lifting up the finger.

ZShift Keyboard: The Callout keyboard has the drawback that once the finger has landed on the screen, it occludes most (if not all) of the keyboard. Based on the Shift pointing technique [10], ZShift creates a callout showing a *zoomed* copy of the occluded screen area (Figure 1c).

Study

Twenty participants submitted 888 phrases (copy-text tasks) with our keyboard prototypes [6]. Participants were allowed to correct mistakes as they went, however we observed that 160 phrases had one or more transcription errors, either because of typos (*Typo*, 110 cases), because a different word—although grammatically correct—was entered (*Diff*, 69), or because fewer words (*Less*, 31) or more words (*More*, 15) than the reference phrase were entered. Figure 2 shows a histogram of the error distribution on a per-phrase basis. Table 1 shows some examples of the errors committed by the users.

Error type	Example (reference text above transcribed phrase)
Typo	`do not lie in court or else` `do not lie in coiurt or else`
Diff	`a yard is almost as a meter` `a yard is almost like a meter`
Less	`consequences of a wrong turn` `consequences of _ wrong turn`
More	`keep receipts for all your expenses` `keep receipts for all of your expenses`

Table 1: Examples of transcription errors (underlined).

Since typos were the most common typing errors, we studied 3 approaches to automatically amend these, with the aim to produce the correct transcriptions: spell checker with n-best lists, language models, and a combination of both. These techniques have potential for wearable devices, as these devices have limited autonomy and limited computing capabilities, even when paired to a smartphone. Thus, we should aim for simple, tried-and-true approaches.

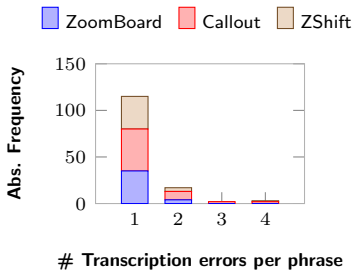


Figure 2: Histogram of phrases containing transcription errors. As observed, all keyboard prototypes achieved similar performance, Callout performing slightly worse.

Spell Checker

We used GNU Aspell,[1] the open source spell checker. Aspell suggests possible replacements for each misspelled word, by looking for *soundslikes* within two edit distance apart of the misspelled word. Then, a typo was replaced by the n-best Aspell suggestion, $n \in \{1, \ldots, 5\}$.

As shown in Figure 3, increasing the n-best lists beyond 3 items did not improve the results substantially. Concretely, using 3-best lists, 71 typos (64.5%) could be transformed into the words that had to be transcribed, amending 54 phrases and thus resulting in an overall improvement of 33%. This suggests that, considering the very limited screen space of the devices we are dealing with, at most 3-best lists should be displayed to the user at a time.

Language Models

We built a bigram language model using the NLTK toolkit.[2] The bigram model approximates the probability of a word given all the previous words by the conditional probability of the preceding word. In order to set an upper bound for this technique, the language model was trained with the same phrase set used as input stimuli in our study, and was tested against the phrases that had transcription errors. Unsurprisingly, this "perfect" language model turned out to be far superior to the other alternatives, see dashed lines Figure 3, although not *all* transcription errors could be fixed because the non-*Typo* cases made it difficult to estimate the likelihood of the entered words with regard to the reference words.

We repeated the same experiment with a small but general-purpose training dataset (the Brown corpus, 57K sentences, 50K unique words), in order to set a lower bound for this technique. We observed that 26% of the erroneous words could be corrected, resulting in 24% of fixed phrases. Notice that this outcome is comparable to using Aspell with 1-best lists.

Spell Checker + Language Model

Aspell results can be further improved by reordering the spelling suggestions according to a probability estimator and suggesting thus always the highest-probability correction. We used the "perfect" language model as said probability estimator, shown in Figure 3 as solid lines, and observed that it performed slightly better than Aspell with 5-best lists. This is interesting for devices where n-best lists are not an option, e.g. when there is no space to display word suggestions onscreen. Anecdotally, the small language model provided the same results as Aspell with 1-best lists.

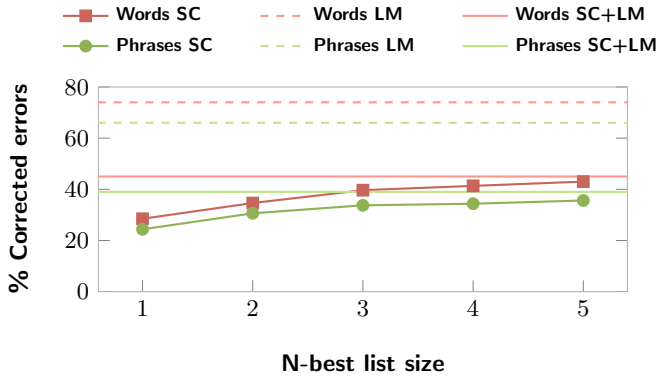


Figure 3: Error auto-correction of words (red) and, as a result, phrases (green). We compare n-best lists of a spell checker (SC) against a "perfect" bigram language model (LM) and a spell checker plus language model (SC+LM).

[1] `http://aspell.net/`
[2] `http://nltk.org/`

Discussion and Future Work

The best results were achieved with a "perfect" language model, trained on the phrases used as input stimuli. However, for obvious reasons this cannot be extrapolated to real-world scenarios. Instead, we observed that most of the typos committed by the users could be auto-corrected using Aspell with 3-best lists. If n-best lists is not an option, e.g. because of the limited space onscreen to display word suggestions, then the combination of Aspell and a language model would allow for similar, competitive results.

As a result of these experiments, we have incorporated error auto-correction to our prototypes. Whenever a space is entered, the last typed word is checked against Aspell. By default, all misspelled words are automatically corrected as shown in Figure 4b. However, the user may want to supervise all errors manually, so automatic corrections can be disabled, in which case the misspelled words are highlighted in red after entering a space, as shown in Figure 4a. Then the cursor automatically goes back one character, to allow the user amend the error more easily. (Actually this behavior is customizable.)

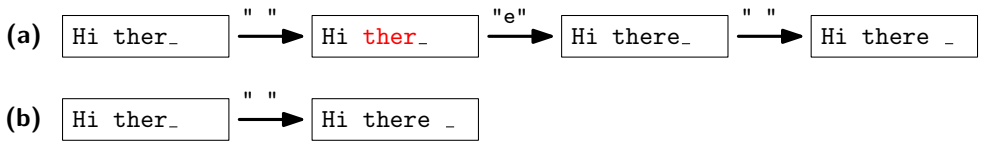


Figure 4: Two error correcting interfaces have been implemented in our prototypes. (a) Error highlighting: aimed at manually correcting misspelled words. (b) Automatic correction of misspelled words.

These error auto-correction techniques were developed as a web service, so that further enhancements can be easily incorporated. These are in fact our current plans for future work. Our prototypes are open source, so that anyone can contribute to improving them or build alternatives by reusing parts of the code.[3]

References

[1] Clawson, J., Lyons, K., Rudnick, A., Jr., R. A. I., and Starner, T. Automatic whiteout++: Correcting mini-QWERTY typing errors using keypress timing. *Proc. CHI* (2008).

[2] Golding, A. R., and Schabes, Y. Combining trigram-based and feature-based methods for context-sensitive spelling correction. *Proc. ACL* (1996).

[3] Goodman, J., Venolia, G., Steury, K., and Parker., C. Language modeling for soft keyboards. Tech. Rep. MSR-TR-2001-118, Microsoft Research, 2001.

[4] Hodge, V. J., and Austin, J. A comparison of standard spell checking algorithms and a novel binary neural approach. *IEEE TKDE 15*, 5 (2003).

[5] Komninos, A., and Dunlop, M. Text input on a smart watch. *Pervasive Computing 13*, 4 (2014).

[6] Leiva, L. A., Sahami, A., Catalá, A., Henze, N., and Schmidt, A. Text entry on tiny QWERTY soft keyboards. *Proc. CHI* (2015).

[7] MacKenzie, I. S., Kober, H., Smith, D., Jones, T., and Skepner, E. LetterWise: prefix-based disambiguation for mobile text input. *Proc. UIST* (2001).

[8] Oney, S., Harrison, C., Ogan, A., and Wiese, J. ZoomBoard: a diminutive QWERTY soft keyboard using iterative zooming for ultra-small devices. *Proc. CHI* (2013).

[9] Oulasvirta, A., Reichel, A., Li, W., Zhang, Y., Bachynskyi, M., Vertanen, K., and Kristensson, P. O. Improving two-thumb text entry on touchscreen devices. *Proc. CHI* (2013).

[10] Vogel, D., and Baudisch, P. Shift: A technique for operating pen-based interfaces using touch. *Proc. CHI* (2007).

[3] `http://personales.upv.es/luileito/tinyqwerty/`