Automatic Whiteout++: Correcting Mini-QWERTY Typing Errors Using Keypress Timing

James Clawson, Kent Lyons†, Alex Rudnick, Robert A. Iannucci Jr., and Thad Starner
College of Computing and GVU Center
Georgia Institute of Technology
Atlanta, GA 30332
{jamer,alexr,thad}@cc.gatech.edu, riannucc@calpoly.edu
†Intel Research
2200 Mission College Blvd.
Santa Clara, CA 95054
kent.lyons@intel.com

ABSTRACT
By analyzing features of users' typing, Automatic Whiteout++ detects and corrects up to 32.37% of the errors made by typists while using a mini–QWERTY (RIM Blackberry style) keyboard. The system targets "off-by-one" errors where the user accidentally presses a key adjacent to the one intended. Using a database of typing from longitudinal tests on two different keyboards in a variety of contexts, we show that the system generalizes well across users, model of keyboard, user expertise, and keyboard visibility conditions. Since a goal of Automatic Whiteout++ is to embed it in the firmware of mini-QWERTY keyboards, it does not rely on a dictionary. This feature enables the system to correct errors mid-word instead of applying a correction after the word has been typed. Though we do not use a dictionary, we do examine the effect of varying levels of language context in the system's ability to detect and correct erroneous keypresses.

Author Keywords
Keyboard Input, Error Correction, Mobile Phones, Mini–QWERTY, Mobile Text Entry.

ACM Classification Keywords
H.5.2 :: User Interfaces, Input devices and strategies

INTRODUCTION
The mini–QWERTY keyboard is a mobile two–handed keyboard that is built into many mobile devices such as mobile phones and personal digital assistants. The mini–QWERTY keyboard often takes the place of the mobile phone keypad on mobile phones such as the RIM Blackberry, the Palm Treo, or T-Mobile Sidekick (see Figure 1). The keyboard has the same layout as a full desktop QWERTY keyboard, complete with a spacebar, delete key, return key, and other non-letter keys (arrow keys, shift, etc.).

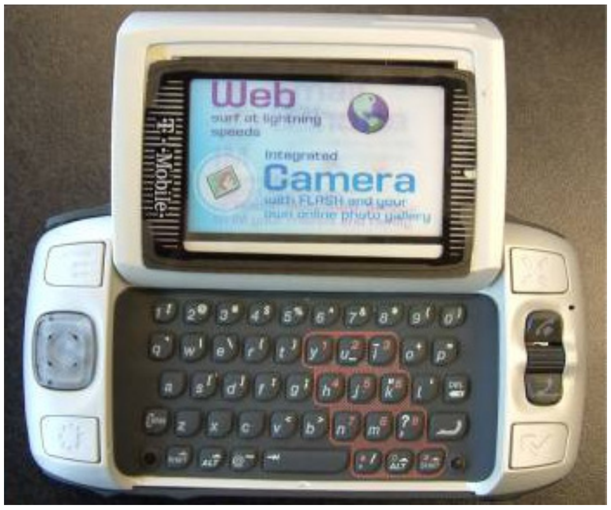


Figure 1. Commercial mobile phones with mini–QWERTY keyboards such as the Danger/T–Mobile Sidekick

Though there has been little work published on mini–QWERTY keyboards [1–3], there exists a canon of research in the area of mobile text input. Mobile text input systems employ either physical, button–based, input systems (e.g. mini–QWERTY keyboards, phone keypads, half–QWERTY keyboards [14], Twiddlers, etc.) or software, touchscreen–based systems (e.g. on–screen keyboards like the iPhone or other pen–stroke based systems). This paper focuses on the former.

Physical keyboards on mobile phones have decreased in size though the size of the average human hand remains constant. These small form–factor keyboards can be accommodated in two ways: make the keys very small, like on mini–QWERTY keyboards, or remove the one–to–one mapping between keys and characters. Much recent research has focused on the latter approach evaluating existing systems [7, 10, 11, 19, 20], designing and evaluating new systems to increase input rates and accuracies [11, 18, 19], or building models or taxonomies to further explain current practices [12, 15, 20].

Mini–QWERTY keyboard typists typically employ two–thumbs when operating a mobile device. Mini–QWERTY keyboards have the same one–to–one key–to–letter ratio as seen on a full desktop QWERTY keyboard. In order to fit so many keys into the space normally occupied by the twelve keys of a mobile phone keypad, the letter keys need to be very small and densely packed together on the mobile device. It is not uncommon for this design to result in keyboards that contain keys with a surface area of less than 25 mm^2 and inter–key spacings of less than two millimeters. The keys are significantly smaller than a user's thumbs which results in difficulty of use. The user's thumbs occlude visibility of the keys, introducing ambiguity as to which key was actually pressed. Further, Fitts' Law implies that typing accuracy decreases as typing speed increases due to the relationship between target size, movement speed, and accuracy [17]. Taken together, these effects combine to lead to errors in mini-QWERTY keyboard text input where a user's thumb presses multiple keys at the same time (usually pressing the key either to the left or the right of the intended key) or presses the intended key twice [2].

We previously performed an analysis of errors that occur when people type on mini–QWERTY keyboards [2] and found that off–by–one errors account for approximately 45% of the total errors committed. Off–by–one errors consist up of row substitution and row insertion errors. A row error occurs when the unintended key press (the error) occurs in the same row on the keyboard as the intended key press[1]. Off–by–one errors are:

- **Row substitution error** – occurs when the user unintentionally presses a key directly to the left or to the right of the intended key (e.g. a user types "ca**y**" or "ca**r**" when she intended to type the word "cat").
- **Key repeat error** – occurs when a user unintentionally presses the same key twice (e.g. the user types "cat**t**" when she intended to type "cat").
- **Roll–on insertion error** – occurs when a character is inserted before the intended character (e.g. the user types "ca**r**t" when she intended to type "cat").
- **Roll–off insertion error** – occurs when the character is inserted after the intended character (e.g. the user types "cat**r**" when she intended to type "cat").

Noticing that these types of errors occur frequently inspired us to attempt to employ machine learning techniques to automatically detect and correct these errors based on features of the users' typing.

The previous version of the system, Automatic Whiteout [4], was only able to detect and correct insertion errors (roll–on, roll–off, and key repeats) that occur in expert typing data. It employed decision trees, a machine learning technique, to detect errors by recognizing patterns in certain features of the user's typing. Having identified an error the algorithm corrected the error by simply deleting the errorful character. Theoretically the initial system was able to detect and correct 26.41% of the expert typing errors on a single keyboard.

[1]Column errors occur rarely in mini–QWERTY keyboard text input and as such are not included in our definition of off–by–one errors

AUTOMATIC WHITEOUT++

Automatic Whiteout++ extends our previous algorithm with better features for roll–on, roll–off and key repeat errors. Using these new features, we compare Automatic Whiteout++ to Automatic Whiteout on our expert typing data. Additionally, Automatic Whiteout++ allows for the correction of off–by–one substitution errors. Furthermore, we demonstrate how Automatic Whiteout++ could be incorporated into a mobile device by showing that it is generalizable. Specifically, in this paper we demonstrate that the algorithm can generalize to different levels of user expertise, to different models of keyboards, and to typists inputting text in conditions of limited feedback. Finally, we evaluate the effect of the correction on overall keystroke accuracy and discuss how our algorithm can be employed to improve mobile text input on mini–QWERTY keyboards with the goal of correcting errors before they are noticed by the user.

Automatic Whiteout++ incorporates more features than its predecessor (82 vs. 36). Many of these new features take advantage of simple language features, specifically bi–letter and tri–letter frequencies. While Automatic Whiteout++ does not include a dictionary, we do include letter probability tables based on a large plain–text corpus. We have also allowed the algorithm to look at subsequent keypresses (in addition to prior keypresses) when evaluating a potential error. We call the number of additional keys that the algorithm evaluates first- and second-order contexts. In all our tests, we only use symmetrical contexts (e.g. the first order context is one keystroke in the future as well as one keystroke in the past). In the section Generalization Across Corpora, we explore how context improves system performance. In addition to these new features, we also use the features from the previous system such as the keys pressed, the timing information between past and subsequent keystrokes around the letter in question, a letter's frequency in English, and the physical relationship between keystrokes (whether the keys involved are located physically adjacent to each other horizontally).

While the decision trees can become quite complex, a simplified example of how Automatic Whiteout++ classifies roll–off insertion errors is illustrative (see Figure 2). The roll–off classifier first determines if the key of the letter it is inspecting (in this case the letter **"Y"**) is located on the keyboard either one key to the left or to the right of the key of the previous letter (in this case, the letter **"T"**). Next it examines if the time between the previous keystroke and the current keystroke is less than or equal to a threshold (in this case 47 milliseconds). In our testing below, this timing information is the key to correcting errors without mistakes. Finally, Automatic Whiteout++ compares the probability of the current key to the probability of the previous key given each key combination's specific tri–letter frequencies and then classifies the current key as a roll–off insertion error

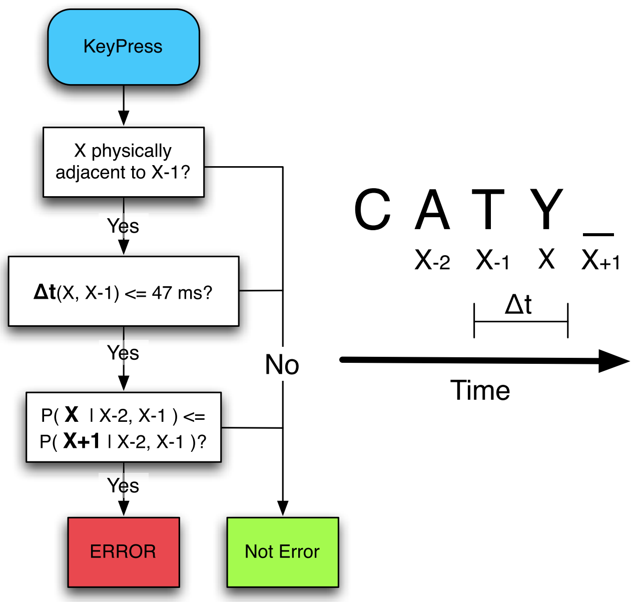


Figure 2. Example Roll–Off Decision Tree.

according to these probabilities. In this case, the three letter combination **"CAT"** occurs much more frequently in English that the three letter combination **"CAY"**. Similar trees are learned for detecting the other types of errors as well. The final Automatic Whiteout++ system tests each keystroke as a key repeat, roll–on, roll–off, and substitution error sequentially, stopping the process and correcting the keystroke if any test returns a positive result.

The correction of roll–on, roll–off, and key repeat insertion errors is relatively simple. The system deletes the offending key stroke thus removing the insertion. Substitution errors, however, require more information to correct. Letter frequency, bi–letter frequency, and tri–letter frequency are used to help correct off–by–one substitution errors. When Automatic Whiteout++ determines that a substitution error has happened, it compares the letters to the right and left of the key typed and selects the most probable one. For example, if the user types **"t h r"** and the system determines that a substitution error has occurred, the possible alternatives are **"t h t"** or **"t h e"**. Since **"the"** is more likely than **"tht"** based on the tri–letter frequencies, Automatic Whiteout++ replaces the **"r"** with an **"e"**. A similar technique for letter disambiguation was used by Goodman et al. [6] previously.

EXPERIMENTAL DATA

Our data set is the output of two longitudinal studies that investigate mini–QWERTY keyboard use [1, 3] (see Table 1). Fourteen participants who had no prior experience with mini-QWERTY keyboard typing participated in the original study [1]. The participants were randomly divided into two subject groups, each of which was assigned to use one of two different keyboard models (see Figure 3). The participants used the assigned keyboard throughout the study. All fourteen participants completed twenty 20–minute typing

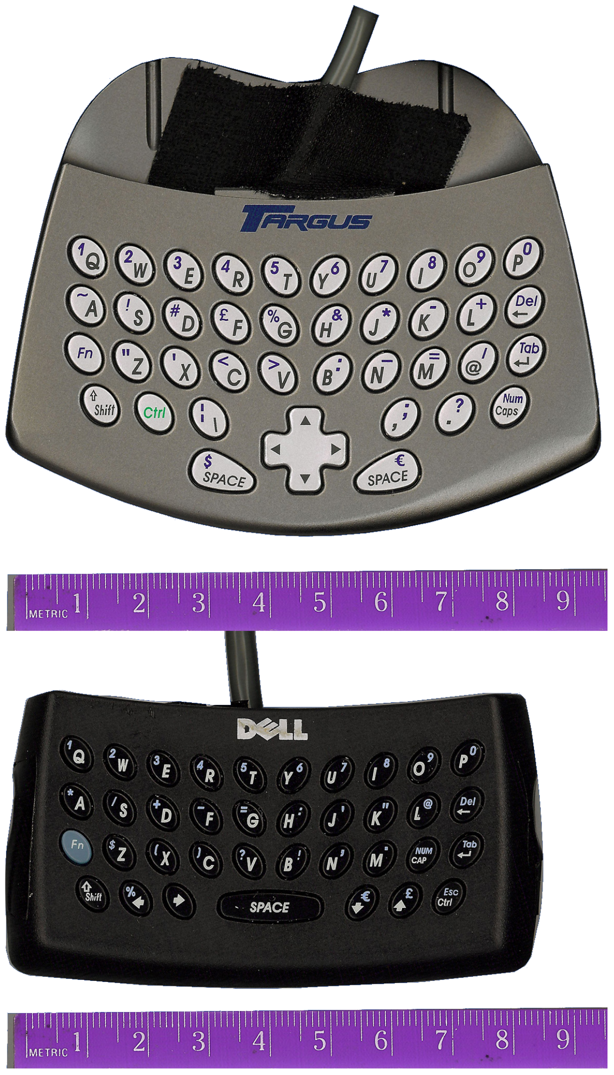
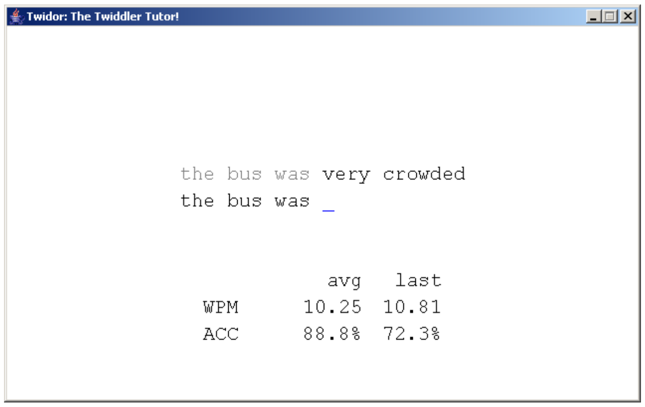


Figure 3. The Targus (top) and Dell (bottom) mini–QWERTY keyboards used in both studies (rulers are indicating centimeters).

sessions for a total of 400 minutes of typing. Eight subjects continued to the "blind" study (see below).

We employed the Twidor software package (used in our series of studies on the Twiddler chording keyboard [9]), and adapted it to accept data from our modified keyboards. Phrases were served to participants in blocks of ten. Participants typed as many blocks of ten phrases as they could during a single twenty minute session. The phrases were taken from a set of 500 phrases designed by MacKenzie and Soukoreff for use in mobile text entry studies [13]. The canonical set was altered to use American English spellings as this study took place in the United States. Additionally, we altered the set so that there was no punctuation or capitalization in any of the phrases in the set. Twidor displays a phrase to the user, the user inputs the phrase and Twidor displays the text produced by the user. Twidor records the user's words per minute (WPM) and accuracy (ACC) and displays both the WPM and the ACC of the input


Twidor: The Twiddler Tutor!

```
the bus was very crowded
the bus was _

          avg   last
WPM     10.25  10.81
ACC     88.8%  72.3%
```

Figure 4. The Twidor experimental software used in both the original and the blind studies.

	Original Study	Blind Study
Date	Fall 2004	Spring 2005
Participants	14	8
Expertise	Novice	Expert
Sessions	20	5
Conditions	2	6
Phrases Typed	33,947	8393
Keystrokes Typed	1,012,236	249,555
Total Phrases Typed	42,340	
Total Keystrokes Typed	1,261,791	

Table 1. The complete mini-QWERTY data set.

phrase to the user (see Figure 4). Twidor also displays a user's average WPM and ACC calculated over the course of the session as well.

In the original study, the participants typed 33,945 phrases across all sessions, encompassing over 950,000 individual characters. Participants were compensated proportional to their typing rate and accuracy over the entire session: \$0.125 $\times$ wpm $\times$ accuracy, with a \$4 minimum for each twenty minute session. Averaged over both keyboards, participants had a mean first session typing rate of 31.72 wpm. At the end of session twenty (400 minutes of typing) the participants had a mean typing rate of 60.03 wpm. The average accuracy rate (as measured using MacKenzie and Soukoreff's Total Error Metric [16]) for session one was 93.88% and gradually decreased to 91.68% by session twenty.

Previously, we investigated participants' ability to input text with limited visual feedback from both the display and the keyboard [3]. In the text input community, studies that examine the entering of text in visually occluded conditions are known as "blind" text input studies. Therefore, we refer to this study as the blind study (see Table 1). When mobile, users must split their attention between the environment and the device with which they are interacting. To simulate this notion of partial attention being paid to the device, we designed a study to investigate typing in conditions of limited visual feedback. Previously we had found that users can effectively type in such "blind" conditions with the Twiddler one–handed keyboard [8]. In the blind study we examined blind typing on mini–QWERTY keyboards in which eight expert mini–QWERTY typists participated in 5 typing sessions. The expert subjects for this study had all previously participated in the original study and used the same keyboard in the blind study that they had learned earlier. Unlike the original study, each session now consisted of three twenty–minute typing conditions.

Data Set	phrases	key presses	errors	OBOs	OBO %
Expert Dell	4,480	64,482	2,988	1,825	61.08%
All Targus	16,407	246,966	8,656	4,983	57.56%
All Dell	15,657	272,230	9,748	6,045	62.01%
Blind Targus	3,266	30,187	2,795	2,072	74.13%
Blind Dell	3,516	29,873	3,287	2,527	76.88%

Table 2. The sampled data sets used for all training and testing of Automatic Whiteout++.

In the first condition, the "normal" condition, the participants had full visual access to both the keyboard and the display. This condition is the same condition that was used in the original study. In the second condition, "hands blind," we obstructed the view of the keyboard by making the participants type with their hands under a desk. Though they could not see their hands, the participants were able to view the display in the same manner as when typing in the normal condition. The final "fully blind" condition not only obstructed the view of the keyboard by making the participants type with their hands under the desk but also reduced visual feedback from the display. In this condition the participant could see the phrase to type but not their actual output. Instead, a cursor displayed the participants' location within the phrase as they typed, but no actual characters were shown until the participant indicated the completion of the phrase by pressing the enter key. At that time, the participant's output was displayed, and the participant could then re–calibrate the position of her hands on the keyboard if necessary.

The 8 participants typed 8,393 phrases across all sessions for a total of 249,555 individual key presses. In contrast to our Twiddler work, we found that in the visually impaired conditions, typing rates and accuracies suffer, never reaching the non–blind rates. Averaged over both keyboards in the blind mini–QWERTY conditions, our participants had a mean first session typing rate of 38.45 wpm. At the end of session five (200 minutes of typing) the participants had a mean typing rate of 45.85 wpm. The average accuracy rate for session one was 79.95% and gradually increased to 85.60% by session five.

Combining both studies we collected 42,340 phrases and 1,261,791 key presses. The data set discussed in this paper is available for public use and can be found at `http://www.cc.gatech.edu/~jamer/mq/data`.

Sampling the Experimental Data

We analyzed the data from all sessions of both data sets and identified each character typed as either correct or as an error. If a phrase contained an error, the characters up to and including the error were kept but the characters that occurred after the initial error were discarded. Truncating the phrase in this manner avoids errors that may have cascaded as an artifact of the data collection. Specifically, Twidor highlights errors as the user enters them. Providing users with visual feedback that indicates when they make mistakes potentially distracts the user, increasing her cognitive load and forcing her to make a decision about whether or not to correct the error. This disruption in the participant's natural behavior potentially effects performance, hence the truncation after the initial error. If the initial error is one of the first two characters in the phrase, the entire phrase is discarded. Additionally, all of the session in which participants entered text in the "normal condition" were removed from the blind study and are not used in our analysis. Sampling our data set reduces the number of phrases and key strokes typed to 30,896 and 449,032 respectively. The sampled set contains 20,879 total errors and 13,401 off–by–one errors.

Data Sets: Dell complete, Dell expert, Targus complete, and blind

The experimental data set was further segmented into four sets for training and testing purposes: Dell complete, Dell expert, Targus complete, and blind (see Table 2 for the distribution of data across the various sets). We analyzed the data for all twenty typing sessions for the Dell keyboard (Figure 3 bottom). The complete set of Dell data contain 15,657 phrases, 272,230 key presses, 9,748 errors, and 6,045 off–by–one errors.

By the time participants began the 16th typing session in the original study they were considered to be expert typists (their learning curves had flattened). We analyzed the data for the last five typing sessions. This subset of the Dell data contain 4,480 phrases, 64,482 key presses, 2,988 errors, and 1825 off–by–one errors and represents expert usage of a mini–QWERTY keyboard. Of the two keyboards used in the studies, the keys on the Dell keyboard are smaller and are more tightly clustered.

Next we analyzed the data for all twenty typing sessions in the original study for the Targus keyboard (Figure 3 top). The complete set of the Targus data contain 16,407 phrases, 246,966 key presses, 8,656 errors, and 4,983 off–by–one errors. The Targus keyboard is the larger of the two keyboards. The keys are large, spaced further apart, and are more ovoid than the keys on the Dell keyboard.

The blind data set consists of data from both the Dell and the Targus keyboards. Four participants per keyboard typed in two different blind conditions for five sessions. The blind conditions have been combined to form one set of data (wpm and accuracy performance in the different conditions was not statistically significantly different). This data set comprises 200 minutes of typing from eight different participants, four of whom used Dell keyboards, and four of whom used Targus. The blind set of data contains 6360 phrases, 55,642 key presses, 5874 errors, and 4326 off–by–one errors.

Training

To detect off–by–one errors, we use the Weka [5] J48 algorithm to learn decision trees with metacost to weight strongly against false positives (10X). Weighting so heavily against false positives helps to ensure that Automatic Whiteout++ minimizes the number of errors that it introduces to the user's typing output. This attribute is important as degrading the user experience is the certainly not a goal of the algorithm.

From the expert Dell data in the original study we randomly assigned 10% of the phrases to be an independent test set and declared the remaining 90% to be the training set. We did not examine the independent test set until all features were selected and the tuning of the algorithm was complete.

From the training set we iteratively built a series of four training subsets, one for each error classifier (roll–on, roll–off, repeats, and substitutions). The training subsets were built by sampling from the larger training set; each subset was designed to include positive examples of each error class, a random sampling of negative examples, and a large number of negative examples that previously generated false positives (i.e., likely boundary cases). Due to our desire to avoid incorrectly classifying a correct keystroke as an error, we iteratively constructed these training sets and searched for proper weighting parameters for penalizing false positives until we were satisfied with the classification performance across the training set. For a list of the most discriminative features for each error classifier, see Table 3.

THE EVALUATION OF AUTOMATIC WHITEOUT++

In the following sections, we demonstrate that Automatic Whiteout++ can successfully generalize across users as well as across different levels of user expertise, different visibility conditions (such as typing while not looking at the keyboard), and different models of keyboards (see Table 4).

Automatic Whiteout++ vs. Automatic Whiteout

Using the expert Dell data set from the original study, we employed "leave–one–out" testing in which we train on data from six of the seven users and test on data from the seventh user. We iterate though this procedure to generates seven combinations of training and test users which yields an approximation of the correction rate Automatic Whiteout++ would achieve when applied to a user whose data is not in the data set.

Comparing the performance of our previous work Automatic Whiteout [4] (Table 5), to Automatic Whiteout++ (Table 6), we find that the performance of Automatic Whiteout++ improves for every class of errors and that the solution as a whole performs considerably better (corrects 46.89% vs. 34.63% of off–by–one errors resulting in a total error rate reduction of 28.64% instead of 26.41%). Roll-on detection rates improved from 47.56% to 64.43%, but roll-off and key repeat detection rates were similar. Most notable is the ability of Automatic Whiteout++ to detect and correct off–by–one substitution errors (this ability did not exist in Automatic Whiteout). While substitution errors remain

Feature Name	Description
Repeats	
sameasprev	that the current key press is the same letter as the previous key press
prob	frequency of finding the current key preceded by the previous two keys. P(X given X-2, X-1)
prevdt	The dt between the current key and the previous key. dt(X,X-1)
nextnext	X+2 's ascii value
Roll–On	
dt	The time between the previous key and the current key dt(X-1, X)
prevcuradjacent	True if and only if the current key is physically adjacent to the previous key.
neighborprobdiff	P(X given X-2, X-1) - P(bestneighbor given X-2, X-1)
Roll–Off	
futdt	The time between the current key and the next key dt(X, X+1)
curfutadjacent	True if and only if the current key is physically adjacent to the next key typed.
prob	frequency of finding the current key preceded by the previous two keys. P(X given X-2, X-1)
Substitutions	
neighborprobdiff	P(X given X-2, X-1) - P(bestneighbor given X-2, X-1)
prob	frequency of finding the current key preceded by the previous two keys. P(X given X-2, X-1)
neighborprob	the probability of the best neighbor occurring after X-2,X-1
futisletter	True if and only if the next key is in [a-z]

Table 3. The most discriminative features learned by the system for each error classifier, ordered by importance.

	across expert–users		across expertise		across keyboards		across visibility	
	train	test	train	test	train	test	train	test
Keyboard	Dell	Dell	Dell	Dell	Dell	Targus	Dell (original)	Targus (blind)
Users	6 of 7 (leave–1–out)	7th (leave–1– out)	6 of 7 (leave–1– out)	7th (leave–1– out)	7	7	7	4
Sessions	Expert (16–20)	Expert (16–20)	All (1–20)	All (1–20)	All (1–20)	All (1–20)	All (1–20)	All (1–5)

Table 4. Summary of training and independent test sets for each of our experiments detailing keyboard, users, and sessions. All tests are user–independent except for the Dell blind study.

Error Type	Avg. corrections (possible)	Avg. detected	Avg. wrong corrections	Avg. OBO error reduction
Roll–On	27.86(58.57)	47.56%	1.0	10.30%
Roll–Off	52.00(64.71)	80.35%	1.71	19.29%
Repeats	13.86(25.29)	54.79%	.71	5.04%
Automatic Whiteout	93.71(146.71)	63.87%	3.43	34.63%

Table 5. Original performance of Automatic Whiteout (from [4]) averaged across seven user–independent tests on the expert Dell data set. Note the absence of substitution corrections

Error Type	Avg. corrections (possible)	Avg. detected	Avg. wrong corrections	Avg. OBO error reduction
Roll–On	37(57.4)	64.43%	2.9	13.36%
Roll–Off	53(63.9)	83.00%	2.3	19.71%
Repeats	14.7(25.9)	56.91%	0.6	5.30%
Subs	25.4(103.1)	24.24%	3.1	8.50%
AW++	120.9(250.3)	48.29%	8.9	46.89%

Table 6. Automatic Whiteout++ across expert users by training and testing on the expert Dell data set. Automatic Whiteout++ performance averaged across seven user–independent tests. On average, users made 260.71 off–by–one errors.

difficult to detect and correct, we can significantly improve results if we keep false positives low. Overall, Automatic Whiteout++ corrects approximately 47% of the off–by–one errors in the data set.

Generalization Across User Expertise

In our preliminary work, we felt that Automatic Whiteout would only be suitable for expert mini-QWERTY typists. Using the new features of Automatic Whiteout++ allows us to further discriminate between correct and incorrect keystrokes and extend the algorithm to correct errors from less experienced typists.

Using the entire Dell data set from the original study we tested the ability of Automatic Whiteout++ to generalize across various levels of user expertise. Again we performed leave–one–out testing. This test yields the rate that Automatic Whiteout++ will detect and correct off–by–one errors at any level of expertise from complete novices (someone who had never used a mini–QWERTY keyboard before) to expert mini–QWERTY keyboard typists. Table 7 shows the results from these tests which are quite encouraging. Given our subject set (expert desktop keyboard users but novice mini–QWERTY users), Automatic Whiteout++ could have improved their typing accuracies significantly at all stages of their training. This result suggests that Automatic Whiteout++ can assist both novice and expert users of such keyboards. It is interesting to note that the percentage of average off–by–one error reduction decreased slightly for roll–on and roll–off errors. This result is because the proportion of these errors as compared to total off–by–one errors increases as the user gains experience.

Generalization Across Keyboards

Using both the entire Dell and Targus data sets from the original study we demonstrate that Automatic Whiteout++ can successfully generalize across different models of mini–

Error Type	Total. corrections (possible)	Avg. detected	Total. wrong corrections	Avg. OBO error reduction
Roll–On	762(1034)	73.69%	44	12.16%
Roll–Off	1092(1234)	88.49%	38	17.46%
Repeats	485(649)	74.73%	9	7.97%
Subs	1120(2888)	37.02%	181	14.69%
AW++	3136(5805)	54.02%	272	52.20%

Table 7. Automatic Whiteout++ across expertise by employing leave–one–out user testing. Trained and tested across all sessions of the Dell data set, Automatic Whiteout++ performance is averaged across seven user–independent tests.

Error Type	Total. corrections (possible)	Avg. detected	Total. wrong corrections	Avg. OBO error reduction
Roll–On	441(666)	66.22%	29	8.55%
Roll–Off	635(765)	83.01%	25	12.26%
Repeats	717(909)	78.88%	9	14.33%
Subs	796(2383)	32.52%	127	13.00%
AW++	2378(4723)	50.35%	190	48.05%

Table 8. Automatic Whiteout++ across different keyboard models. Automatic Whiteout++ was trained on the entire Dell set and was tested on the entire Targus data set from the original experiment.

QWERTY keyboards. Though all mini–QWERTY keyboards by definition have the same alphabetic keyboard layout, not all keyboards have the same sized keys or the same inner–key spacings. As such, not all mini–QWERTY keyboards are used in the same manner. Generalizing across different keyboard models demonstrates the applicability of using the Automatic Whiteout++ solution successfully in mobile devices using different models of mini–QWERTY keyboards.

Perhaps the strongest result in this study (Table 8) is that Automatic Whiteout++ generalized across keyboards. The system had not been trained on either the Targus keyboard or its users' typing in this data set. Yet the system still corrected almost half of the off–by–one errors, corresponding to over a quarter of the total errors made.

Comparing Table 8 to Table 7 which were both trained on all the Dell data (both novice and expert) shows that the types of errors detected were similarly successful across both keyboards. However, despite being trained on Dell data, the Targus keyboard had a lower error rate in general and proportionally fewer roll–on and roll–off errors than the Dell keyboard (probably due to the larger keys of the Targus). Key repeat errors were more common on the Targus, resulting in key repeats having a larger effect on the total off–by–one error reduction, while roll–ons and roll–offs had a lesser effect.

Generalization Across Different Visibility Conditions

To generalize across typing in different visibility conditions, we again used the entire Dell data set from the original study to train the system. As in the previous section, we test on data from both the Dell and the Targus keyboards. However, for this analysis, we use the data for both keyboards from the blind study to evaluate the effectiveness of Automatic Whiteout++ on errors from typing in conditions of limited

Error Type	Total. corrections (possible)	Avg. detected	Total. wrong corrections	Avg. OBO error reduction
Dell				
Roll–On	166(252)	65.87%	18	5.90%
Roll–Off	188(213)	88.26%	13	6.99%
Repeats	43(70)	61.43%	6	1.49%
Subs	581(1941)	28.75%	37	20.63%
AW++	881(2476)	35.58%	74	34.95%
Targus (User Independent)				
Roll–On	68(114)	59.65%	8	2.95%
Roll–Off	138(169)	81.66%	1	6.69%
Repeats	71(92)	77.17%	1	3.38%
Subs	415(1650)	24.06%	37	17.37%
AW++	627(2025)	30.96%	47	30.32%

Table 9. Automatic Whiteout++ across different visibility conditions. Automatic Whiteout++ was trained on the entire Dell set and was tested on the blind Dell as well as the blind Targus data sets.

feedback. In addition to performing a user–independent test on the blind Targus data, we also tested on the blind Dell data. In the original experiment there were seven Dell keyboard users. Four of those seven users participated in the blind study. Due to anonymity procedures for human subjects testing, we did not retain the identities of the subjects who continued to the blind study. Thus, we cannot perform a user–independent test as with our other analyses. Instead, training on the entire Dell data set and testing on the blind Dell data set can be considered neither a user–dependent test nor a user–independent test.

Table 9 shows the results from these tests. As expected, testing on the blind Dell data performed better than testing on the blind Targus data. In the Targus condition, the system was not trained on the users, the keyboard, or the visibility condition. Yet it still corrected 30.3% of the off–by–one errors. Arguably, in practice these rates would be higher because one could train Automatic Whiteout++ on a representative sample of keyboards and operating conditions. Thus, the 22.5% total error corrected in this condition might be considered a low value.

Generalization Across Corpora

Up until this point, the results have been calculated using the letter frequency tables derived from the MacKenzie and Soukoreff phrase set [13]. The phrase set correlates with written English at the single letter frequency level at 95%. However, Automatic Whiteout++ uses bi–gram and tri–gram letter frequencies to assist in error detection and correction.

Table 10 shows the impact that various amounts of context have on the ability of Automatic Whiteout++ to successfully identify and correct errors in mini–QWERTY keyboard text input. With no context Automatic Whiteout++ is able to identify and correct 25.85% of all off–by–one errors. Being able to examine character pairs, Automatic Whiteout++ is able to identify and correct 35.97% of all off–by–one errors. Three letter context improves the efficacy of Automatic Whiteout++ to over 50% (50.32%). Using a dictionary does not improve the solution as recognition rates drop slightly from 50.32% to 50.18%. This lack of improved performance when using a dictionary is worth noting — Automatic

	No Context	1st order	1st + 2nd	1st + 2nd + dict
Roll–On	25.12%	43.03%	64.43%	65.42%
Roll–Off	73.38%	79.42%	83.00%	81.43%
Repeats	6.70%	24.31%	56.91%	59.89%
Subs	3.46%	10.66%	24.24%	23.55%
AW++	25.85%	35.97%	50.32%	50.18%

Table 10. The averaged results (% of off–by–one errors corrected) of leave–one–out user training and testing on the expert Dell data set from the original study using different levels of context.

Whiteout++ is equally successful using a dictionary as it is without a dictionary. The ability to implement Automatic Whiteout++ without having to rely on a dictionary enables the solution to be built directly into the firmware of the keyboard rather than being built into the software of the mobile device. As such, the speed performance gained means that the solution has the potential to detect the error and display the correction without interrupting the user. We hypothesize that the ability to detect and correct errors without visually distracting the user (making a correction within milliseconds before the character is displayed on the screen), will enable faster rates of input and generally a better user experience. In the future we plan to test this hypothesis by running a longitudinal user study of Automatic Whiteout++ to gather human performance data. Additionally, the ability to implement Automatic Whiteout++ in the firmware of a mobile device enables it to work in concert with error correction software native to a particular mobile device. Automatic Whiteout++ simply would pass already corrected input to the software correction system which could then proceed as it would normally on unaltered text.

In general, using a dictionary does not improve the results above the use of tri–letter frequencies. However, there is a distinct improvement in results between the use of single letter frequencies and bi–letter frequencies, and the use of bi–letter frequencies and tri–letter frequencies. The only exceptions are the roll–off errors, which have a consistently high detection rate across language contexts. Given our features, this result suggests that detection of roll–off errors are most dependent on key press timings.

Next we perform a sensitivity analysis using different letter frequency data. We generated up to tri–letter frequencies from the Wikipedia database downloaded on August 5th, 2007. We processed the data keeping only plain text and removing all punctuation, tags, markups, tables, etc. Table 11 shows the results of using the Wikipedia letter frequencies on the expert Dell data set. Comparing these results to those of Table 6 shows average off–by–one error reduction decreases by approximately 3% (46.89% vs. 43.36%). This finding gives a more realistic estimate of how the algorithm would perform on more generalized text.

Sensitivity to Timing

Realizing the important role of timing to the success of our solution, we embarked upon an analysis to determine the impact of imprecise clocks on the performance of Automatic Whiteout++. This analysis was done in an effort to understand how robust the algorithm is to permutations in the timing information it receives from the keyboard. We artificially reduced the resolution of the timing by rounding to the nearest 5, 10, 50, 100, 500, and 1000 milliseconds. The impact on the number of false positives detected by the system can be seen in Table 12. If values are reported to the system with a resolution of at least 100 milliseconds, relatively successful performance of the algorithm can be maintained. If the granularity of the timing information becomes any larger however, the accuracy of the algorithm suffers and would negatively affect the user. Small permutations in the precision of the timing information, however, do not appear to have a large negative impact on performance.

Error Type	Avg. corrections (possible)	Avg. detected	Avg. wrong corrections	Avg. OBO error reduction
Roll–On	34.0(57.4)	59.20%	3.1	12.71%
Roll–Off	54.6(64,7)	84.33%	2.7	21.35%
Repeats	11.6(23.3)	49.69%	0.7	4.49%
Subs	14.1(97.7)	14.47%	2.3	4.85%
AW++	114.3(243.1)	47.01%	8.9	43.36%

Table 11. Automatic Whiteout++ across expert users by training and testing on the expert Dell data set with Wikipedia letter frequencies. Comparing with Table 6, there was a 3.5% absolute reduction in OBO errors corrected.

Timing	Total False Pos	Total True Pos	Total False Negs
Pure	15	2763	2078
5	19	2739	2102
10	23	2726	2111
50	45	2637	2174
100	43	2535	2151
500	3790	2596	2290
1000	5924	2587	2352

Table 12. A timing sensitivity analysis for Automatic Whiteout++ across expert users by training and testing on the expert Dell data set with Wikipedia letter frequencies (timing data reported in milliseconds).

FUTURE WORK

While we are encouraged by our results, many questions remain. Leveraging features of the user's typing and using Automatic Whiteout++ enables us to detect and correct errors as the user types, often mid–word. As a result, the correction can happen almost transparently to the user, and errors can be fixed before the incorrect character distracts the user. We believe that such automatic keystroke level correction might allow the user to sustain rapid typing speeds since the user will be able to input text without being distracted by errors. We are very interested in conducting user evaluations to assess individuals' reaction to the system and to collect mini–QWERTY typing speeds and accuracies both with and without the use of the Automatic Whiteout++ correction system. A longitudinal study that gathers live data will allow us to determine the effects, and noticeability, of the system on users. Do users notice the automatic corrections or the false positives? In an informal pilot test conducted in preparation for the longitudinal study, users did not perceive the presence of the correction system when it was active. However, they did have fewer errors. Unfortunately, the study was not long enough to determine

Test Across	%Off–by–one Errs Corrected	Total Errs Corrected	Keystrokes Corrected
Expert Users	46.89%	28.64%	1.31%
Expertise	52.20%	32.37%	1.15%
Keyboards	48.05%	27.66%	0.96%
Dell Blind	34.95%	26.87%	2.95%
Targus Blind	30.32%	22.48%	2.08%

Table 13. The results of generalizing Automatic Whiteout++ to different expert users, to users of differing levels of expertise, to different keyboards, across visibility conditions and across visibility conditions and keyboards.

any effect on typing speeds, nor did it reveal whether users could become dependent upon Automatic Whiteout++ with long–term use. Expert typists often intuitively "feel" when they make an error in typing and anticipate it, pressing delete before visually confirming the error on the screen. How will Automatic Whiteout++ effect this behavior? Will expert users pause and verify the impact of their preemptive corrections? Such questions certainly merit further investigation.

CONCLUSION

In general, Automatic Whiteout++ can correct approximately 25% of the total errors in the data set (1-3% of the keystrokes typed across users, keyboards, and keyboard and screen visibility conditions). The system introduces less than one tenth as many new errors as it corrects. These false positives could be further reduced with tuning, satisfying our initial concern of the system becoming too intrusive to use. These results are surprisingly good, especially given Automatic Whiteout++ uses only tri–letter frequencies instead of dictionaries for error detection and correction.

Table 13 provides a summary of the results from this study. While all conditions yielded approximately a 25% total error reduction, the percentage of keystrokes corrected ranged between 1% (in the Targus condition) and 3% (in the Dell blind condition). This result is explained by the distribution of errors made in the different conditions. As Targus users gained experience, they made approximately 25% fewer errors than Dell typists. Meanwhile, in the blind conditions, users doubled their error rates on both keyboards. Using these observations and Table 13 as a guide, Automatic Whiteout++ would seem to be most effective on smaller keyboards where device visibility is limited. With consumers buying smaller devices and users' desire to "multitask" sending mobile e-mail in a variety of social situations, Automatic Whiteout++ seems well suited to assist mini–QWERTY typists. If, as we suspect, error correction is time consuming and errors cascade after the first error is made, Automatic Whiteout++ may not only improve accuracies but also improve text entry rates. Future work will investigate the system's effect on the learnability of a keyboard as well as its impact on typing speeds and accuracy.

ACKNOWLEDGEMENTS

This work is funded in part by the National Science Foundation and the National Institute on Disability and Rehabilitation Research. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 0093291. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF. The Rehabilitation Engineering Research Center for Wireless Technologies is sponsored by the National Institute on Disability and Rehabilitation Research (NIDRR) of the U.S. Department of Education under grant number H133E060061. The opinions contained in this paper are those of the author and do not necessarily reflect those of the U.S. Department of Education or NIDRR.

REFERENCES

1. E. Clarkson, J. Clawson, K. Lyons, and T. Starner. An empirical study of typing rates on mini-qwerty keyboards. In *CHI '05: CHI '05 Extended abstracts on Human factors in computing systems*, pages 1288–1291, New York, NY, USA, 2005. ACM Press.
2. J. Clawson, K. Lyons, E. Clarkson, and T. Starner. Mobile text entry: An empirical study and analysis of mini–qwerty keyboards. *Submitted to the Transaction on Computer Human Interaction Journal*, 2006.
3. J. Clawson, K. Lyons, T. Starner, and E. Clarkson. The impacts of limited visual feedback on mobile text entry for the twiddler and mini-qwerty keyboards. In *Proceedings of the Ninth IEEE International Symposium on Wearable Computers*, pages 170–177, 2005.
4. J. Clawson, A. Rudnick, K. Lyons, and T. Starner. Automatic whiteout: Discovery and correction of typographical errors in mobile text input. In *MobileHCI '07: Proceedings of the 9th conference on Human-computer interaction with mobile devices and services*, New York, NY, USA, 2007. ACM Press.
5. S. R. Garner. Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.
6. J. Goodman, G. Venolia, K. Steury, and C. Parker. Language modeling for soft keyboards. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 194–195, New York, NY, USA, 2002. ACM.
7. C. L. James and K. M. Reischel. Text input for mobile devices: comparing model prediction to actual performance. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–371. ACM Press, 2001.
8. K. Lyons, D. Plaisted, and T. Starner. Expert chording text entry on the twiddler one–handed keyboard. In *Proceedings of IEEE International Symposium on Wearable Computing*, 2004.
9. K. Lyons, T. Starner, and B. Gane. Experimental evaluations of the twiddler one-handed chording mobile keyboard. *Human-Computer Interaction*, 2006.

10. K. Lyons, T. Starner, D. Plaisted, J. Fusia, A. Lyons, A. Drew, and E. Looney. Twiddler typing: One-handed chording text entry for mobile phones. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 2004.

11. I. S. MacKenzie, H. Kober, D. Smith, T. Jones, and E. Skepner. LetterWise: prefix-based disambiguation for mobile text input. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 111–120. ACM Press, 2001.

12. I. S. MacKenzie and R. W. Soukoreff. Text entry for mobile computing: Models and methods, theory and practice. *Human-Computer Interaction*, 17:147–198, 2002.

13. I. S. MacKenzie and R. W. Soukoreff. Phrase sets for evaluating text entry techniques. In *CHI '03 extended abstracts*, pages 754–755. ACM Press, 2003.

14. E. Matias, I. S. MacKenzie, and W. Buxton. Half-QWERTY: a one-handed keyboard facilitating skill transfer from QWERTY. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 88–94. ACM Press, 1993.

15. M. Silfverberg, I. S. MacKenzie, and P. Korhonen. Predicting text entry speed on mobile phones. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 9–16. ACM Press, 2000.

16. R. W. Soukoreff and I. S. MacKenzie. Metrics for text entry research: an evaluation of MSD and KSPC, and a new unified error metric. In *Proceedings of the conference on Human factors in computing systems*, pages 113–120. ACM Press, 2003.

17. R. W. Soukoreff and I. S. MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci. *Int. J. Hum.-Comput. Stud.*, 61(6):751–789, 2004.

18. D. Wigdor. Chording and tilting for rapid, unambiguous text entry to mobile phones. Master's thesis, University of Toronto, 2004.

19. D. Wigdor and R. Balakrishnan. TiltText: Using tilt for text input to mobile phones. In *Proceedings of UIST 2003*. ACM Press, 2003.

20. D. Wigdor and R. Balakrishnan. A comparison of consecutive and concurrent input text entry techniques for mobile phones. In *Proceedings of CHI 2004*, pages 81–88. ACM Press, 2004.