Eyes-free Text Entry with Error Correction on Touchscreen Mobile Devices

Hussain Tinwala
Dept. of Computer Science and Engineering
York University
4700 Keele St.
Toronto, Ontario, Canada M3J 1P3
hussain@cse.yorku.ca

I. Scott MacKenzie
Dept. of Computer Science and Engineering
York University
4700 Keele St.
Toronto, Ontario, Canada M3J 1P3
mack@cse.yorku.ca

ABSTRACT
We present an eyes-free text entry method for mobile touchscreen devices. Input progresses by inking *Graffiti* strokes using a finger on a touchscreen. The system includes a word-level error correction algorithm. Auditory and tactile feedback guide eyes-free entry using speech and non-speech sounds, and by vibrations. In a study with 12 participants, three different feedback modes were tested. Entry speed, accuracy, and algorithm performance were compared between the three feedback modes. An overall entry speed of 10.0 wpm was found with a maximum rate of 21.5 wpm using a feedback mode that required a recognized stroke at the beginning of each word. Text was entered with an overall accuracy of 95.7%. The error correction algorithm performed well: 14.9% of entered text was corrected on average, representing a 70.3% decrease in errors compared to no algorithm. Where multiple candidates appeared, the intended word was 1st or 2nd in the list 94.2% of the time.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *input devices and strategies (e.g., mouse, touchscreen)*

General Terms
Performance, Design, Experimentation, Human Factors

Keywords
Eyes-free, text entry, touchscreen, finger input, gestural input, *Graffiti,* auditory display, error correction, mobile computing.

INTRODUCTION
Mobile phones are an integral part of modern day communication, enhancing both information exchange and social interaction in the physical world. One simple example is the coordination of face-to-face meetings using text messaging. Although initially designed for voice calls, mobile phones are now used for text messaging, multimedia sharing, email, web connectivity, media capture and playback, GPS mapping, and so on.

Recently, there is an increased use of touch sensitive technologies on mobile phones. Consumer products employing such interactions were initially limited and unsuccessful, with early products requiring pixel-point accuracy and stylus input. Such accuracy is difficult in mobile contexts. The shift from stylus to finger input changed the landscape and increased user adoption – the Apple *iPhone* is a classic example. Following the *iPhone'*s release in June 2007, many competing products emerged such as Nokia's *5230*, HTC's *Touch HD*, LG's *Prada*, and RIM's *BlackBerry Storm*.

Text input on mobile devices varies considerably. Most devices employ either physical, button-based input or touch-based input using soft controls. Common button-based techniques include the 12-key keypad or a mini-QWERTY keyboard. Because the keys are physical, users feel the location of buttons and eventually develop motor memory of the device. This facilitates eyes-free operation. Eyes-free use is important since mobile interaction often involves a secondary task, such as walking or shopping.

Text input on touch systems typically uses a soft keyboard or gesture recognition. Without physical buttons, tactile feedback is absent, however. This limits the user's ability to engage the kinesthetic and proprioceptive senses during interaction, and imposes an increased need to visually attend to the device. The effect is particularly troublesome if the user is engaged in a secondary task. Consequently, the high visual demand of touch input compromises the "mobile" in "mobile phone".

In the following section, we briefly describe our original prototype. This is followed with a review of related work on automatic error correction. A redesign of the original prototype is then described followed by details of a user study to test the prototype.

NordiCHI 2010, October 16–20, 2010, Reykjavik, Iceland.
 ISBN: 978-1-60558-934-3...$5.00.

OVERVIEW OF THE ORIGINAL PROTOTYPE

In earlier work, we presented a gesture-based text entry interface using *Graffiti* (an example of *Unistrokes* [5]) for eyes-free input on a touchscreen device [16]. The system provided visual feedback but eyes-free entry was also possible using auditory and tactile stimuli. The system described here includes several improvements (described later).

To enter text, users draw strokes on the display surface using a finger. Digitized ink follows the user's finger until it is raised. At the end of a stroke, the application analyses the stroke shape to identify the intended character. A recognized character is complemented with speech feedback: the letter is spoken. Upon word completion, a SPACE is inserted and the word is appended to the message (see Figure 1). If a stroke is unrecognized, the user is alerted with a short pulse of vibration from the built-in actuator.

Figure 1. Text entry interface for eyes-free input. The stroke map is enhanced for clarity.

The *Graffiti* alphabet is overlaid on the screen to promote learning. In related work, the strokes were displayed away from the interface [9, 18], thus demanding visual attention at a separate location from the interface. This could potentially affect throughput. The stroke alphabet was display-only; so, the entire display surface was available as the drawing surface.

An evaluation with 12 participants comparing eyes-on and eyes-free modes found an overall entry speed of 7.3 wpm (7.0 wpm eyes-on and 7.6 wpm eyes-free). A higher *KSPC* (keystrokes per character) was observed in the eyes-free mode, suggesting that the lack of visual feedback decreases input accuracy, necessitating more corrective strokes.[1] The results of the initial evaluation were promising; however, areas of improvement were apparent. One deficiency was the lack of system-assisted error correction. In this paper, we present an improved version of the system. One of the main features is an algorithm for automatic error correction.

ERROR CORRECTION

Error correction methods use algorithms for approximate or exact text matching [e.g., 1, 7, 12, 13, 15, 17]. Three temporal points of error identification and correction in text entry are *error prevention*, *automatic error correction*, and *user-initiated spell checking*. The following sections review error correction techniques based on this categorization.

Error Prevention

At first, it seems paradoxical to consider correcting an error before it is committed. The idea is error *prevention*, rather than error correction. MacKenzie et al. proposed *LetterWise*, where a prefix determines the most likely character(s) to follow [10]. Some systems deal with errors in the prefix as well, but we discuss these in the next section. With fixed vocabularies, prefix-based methods provide an efficient means to prevent user errors before they occur. An example is the entry of street and city names on a GPS device. As the prefix length increases, the list of names narrows. Once the list is small enough, it is displayed to the user as options.

In a similar vein, Hoffman et al. presented a hardware-based solution called *TypeRight* [6]. Based on a prefix sequence, a dictionary, and grammar rules, the keyboard decreases errors by dynamically increasing the tactile resistance of less likely keys. Error correction rates were decreased by 46%.

Automatic Error Correction

Automatic Whiteout++ corrects common errors during entry, such as hitting a neighboring key, character substitution, or transposition ("the" instead of "teh") [3]. When tested on data from previous mini-QWERTY keyboard experiments, the system corrected 32% of the errors automatically. Instead of using a dictionary, the algorithm detects errors based on keypress timings and letter (di-graph) frequencies.

Kristensson and Zhai proposed an error correction technique using geometric pattern matching [8]. For example, entering "the" on a QWERTY keyboard forms a spatial pattern. With their method, it is possible to enter "the" even if the user actually enters "rjw", because the patterns are geometrically similar. Pattern recognition was performed at the word level, when SPACE was entered. Overall, their system had a success rate of 83%.

[1] The "K" for keystrokes in *KSPC* applies to any primitive action, including stylus or finger strokes.

Some input techniques make corrections when the prefix is erroneous or when a delimiter appears at the end of a word. The default error correction software on the Apple *iPhone* uses both these methods. The approach is to analyze the prefix in the input stream for each word. If the prefix is erroneous, while neighboring keys of the prefix yield a valid dictionary word, the valid word is presented as a suggestion. At the same time, the system learns and reorders the dictionary based on the user's accept/reject selections, thus influencing future suggestions.

Other methods are more familiar such as the capitalization and de-capitalization of letters and the reordering of letters ("adn" to "and"). These methods are part of the *auto-correct* feature found on most word processors. If multiple matches are found for a sequence of characters, the word is marked with a dotted or squiggly red underline. At this point the user can correct a misspelled word, or run a "spell checker" to correct words one by one.

The Apple *Macintosh* supports error identification and correction at the operating system level, independent of the application. In addition to the basic techniques, the system uses context and grammar to determine if a correction is needed. For instance, entering "teh byo adn" identifies all three words as errors. Entering "teh byo adn girl" corrects the text to "the boy and girl". This is an interesting behaviour, since correcting each word individually reveals multiple suggestions. This is a departure from the way many word processors handle spelling errors (i.e., at the word level).

The idea of automatic correction described by Robinson et al. in a U.S. patent comes close to the solution we describe [14]. The patent does not present a concrete system or an evaluation, but articulates the following concept (some details omitted). Receive handwriting input → determine a list of word candidates based on the input → use frequency indicators to decide which words to present → present one or more candidates for user selection.

User-Initiated Spell Checking

As interfaces are increasingly intelligent, the number of applications that identify and correct errors after text entry is decreasing. However, certain applications are still available for dynamic error identification and correction. Many online web applications, such as blogs and site builders, are examples. Usually, they rely on a spell checker run by the user after text entry is complete (post-processing). However, this is rapidly changing. Web applications are improving in this regard (e.g., Google *Wave*[2]). Furthermore, some browsers provide spell checking at the application level.

Note that error correction techniques often fail. Failures are due to a variety of reasons such as high ambiguity, insufficient context, etc. These techniques enter a fallback mode where error correction is initiated by the user, and performed by the system in cooperation with the user on a word-by-word basis. For each erroneous word, the user selects from a list of options or provides one.

THE REDESIGN PROCESS

Issues Found

In the original prototype, the first shortcoming was the speech feedback. Although of good quality, informing the user of every letter via speech was tedious – even for the eyes-free mode. In addition, users invested time confirming each letter after each stroke. This added significantly to the overall text entry time, thus lowering throughput. Furthermore, there was no feedback at the end of a word, making it difficult to determine what word was entered in the eyes-free mode. This increases the potential for the user to forget her position in a phrase.

Second, the interaction provided vibrotactile feedback when a stroke was not recognized (*unrecognized stroke)*. The device vibrated to alert the user and allowed for repeated attempts. Users acknowledged this as useful during training, but found it cumbersome and time-consuming. Because novice users are unaware of the nuances of *Graffiti*, this led to multiple retries for certain strokes until they were learned. This generated many vibrations and frustrated users.

Lastly, the lack of automatic error correction meant that the system did not assist users when entering text. Automatic error correction ("system help") can potentially improve interaction quality, particularly in the eyes-free mode.

Speech and Vibrotactile Feedback

From our observations, we decided on a different approach to system feedback. The first enhancement involved shifting the speech feedback from the character-level to the word-level. Users are alerted to the word entered, via speech, when a SPACE is entered (double-tap).

Redesigning the interaction can produce new problems, however. For instance, providing word-level feedback suggests removing the character-level vibrotactile feedback for unrecognized strokes. Without the vibrotactile effect, users would be unaware of unrecognized strokes and without character-level speech, users would be unaware of misrecognized strokes.

In the redesigned system, users hear a short, non-speech "click" with each character/stroke received. The click is sounded even if the stroke is unrecognized. Once the word is complete, users double-tap to enter a SPACE. At this point, the system speaks the word entered (subject to the results of the error correction algorithm; see below). Chunking text at the word level allows for fewer interruptions during text entry and alerts users to the last entered word instead of the last entered character. It is anticipated that this will improve the flow of the interaction

[2] http://wave.google.com/

and increase throughput. However, this approach has the potential to produce more errors, since no feedback is provided at the character-level other than the click sound. To handle this, an error correction algorithm was employed.

Error Correction Algorithm

We designed an algorithm with several goals in mind. The algorithm handles errors that occur when a character is unrecognized, misrecognized, or wrong (i.e., a spelling error). As well, it assists users in finding the right word using a dictionary if multiple candidate words are found.

Handling Stroke Errors

When an unrecognized stroke is encountered, a period is inserted in the text stream. As an example, consider "hello" where the first "l" is unrecognized. There is no interruption to the user. Instead the unrecognized letter is replaced with a period, forming "he.lo". In essence, the period acts as a marker. The system knows there is a character at the marker position, but it is an unknown character. Auto correct mechanisms do not accommodate situations like this.

In the event of a misrecognized stroke (or spelling mistake), no changes are made. The application simply accepts the stroke because, at this point, it is not known if the stroke was misrecognized. For instance, consider again the word "hello", where the second occurrence of "l" is misrecognized as "i". In this case, the text is "helio". Combining the two errors, the text is "he.io".

Although bigram and trigram frequency lists can aid in detecting misrecognized strokes *when* they occur (i.e., *during* entry for a word), it is not convenient to use them in an eyes-free setting where there is no visual feedback.

The next step is to handle these errors. Once the user finishes a word and double taps to enter a SPACE, the word is spoken provided the character sequence matches a word in the dictionary. If there is no match, the error correction algorithm is invoked. The algorithm works with a dictionary in attempting to correct the error. The dictionary in the prototype was obtained from the British National Corpus [2]. There are 9,000 unique words and frequencies, beginning as follows:

```
the 5776384
of 2789403
and 2421302
...
```

The error correction algorithm is discussed next.

Regular Expression Matching

The first task involves narrowing the search space. It is assumed the user entered the correct length of the word. Based on this, the search is conducted on all words of the same length in the dictionary. So, for the example of "hello", a search is conducted on all words of length 5. There are about 1200 such words in the test dictionary.

If "hello" was entered as "he.lo" the algorithm searches for all words that match "he.lo" such that any matching character replaces the period. The result is a single match, "hello". Any other word with unrecognized characters is dealt with similarly. If the spelling is correct and some of the characters are unrecognized, regular expression matching provides a resilient mechanism for identifying the correct word. However, if there are spelling errors or misrecognized characters, an alternative technique is employed.

Minimum String Distance Searching

The minimum string distance (*MSD*) between two strings is the minimum number of primitives – insertions, deletions, or substitutions – to transform one string into the other. Using this metric, it is possible to detect misrecognized characters and find matching words. Consider the following example where "heggo" is transformed into "hello":

```
heggo <substitute g with l>
helgo <substitute g with l>
hello <matches 'hello' in dictionary>
```

The above transformation requires two substitute operations to transform "heggo" to "hello". Hence, *MSD* is 2. Note that in this algorithm, the focus is on substitution primitives due to the assumption that the word length is correct. Hence, it is possible to narrow the search space drastically and find a viable match.

However, a problem is determining the bounds of the *MSD* value, since it is not known how many misrecognized characters exist in the entered text. An *MSD* value of 1 may find nothing. On the other hand, searching for all words that fit into an *MSD* value of, say, 1-4 may result in too many inappropriate matches. To handle this, we used data from an earlier experiment to develop a heuristic. The resulting *MSD* mapping is a function of the word length, as follows:

```
if wordLength is 1-4
    use MSD = 1
else if wordLength is 5-6
    use MSD <= 2
else if wordLength is 7-8
    use MSD <= 3
else // wordLength is > 8
    use MSD <= FLOOR(wordLength/2)
```

For words up to length 4, the algorithm assumes 1 misrecognized or unrecognized character of text. For words that are either 5 or 6 characters long, the algorithm allows for 2 erroneous characters, and so on. For words with length >8, the number of allowable erroneous characters is the floor of half the word length. There is one caveat. In the event no matching words are found, the *MSD* limit is incremented by one and the search is repeated. This modifies the mapping for words with length less than 9 as follows: 1-4 characters → MSD ≤ 2; 5-6 characters → MSD ≤ 3; 7-8 characters → MSD ≤ 4; 9 or more characters → no change.

Combining the Results

The final step is to merge the results of the two search operations. The merge operation is done as follows:

```
listA = words found using regular
        expression matching
listB = words found using MSD matching
listC = listA ∪ listB
```

`listC` is sorted by frequency so that the word with the highest frequency is first. Also, duplicates are eliminated. Table 1 presents sample errors and the suggestions found by the correction algorithm, sorted by frequency.

Search Key [word]	Matches Found
hel.o [hello]	*hello*, helen, helps
compu..r [computer]	*computer*, composer
begauze [because]	*Because*
ap.lg [apple]	Apply, *apple*
.uitas [guitar]	*guitar*, quotas
siz..rs [sisters]	*Sisters*, singers
poeans [oceans]	poland, romans, *oceans*
chs..er [chapter]	*chapter*, chamber, charter, cheaper, chester

Table 1. Sample search words and results.

The Auditory Display

The error correction algorithm is pivotal for word-level interaction; however, it must smoothly integrate with the interaction. If the word entered is correct (i.e., the character sequence is in the dictionary), the word is accepted. If the character sequence is not in the dictionary, the error correction algorithm is invoked. If the result is a single match (see 3rd example in table above), the word is accepted. In either of these cases, the user is informed of the word through speech feedback. From the user's perspective, it is not known if an error occurred.

If the algorithm returns multiple words, the device sounds a two-tone bell and enters a "playback mode" – an auditory display. During this mode, the words in the set are spoken one after the other. Recognition of *Graffiti* strokes is suspended. Only three strokes are supported during playback:

North stroke:	restart playback
Delete stroke (left swipe):	clear word and re-enter
Single tap:	accept last played word

Words spoken are separated with 600 ms silence to allow time to accept or reject the last spoken word. Playback is cyclical; the start of each cycle is signaled with the two-tone bell. Users can restart playback by drawing a north stroke, discard the word with a delete stroke, or select the word with a single tap. Word selection is confirmed by pronouncing the selected word again.

Given the error correction algorithm and the interaction possibilities described above, an experiment was carried out to test eyes-free interaction with three different feedback modes. These feedback modes are described next, followed by the methodology and results of the experiment.

FEEDBACK MODES

To test the enhancements and correction algorithm, three feedback modes were used.

Immediate

For the Immediate mode, users receive speech feedback for each character entered. This behavior is the same as in the original prototype with the addition of word-level speech when SPACE is entered. The error correction algorithm is not used in this mode.

OneLetter

For the OneLetter mode, users must enter a valid first stroke. If the first stroke is unrecognized, the system prevents the user from proceeding and outputs a pulse of vibration. For this mode, the first letter is spoken; the remaining letters produce "click", irrespective of the outcome of recognition. When a SPACE is entered, the word is spoken if the character sequence is in the dictionary, or the algorithm is invoked if the character sequence is not in the dictionary. The motivation behind this mode is to narrow the search space to improve the probability of finding the correct word.

Delayed

For the Delayed mode, there are no restrictions on the user. Each stroke is accompanied with "click" and there is no requirement for a valid first stroke. When a SPACE is entered, the word is spoken if the character sequence is in the dictionary, or the algorithm is invoked otherwise. The search space is larger for this mode (in the event of an error); however, throughput may be higher since the user need not hesitate to confirm entry of the first character.

EVALUATING THE INTERACTION

Given the above feedback modes, error correction algorithm, and auditory display, an evaluation testing these enhancements was carried out. Our goal is to investigate whether the prototype changes result in improved interaction. We expect that the Delayed mode will enhance text entry by increasing entry rates and decreasing error rates. The OneLetter mode may result in slightly less throughput but better accuracy due to the requirement of a valid first stroke. The Immediate mode was tested as a baseline for comparison against the original interaction method [16].

Participants

Twelve paid volunteer participants (2 female) were recruited from the local university campus. Participants ranged from 18 to 40 years (*mean* = 26.6, *SD* = 6.8). All were daily users of computers, reporting 2 to 12 hours usage per day (*mean* = 6.7, *SD* = 2.7). Six used a touchscreen phone regularly ("several times a week" or "everyday"). Participants had no prior experience with the

system. Eight participants had tried *Graffiti* before, but none was an experienced user.

Apparatus

The hardware consisted of an Apple *iPhone 3G* (firmware: 3.1.2), an Apple *MacBook* host (2.4 GHz Intel *Core 2 Duo* with 2 GB of RAM), and a private wireless ad-hoc network (see Figure 2). The host system was used for data collection. The two devices communicated wirelessly, allowing users freedom of movement during the experiment.

Figure 2. Hardware for experimentation.

The host application was developed using *Cocoa* and Objective C. The development environment was Apple's *Xcode*. The device application was developed using *OpenGL ES* and in the same environment as the host application (*Xcode*).

The host application listened for incoming connections from the *iPhone*. Upon receiving a request and establishing a connection, a set of 500 test phrases [11] was read to prepare for the first trial.

The software recorded time stamps for each stroke, word and phrase level data, ink trails of strokes, and other statistics for follow-up analyses. Timing for each phrase began when the display was touched for the first stroke and ended with the insertion of SPACE after the last word.

Procedure

The experiment was performed in a quiet room. Participants adjusted their position on a height-adjustable chair to position the device and their hands under the table.

Prior to data collection, participants completed a pre-test questionnaire soliciting demographic data. The experiment began with a training session. This involved entering the alphabet A to Z three times, entering the phrase "the quick brown fox jumps over the lazy dog" twice, and entering one random phrase from the phrase set. The goal was to bring participants up to speed with *Graffiti* and minimize any learning effects or transfer of skill from prior experience. Training was followed by three blocks of entry for each feedback mode: Immediate, OneLetter, Delayed – all eyes-free. Four consecutive phrases formed one block of text entry. The experimenter explained the task and demonstrated each mode, including the method to enter a SPACE (double tap) and interacting with the auditory display for the OneLetter and Delayed modes. User initiated error correction (left swipe) was restricted to the most-recently entered character only. This restriction served as a means to reduce variability across participants.

Participants were asked to proceed "as quickly and accurately as possible" and were allowed to take breaks between phrases and blocks, if desired. Testing lasted 50-60 minutes for all three conditions in the experiment. The interaction was two-handed requiring participants to hold the device in one hand while performing text entry with the index finger of the other hand. Participants held the device in their non-dominant hand and entered text with their dominant hand. During testing, the device was under the table and occluded from view to ensure eyes-free entry in all three conditions.

Certain characters posed difficulty, such as the letter "G". For this and other such characters, alternative entry methods were demonstrated. Figure 3 shows two ways of entering "G". Preliminary tests revealed that entering G as on the left was harder than the alternative – drawing a six.

Figure 3. Ink trails for two ways of drawing "G".

Design

The experiment was a 3 × 3 within-subjects design. There were two independent variables:

Feedback Mode (Immediate, OneLetter, Delayed)

Block (1, 2, 3).

The feedback mode conditions were counterbalanced using a Latin square. Aside from training, the amount of entry was 12 participants × 3 feedback modes × 3 blocks × 4 phrases/block = 432 phrases.

RESULTS AND DISCUSSION

Several dependent variables were measured through the course of the experiment. Results for the basic metrics of speed and accuracy are presented first. These are followed by additional investigations on the performance of the error correction algorithm and a closer look at the quality of the word suggestions.

Entry Speed

The results for entry speed are shown in Figure 4. The overall mean rate was 10.0 wpm. As expected, entry speed increased significantly across blocks ($F_{2,18} = 6.2$, $p < .05$).

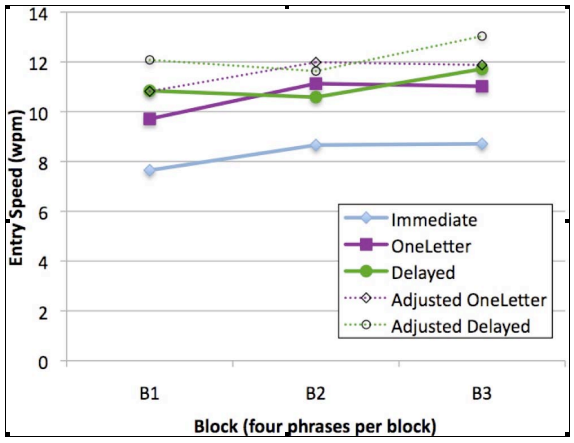

Figure 4. Entry speed (wpm) by entry mode and block.

There was also a significant difference by entry mode ($F_{2,18} = 32.3$, $p < .0001$).

The average entry speed for the Immediate mode was 8.34 wpm. Entry speeds were 27% faster for the OneLetter mode, at 10.6 wpm, and 33% faster for the Delayed mode, at 11.1 wpm. A post hoc Scheffé test revealed significant differences between the Immediate-OneLetter and Immediate-Delayed pairings ($p < .0001$). Overall, these results are quite good. The entry speeds are faster than the 7.6 wpm observed in our earlier experiment [16] and faster than the novice entry rate of 7.0 wpm for *Graffiti* reported by Fleetwood et al. [4].

The maximum entry speed for individual phrases sheds light on the potential of each mode. The OneLetter mode obtained the highest rate at 21.5 wpm, followed by the Delayed mode at 20.8 wpm and Immediate mode at 16.9 wpm. Again, these results are noteworthy, particularly considering there were only about 15 minutes of testing for each mode.

The graph also provides "adjusted" text entry rates for the OneLetter and Delayed modes. The adjusted rates remove the time spent in playback mode, pretending as though there was always a single word that matched the participant's input and so no time was invested in dealing with errors or collisions. For both modes, the improvement is about 10%.

Accuracy

The main accuracy measure is error rate computed using the minimum string distance (*MSD*) between the presented and final text. Since the final text was subject to correction using the error correction algorithm, this measure is called the "final error rate". See Figure 5. Overall, the final error rates were low at 4.3% (accuracy > 95.0%). The effect of feedback mode on final error rate was significant ($F_{2,18} = 8.2$, $p < .005$). The Delayed mode had the highest rate at 7.0%. This was 2× higher than OneLetter at 3.5% and 2.8× higher than Immediate, at 2.5%. Although a block effect

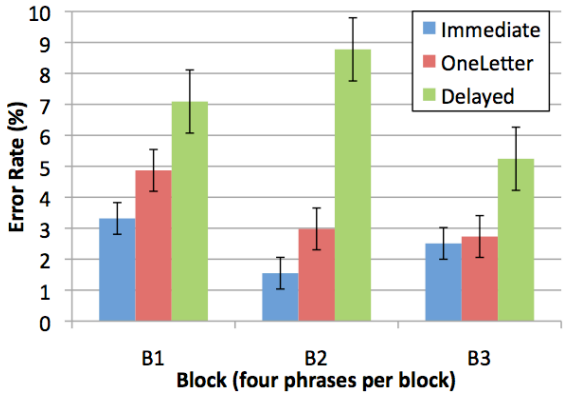

Figure 5. Final error rate (%) by entry mode and block.

was expected, none was found. This is partially due to participants not having a direct influence on the error correction algorithm. The algorithm is designed to cater to individual differences, which may have prevented a block effect from emerging.

A post hoc Scheffé test revealed significant differences between the Immediate-Delayed and OneLetter-Delayed pairings ($p < .0001$). Variation in final error rate between the Immediate-OneLetter pairing was insignificant, suggesting that the error correction algorithm worked better when the first letter of a word was valid. This is best observed in the differences between OneLetter and Immediate in block 3 of the figure; they are marginal.

Table 2 presents examples of the presented phrases and the entered and final text. The variation in the entered and final text gives a sense of the utility of the error correction algorithm.

Presented Entered Corrected	Error Count
elephants are afraid of mice e.e.hancs are a.ratd .. m..e elephants are afraid of mice	9
question that must be answered ....tion tha. must be answered question that must be answered	5
the fax machine is broken th. fax machin. is brpken the fax machine is broken	3
three two one zero blast off three .w. .ne zer. b.ast of. three two one zero blast off	6
fall is my favorite season fal. is m. fau.ritg seas.. fall is my favorite season	7
do not walk too quickly d. n.t wa.. too quic.lo do not walk too quickly	6
stability of the nation stadilit. .. the nati.n stability of the nation	5

Table 2. Sample of presented, entered, and corrected text.

KSPC Analysis

Similar to the previous experiment [16], *KSPC* was used to measure the overhead of user error correction on text entry. If entry was perfect, the number of strokes equals the number of characters and *KSPC* = 1. (Note: double-tap was counted as one stroke.) Results of this analysis are shown in Figure 6. Since the chart uses a baseline of 1 and perfect input has *KSPC* = 1, the entire magnitude of each bar represents the overhead for unrecognized strokes or for errors that users corrected.

Overall, average *KSPC* was 1.27. *KSPC* for the Immediate mode was highest at 1.45. OneLetter was 16.6% lower at 1.21, while Delayed was 19.3% lower than Immediate, at 1.17. The trend was consistent and significant between entry modes ($F_{2,18} = 51.8$, $p < .0001$), but not within blocks. A post hoc Scheffé test revealed significant differences between the Immediate-OneLetter and Immediate-Delayed pairings ($p < .0001$), but not between the OneLetter-Delayed pairing. This is expected as the latter two modes are similar and vary only in the requirement of one valid stroke per word. Also, *KSPC* was the same or decreased from one block to the next in the Immediate and OneLetter modes. For the Delayed mode, *KSPC* increased, albeit slightly, from the first block to the second. It remained constant from block two to block three. The OneLetter and Delayed modes were intended to decrease user effort. This is clearly reflected in the results.

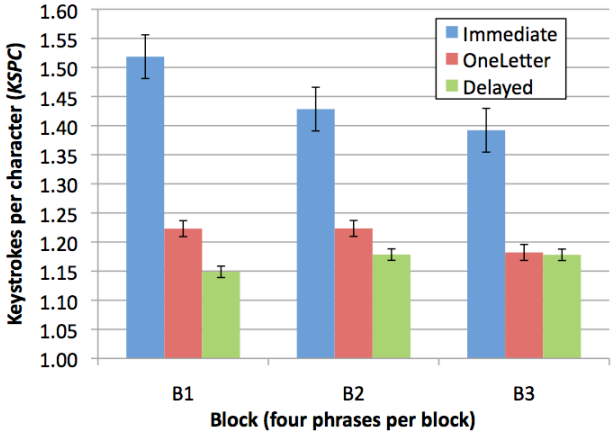

Figure 6. Keystrokes per character (*KSPC*) by block and entry mode.

System Help

The error correction algorithm aimed to enhance the text entry experience through a robust mechanism to handle text entry errors. Simply put, the burden of correcting errors shifted from the user to the system. "System help" is a metric identifying the percentage of entered text transcribed incorrectly but successfully corrected by the correction algorithm. The results are depicted in Figure 7.

The error correction algorithm played an important role in text entry. Overall, the algorithm corrected 14.9% of entered text. For a 30-character phrase, this is equivalent to

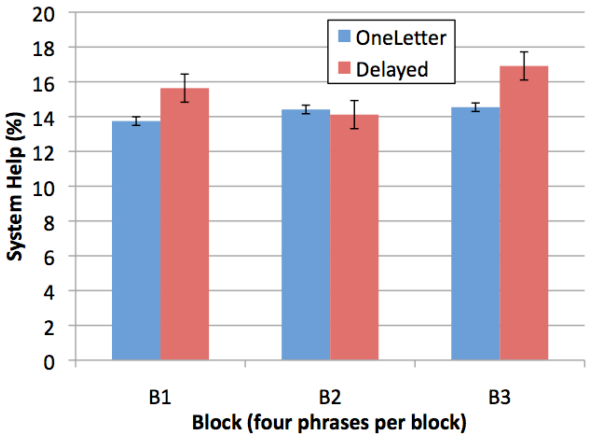

Figure 7. System help (%) by mode and block. (Note: The algorithm was not used with Immediate mode.)

4.5 characters, or one word. In the Delayed mode, 15.6% of entered text was corrected. The value was 14.2% for the OneLetter condition. A post hoc Scheffé test revealed no difference for system help between the OneLetter-Delayed pairing.

The amount of errors is not small. This is expected as the lack of audio feedback at the character-level in the OneLetter and Delayed modes made it impossible for participants to verify input at the character level.

Figure 8 presents one final illustration of how errors were handled. Raw error rate is for the entered text. Corrected error rate is for the final text.

The Immediate mode had no automatic error correction so both rates are equal. However, the stark difference in magnitude between the raw and corrected error rates for the other two modes highlights the effect of the algorithm. Overall, error rates decreased by 70.3%. Error correction worked best in the OneLetter mode with a net improvement of 76.7%. The improvement in the Delayed mode was 64.0%. The OneLetter rates are lower overall due to the requirement of a valid first character. This improves the

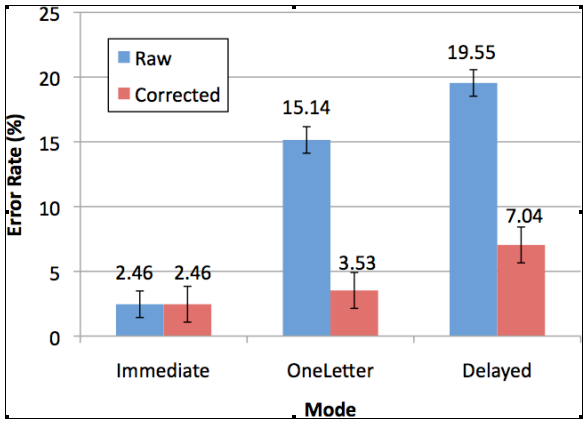

Figure 8. Raw and corrected error rates.

effect of the algorithm, since having the first character correct dramatically narrows the search space and increases the likelihood of finding the intended word.

Playback Mode (Auditory Display)

If the error correction algorithm finds multiple matches for a word, the list is produced using the playback mode. Two points of interest here are the size of the lists and the position of the intended word.

Candidate List Size

Figure 9 shows the average candidate list size per word by block and feedback mode. The average size overall was 2.43 words. List size for OneLetter averaged 1.89 words. Not requiring a valid stroke for the first character for the Delayed mode resulted in a 56.0% higher list size, at 2.96 words on average. The list size difference between the OneLetter and Delayed modes was significant ($p < .05$).

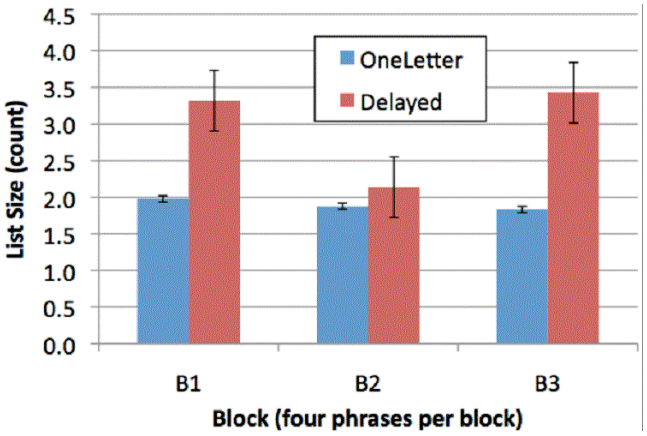

Figure 9. Mean list size per word by block.

OneLetter fluctuated by small amounts, decreasing in value across blocks. The implication of a shorter list is that more characters in each word are entered correctly, so there is a slight learning effect visible, likely due to the audio feedback for the first character. This reinforced the alphabet and allowed users to learn the nuances when the stroke was unrecognized. The same trend is not visible in the Delayed mode. This is due to the lack of character-level feedback, thus inhibiting any meaningful learning to take place.

Word Position

Figure 10 deconstructs the position of the desired word in the candidate lists. About 77.0% of the words in OneLetter mode were at position 1. The figure for the Delayed mode is 82.0%. Grouping positions 1 and 2 together, the number is 94.2% (95.3% OneLetter, 93.2% Delayed). This is a promising result and demonstrates that the regular expression matching and *MSD* searching characteristics of the correction algorithm work well and provide a resilient mechanism for handling unrecognized strokes, misrecognized strokes, and spelling errors.

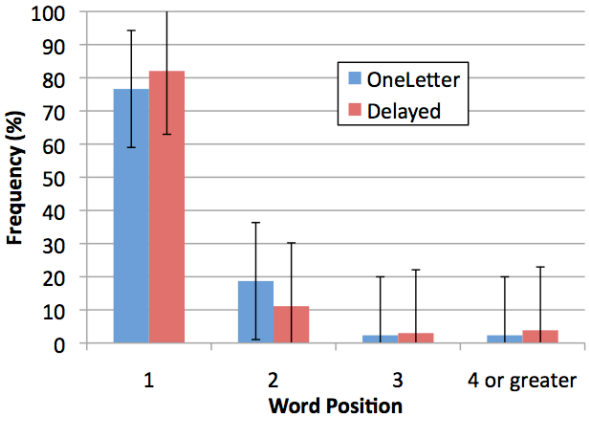

Figure 10. Word position frequency by feedback mode.

CONCLUSION

We presented an enhanced version of an eyes-free text entry interface for touchscreen devices. Audio feedback was shifted from character-level to word-level, providing speech output at the end of each word. Vibrotactile feedback was used only for the OneLetter mode, which required a recognized stroke at the beginning of each word. The entered text (with the errors) is passed through a dictionary-based error correction algorithm. The algorithm uses regular expression matching and a heuristically determined minimum string distance search to generate a list of candidate words based on the entered word. The list is presented in an auditory display, in the form of a playback mode.

In a user study, the overall text entry speed was 10.0 wpm with a maximum rate of 21.5 wpm using a feedback mode that required a recognized stroke at the beginning of each word. Text was entered with an overall accuracy of 95.7%. The error correction algorithm performed well: 14.9% of entered text was corrected on average, representing 70.3% decrease in errors compared to no algorithm. Where multiple candidates appeared, the intended word was 1st or 2nd in the list 94.2% of the time.

As touchscreen phones lack the tactile feel of a physical keyboard, the visual demand on the user is increased. Our research demonstrates that eyes-free text entry is possible on a touchscreen device and with performance that is both reasonably fast and accurate. In a wider context, the text entry method described here can be used in scenarios where users are multitasking and attention is limited. Finally, a contribution of this research is applications in accessible computing for visually impaired users. Although the participants of this research cannot be equated to visually impaired users, their success at entering text eyes-free suggests that the method may serve as an accessible alternative to users with impaired vision. However, determining the extents of this possibility requires further research.

REFERENCES

1. Baeza-Yates, R. and Navarro, G. (1998). Fast approximate string matching in a dictionary. *Proceedings of String Processing and Information Retrieval: A South American Symposium*, 14-22. New York: IEEE.
2. BNC. (2009). British National Corpus of the English Language. BNC, ftp://ftp.itri.bton.ac.uk/.
3. Clawson, J., Lyons, K., Rudnick, A., Robert A. Iannucci, J. and Starner, T. (2008). Automatic Whiteout++: Correcting mini-QWERTY typing errors using keypress timing. *Proceeding of the ACM Conference on Human Factors in Computing Systems – CHI 2008*, 573-582. New York: ACM.
4. Fleetwood, M. D., Byrne, M. D., Centgraf, P., Dudziak, K. Q., Lin, B. and Mogilev, D. (2002). An evaluation of text-entry in Palm OS - Graffiti and the virtual keyboard. *Proceedings of the 46th Annual Meeting of the Human Factors and Ergonomics Society – HFES 2002*, 617-621. Santa Monica, CA: HFES.
5. Goldberg, D. and Richardson, C. (1993). Touch-typing with a stylus. *Proceedings of the ACM Conference on Human Factors in Computing Systems - CHI 1993*, 80-87. New York: ACM.
6. Hoffmann, A., Spelmezan, D. and Borchers, J. (2009). TypeRight: A keyboard with tactile error prevention. *Proceedings of the ACM Conference on Human Factors in Computing Systems – CHI 2009*, 2265-2268. New York: ACM.
7. Horst, B. (1993). A fast algorithm for finding the nearest neighbor of a word in a dictionary. *Proceedings of the Second International Conference on Document Analysis and Recognition – ICDAR 1993*, 632–637. Tsukuba, Japan: IEEE.
8. Kristensson, P.-O. and Zhai, S. (2005). Relaxing stylus typing precision by geometric pattern matching. *Proceedings of the ACM Conference on Intelligent User Interfaces – IUI 2005*, 151-158. New York: ACM.
9. MacKenzie, I. S., Chen, J. and Oniszczak, A. (2006). Unipad: Single-stroke text entry with language-based acceleration. *Proceedings of the Fourth Nordic Conference on Human-Computer Interaction – NordiCHI 2006*, 78-85. New York: ACM.
10. MacKenzie, I. S., Kober, H., Smith, D., Jones, T. and Skepner, E. (2001). LetterWise: prefix-based disambiguation for mobile text input. *Proceedings of the ACM Symposium on User Interface Software and Technology – UIST 2001*, 111-120. New York: ACM.
11. MacKenzie, I. S. and Soukoreff, R. W. (2003). Phrase sets for evaluating text entry techniques. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems – CHI 2003*, 754-755. New York: ACM.
12. Navarro, G. and Raffinot, M. *Flexible pattern matching in strings: Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.
13. Oflazer, K. (1996). Error-tolerant finite-state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, *22*, 73-89.
14. Robinson, A., Bradford, E., Kay, D., Meurs, P. V. and Stephanick, J. (2008). *Handwriting and voice input with automatic correction*. U.S. Patent 7,319,957 B2, Jan. 15, 2008.
15. Sinha, R. M. K. (1990). On partitioning a dictionary for visual text recognition. *Pattern Recognition*, *23*, 497-500.
16. Tinwala, H., and MacKenzie, I. S. (2009). Eyes-free text entry on a touchscreen phone. *Proceedings of the IEEE Toronto International Conference – Science and Technology for Humanity – TIC-STH 2009*, 83-89. New York: IEEE.
17. Wells, C. J., Evett, L. J., Whitby, P. E. and Whitrow, R. J. (1990). Fast dictionary look-up for contextual word recognition. *Pattern Recognition*, *23*, 501-508.
18. Wobbrock, J. O., Myers, B. A., Aung, H. H. and LoPresti, E. F. (2004). Text entry from power wheelchairs: EdgeWrite for joysticks and touchpads. *Proceedings of the ACM Conference on Computers and Accessibility – ASSETS 2004*, 110-117. New York: ACM.