B#: Chord-based Correction for Multitouch Braille Input

Hugo Nicolau[1], Kyle Montague[1], Tiago Guerreiro[2], João Guerreiro[2], Vicki L. Hanson[3, 1]
[1]University of Dundee, Dundee, Scotland
[2]University of Lisbon, Lisbon, Portugal
[3]Rochester Institute of Technology, NY, USA
{hnicolau, kmontague}@dundee.ac.uk, tjvg@di.fc.ul.pt, joao.p.guerreiro@ist.utl.pt, vlh@rit.edu

ABSTRACT
Braille has paved its way into mobile touchscreen devices, providing faster text input for blind people. This advantage comes at the cost of accuracy, as chord typing over a flat surface has proven to be highly error prone. A misplaced finger on the screen translates into a different or unrecognized character. However, the chord itself gathers information that can be leveraged to improve input performance. We present B#, a novel correction system for multitouch Braille input that uses chords as the atomic unit of information rather than characters. Experimental results on data collected from 11 blind people revealed that B# is effective in correcting errors at character-level, thus providing opportunities for instant corrections of unrecognized chords; and at word-level, where it outperforms a popular spellchecker by providing correct suggestions for 72% of incorrect words (against 38%). We finish with implications for designing chord-based correction system and avenues for future work.

Author Keywords
Chord; Braille; Error Correction; Touchscreen; Mobile.

ACM Classification Keywords
H.5.2. Information interfaces and presentation

INTRODUCTION
Since the advent of Apple's iPhone and its built-in accessibility features, blind people had increased access to mainstream mobile applications. However, the flat surface poses challenges that are only partially surpassed. Particularly, typing is still slow compared to what sighted people experience [4]. To address this issue, multitouch Braille chording approaches have been presented and are very effective in improving input speed [1,6]; however, they are characterized by a decrease in typing accuracy [6].

One common approach to improve typing accuracy is the usage of spellcheckers. Indeed, there have been efforts in developing spellcheckers for disabled people, taking into consideration common typing errors and keyboard layout [3]. Further, Sandnes and Wang [5] presented a texting alternative for desktop computers, which relied on chording with spatial mnemonics and automatic word correction. However, to our knowledge, there are no reports of correction approaches that leverage Braille chording information. As a result, when decoding multitouch actions to characters, useful knowledge about the users' intent is lost. The sequence of chords provides more information about the desired word than do the characters, as each chord becomes the atomic unit of information. Distances are computed considering that a chord may be partially correct enriching the selection of the most probable suggestions.

In this paper we present B#, a correction system for multitouch Braille input that resorts to the chord itself as the basis for similarity analysis between words. At character-level, an unrecognized chord is used along with n-gram features to decide the closest character. At word-level, we extended the *Damerau-Levenshtein* distance to assess proximity between chords, and thus use this information to search for the most probable corrections. The contributions of this paper are: first, an analysis of chording errors on multitouch Braille input; second, a correction system that leverages chord-level information; and finally, an analysis on the effectiveness of character- and word-level correction.

COLLECTING TYPING DATA
We conducted a user study with 11 Braille-knowledgeable blind participants (light perception at most, ages between X-Y, 3 females) aimed at understanding the most common errors in multitouch Braille input (Grade 1). The prototype draws inspiration from BrailleTouch's [6] usage setup (Figure 1-a), combined with finger tracking techniques [1].

Procedure
Participants were given warm-up trials (10 minutes) starting from writing individual characters, words, and finally sentences. They were then instructed to write a set of 22 sentences (completion dependent of participant's availability), whereas the first two were practice trials. Error correction (delete) was not available in order to capture errors and uninterrupted typing behaviours. An auditory signal was used to indicate that a letter had been entered. Participants were instructed to continue to the next character even if they recognized they had made an error. Each sentence comprised five words, with an average size of 4.48 characters. These sentences were extracted from a

Paste the appropriate copyright/license statement here. ACM now supports three different publication options:

- ACM copyright: ACM holds the copyright on the work. This is the historical approach.
- License: The author(s) retain copyright, but ACM receives an exclusive publication license.
- Open Access: The author(s) wish to pay for the work to be open access. The additional fee must be paid to ACM.

This text field is large enough to hold the appropriate release statement assuming it is single-spaced in TimesNewRoman 8 point font. Please do not change or modify the size of this text box.

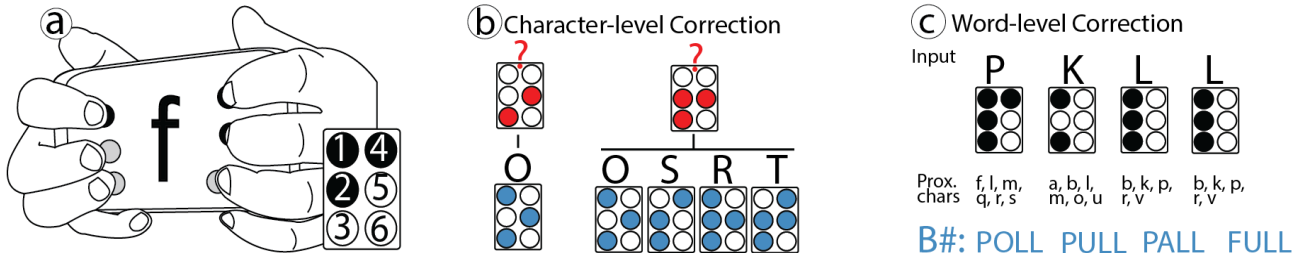

Figure 1. B# is a novel correction system for multitouch Braille input. (a) The user types the letter 'f'. (b) Character-level correction; the closest characters in terms of Braille distance for 2 unidentified chords. (c) Word-level correction; top suggestions return by B# considering the letters that are at a Braille distance of one from the entered chord.

written language corpus, each one having a minimum correlation with language of 0.97.

Major Results

Participants wrote a total of 168 sentences and 5972 characters. Data analysis revealed insights on the most prominent Braille touch typing behaviors, which were the basis for B#'s error correction algorithm principles.

High error rate. Multitouch Braille typing solutions are known to be fast and error prone [6]. Similarly, in this study, participants obtained an average typing speed of 14 words per minute and error rate of 17.7%, whereas 72.4% was due to character substitutions errors.

Single dot errors. Chord-level analysis revealed that 35.2% of substitutions were due to a single dot error within the braille cell. This difference occurred either due to a single omitted or extra dot. If we include a single adjacent finger transposition, this accounts for 42.5% of substitution errors.

Unrecognized Characters. Participants entered a total of 263 Braille chords that were not recognized as a valid letter (4.4% of all chords). In these cases, the characters are automatically classified as errors, and this information is usually discarded from traditional correction systems.

B#: LEVERAGING CHORD INFORMATION

B# deals with mistyped words that result from entering incorrect chords on a touchscreen device. In the remainder of this section, we propose: first, a distance function that enables us to compare Braille chords, and therefore predict misspells; second, a character-level disambiguation mechanism; and finally, a word-level correction algorithm that leverages chord information.

Braille Distance

Our approach to word correction is based on similarity between Braille chords. The proposed distance function aims to model users' behaviours while chording on a touchscreen. The 3x2 Braille matrix is viewed as a vector of 6 bits and thus, the difference between two vectors can be computed as a binary distance. Four types of chording errors can occur whilst chording: omissions (finger was not on the screen), insertions (additional finger on the screen), transpositions (finger was on an adjacent position), and substitutions (wrong finger on the screen).

Our distance function draws inspiration from the *Damerau-Levenshtein* distance with a slight variation. The distance between chords is the sum of the differences between hands; that is, the first and last three bits that correspond to each hand are treated independently. This approach prevents transposition errors from occurring between hands. Overall, omission, insertion, and transposition errors are considered as a one-bit error (distance of one), because these are the most common misspellings, and substitution errors (i.e. wrong fingers), are treated as a 2-bit error.

Character-Level Correction

Character-level correction is achieved by exploiting redundancy in the alphabet. The Braille cell allows the representation of 64 different patterns. However, standard text entry applications do not use its maximum capacity. If we only consider the letters of the English alphabet (26), and the blank space, then this yields 58% redundancy. In cases where the user enters an invalid chord, the closest valid character in terms of Braille distance can be selected (Figure 1-b). There are several situations where this approach retrieves more than one valid character, introducing ambiguities in the correction process. In these cases, we leverage basic linguistic features, such as letter and n-gram frequencies to select the most probable character. Our algorithm uses the maximum of previous available letters to disambiguate invalid chords. Although this approach may not deal with all spelling errors, we are interested in analyzing its effect as a stand-alone solution, as well as in combination with word-level correction.

Word-Level Correction

B# word-level correction algorithm uses a similar approach to traditional spellcheckers by matching the transcribed word against a wordlist. A key difference is that it operates at chord-level. This means that even partially correct or invalid chords can be used to retrieve better matches.

For each transcribed word, B# creates a list of likely suggestions and ranks them using a word distance score. If the word exists in the wordlist, then it is automatically the first suggestion. Our algorithm deals with three types of typing errors: insertions (unintentionally added chords), substitutions (incorrect chords), and omission (omitted chords). Transposition errors were not considered since Braille input is a sequential task with no overlap between

chord entries. A *minimum string distance* (MSD) score is used to calculate the distance between users' input and suggestion candidates. We systematically check chord permutations in the transcribed word against the wordlist.

Since computing all permutations of chords is computationally inefficient, we reduce our search space by only checking against letters that are at a Braille distance of one from the transcribed chord. This distance accounts for 1-bit errors, responsible for most character substitutions. A Trie tree structure is used for checking the existence of the word in the wordlist, while allowing to easily employing character insertion, omission, and substitution rules during the spelling correction stage. A blank space filter is used to separate two words that were transcribed without a space between them. Finally, a word frequency score is computed and subtracted from the MSD score.

B# EVALUATION

In order to evaluate B#, we ran the correction algorithms on the set of sentences previously collected. Analysis was performed using an Android Virtual Device (Galaxy Nexus). From the 751 words, we identified and tagged 364 (48.5%) incorrect entries that could be corrected by B#. We applied Shapiro-Wilkinson normality tests to observed values in all dependent variables. Parametric tests were applied for normally-distributed variables and non-parametric tests otherwise. Greenhouse-Geisser's sphericity corrections were applied whenever Mauchly's test of sphericity showed a significant effect. Bonferroni corrections were used for post-hoc tests.

Character-Level Results

Character-level correction was applied to unrecognized characters, i.e. inexistent chords in the native alphabet. To evaluate character correction, we compared B# disambiguation with the baseline substitution errors. Additionally, we included a version of B# without the linguistic features in order to assess its effect on correction accuracy. The letters' relative frequencies, bigrams, and trigrams were extracted from a corpus with more than 11 million letters and 2 million words.

Chord and Linguistic features enable Disambiguation. We obtained an average of 12.8% (sd=8.6%) substitution errors. By only exploiting the redundancy of the Braille alphabet, substitution errors slightly decreased to 12.2% (sd=7.3%). On the other hand, B# character-level disambiguation successfully corrected 2.4% of substitution errors, reaching an average of 10.4% (sd=5.9%). A Friedman test revealed a significant effect on methods [$\chi^2_{(2)}$=18.474, p<.001]. Wilcoxon pairwise comparisons showed significant differences between B# disambiguation and both baseline [p<.005, r=.2] and redundancy-only [p<.01, r=.1] results. As 33.6% of substitution errors were due to unrecognized characters, it means that 18.8% of all chord ambiguity errors were automatically corrected combining redundancy and simple language features. Overall, B# disambiguation corrected 29 words (8% of all incorrect words).

Word-Level Results

In order to evaluate B# word-level correction, we compare our solution with a popular spellchecker for mobile devices: Android (AOSP) spellchecker, which for brevity purposes will herein be referred to as AOSP. This solution makes no use of contextual information, similarly to B#. Additionally, to conduct a fair comparison between correction methods, we used the same wordlist (213,133 words) and word frequencies. All required words used in the data collection stage existed in the wordlist. To assess the effect of each correction component, the results presented here were obtained without character-level disambiguation. We also analyze a variant of B# that uses the Hamming distance (XOR between binary vectors) to compute similarity between chords enabling us to assess our effectiveness against a commonly used metric to compare vectors.

Method	Top 1	Top 2	Top 3	Top 4	Top 5
AOSP	28 (12)	34 (12)	36 (13)	37 (13)	38 (13)
Hamming	53 (19)	63 (19)	66 (21)	62 (22)	69 (22)
B#	56 (17)	65 (18)	69 (20)	70 (21)	72 (20)

Table 1. Mean percentage (standard deviation) of correct words returned for the top n suggestions (1≤n≤5).

B# is consistently more accurate. Table 1 shows the results for the top *n* suggestions (*1≤n≤5*), while Figure 2 illustrates the results. The percentage of accurate words returned by the AOSP spellchecker is rather small and never exceeds 38%. On the other hand, chord-based solutions outperform the AOSP spellchecker, with mean accuracy rates between 53% and 72%. Indeed, when analyzing the first suggestion, there is a significant effect [$F_{(1.102,11.021)}$=19.279, p<.001, partial η^2=.66], with both B# [p<.005] and Hamming [p<.01] methods being more accurate than AOSP spellchecker. This result suggests that leveraging chord information plays a major effect on correction accuracy. No significant effect between B# and Hamming variant was found for the first suggestion. Overall, B# would be able to correct 204 of 364 incorrect words automatically, i.e. without user intervention. The Hamming variant and AOSP would correct 193 and 100 words, respectively.

B# is more accurate for the top three suggestions. While we agree that the most important result is to propose the correct word as the first suggestion, we also acknowledge that providing the highest possible fraction of accurate suggestions at the top of the list is critical, especially, when user intervention is needed. Therefore, although B# and Hamming variant methods are equally accurate for *n=1* and *n=2*, B# significantly outperforms its counterpart for *n=3* [Z=-2.023, p<.05, r=0.08], *n=4* [Z=-2.023, p<.05, r=0.11], and *n=5* [$t_{(10)}$=-2.25, p<.05, Cohen's d=.17].

No gain beyond top five suggestions. We can observe in Figure 2 that the highest gain is achieved from *n=1* to *n=2* with an increase of 6.6%, 9.5%, and 9.5% for AOSP spellchecker, Hamming variant, and B#, respectively. For

n>2 improvements are smaller and then start to fade at *n=5*. B# and Hamming curves are very similar, which can be explained by the similar correction paradigm; however, Hamming is consistently outperformed by B#. This result suggests that indeed the proposed extension to the *Damerau-Levenshtein* distance is effective in dealing with Braille chording misspellings. B# continues to provide useful suggestions for *n≤5*, with significant differences for *1≤n≤2* [Z=-2.803, $p<.005$, r= 0.26], and *2≤n≤3* [Z=-2.201, $p<.05$, r= 0.09], *4≤n≤5* [Z=-2.023, $p<.05$, r= 0.05].

B# provides a low number of incorrectly corrected words. All methods obtained low false positives (top suggestion incorrect): 1.4% (sd=1.9%), 1.7% (sd=2.2), and 1.7% (sd=2.2). In fact, a Friedman test revealed no significant effect on method [$\chi^2_{(2)}=2$, $p=.368$]. B# high accuracy comes at no cost regarding incorrectly corrected words.

Chord information matters. In order to measure how important chord-level information is on providing accurate suggestions, the word frequency factor was removed from our spellchecker. This means that the scoring solely depended of Braille chord similarity. Results showed an accuracy rate of 44.8% (sd=15%) for the top suggestion, which is a significant decrease of accuracy (*n=1*) from the B# condition. This result shows that word frequency features are important to obtain the best possible suggestion. Despite the decrease in performance, this condition still outperforms AOSP's spellchecker [$t_{(10)}=3.182$, $p<.01$, Cohen's d=1.3], suggesting that taking advantage of the chord similarity is crucial to correct errors.

Combined character-word-level correction is ineffective. We also assess the effect of feeding disambiguation results into the word-level component. Overall, the combined solution presents a similar accuracy curve to word-level correction; however, it is outperformed for all *1≤n≤10*. A paired t-test showed a significant effect for methods [$t_{(10)}=3.080$, $p<.05$] with the word-level correction outperforming the combined solution for *n=1* (m=50.1%). This suggests that erroneous corrections at character-level affect word-level accuracy. The spelling component loses important chord information that would otherwise be used.

IMPLICATIONS FOR DESIGN

Leverage input method information. Results showed that leveraging chord distances significantly improves word correction accuracy. Even simple distance functions allow improvement over traditional spellcheckers. As previously suggested [2], one should take advantage of common misspellings, tightly related to each method, and use that knowledge to improve spelling accuracy.

Do not discard chord information. Data showed that combining character- and word-level is an ineffective solution, mostly because information is lost in the first correction stage. Therefore, if using character-level correction to provide instant feedback to users, allow the word-level component to receive the original chords.

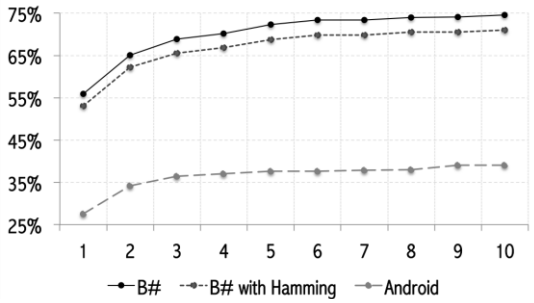

Figure 2. Mean percentage of correct words returned for the top n suggestions (1≤n≤10)

Focus on top 5 suggestions. Most spellcheckers provide a list of most probable suggestions enabling users to intervene when corrections are inaccurate. Results showed that B#'s top two suggestions yield most corrections; however, if possible, the top five should be considered.

CONCLUSION AND FUTURE WORK

We presented B#, a chord-based spellchecker for multitouch Braille input. Results show that by leveraging a distance of similarity between chords B# outperforms other spellcheckers. These results should encourage researchers to develop correction algorithms tailored to novel text-input techniques. Future work will explore the design of non-visual interfaces to leverage chord-based suggestions.

ACKNOWLEDGMENTS

This work was supported by RCUK Digital Economy Programme grant number EP/G066019/1 – SIDE, FCT through the Multiannual Funding Programme and individual grant SFRH/BD/66550/2009. We thank all participants in the study for their time.

REFERENCES

1. Azenkot, S., Wobbrock, J., Prasain, S., Ladner, R. Input finger detection for nonvisual touch screen text entry in Perkinput. In *Proc. GI* (2012).
2. Deorowicz, S., & Ciura, M. G. Correcting spelling errors by modelling their causes. *Int. journal of applied mathematics and computer science* (2005), *15*(2), 275.
3. Kane, S., Wobbrock, J., Harniss, M., & Johnson, K. TrueKeys: identifying and correcting typing errors for people with motor impairments. In *Proc IUI* (2008).
4. Oliveira, J., Guerreiro, T., Nicolau, H., Jorge, J., Gonçalves, D. Blind people and mobile touch-based text-entry: acknowledging the need for different flavors. In *Proc. ASSETS* (2011).
5. Sandnes, F. E., Huang, Y. P. Chording with spatial mnemonics: automatic error correction for eyes-free text entry. In *Journal of information science and engineering*, 22, 5 (2006).
6. Southern, C., Clawson, J., Frey, B., Abowd, G., Romero, M. An evaluation of BrailleTouch: mobile touchscreen text entry for the visually impaired. In *Proc. Mobile HCI* (2012).