Interpreting Gestures for Text Entry on Touch Screen Devices

Gennaro Costagliola, Vittorio Fuccella
University of Salerno
Via Ponte Don Melillo
84084 Fisciano (SA), Italy
{gencos, vfuccella}@unisa.it

Michele Di Capua
Unlimited Software
Centro Direzionale, Isola F/11
80143 Napoli, Italy
md@unlimitedsoftware.it

Abstract

Text entry on touch screen devices is often performed through Soft keyboards. One of the latest research trends is to abandon the traditional tapping interaction in favor of more natural gesture-based interactions on these keyboards. The interpretation of the gestures is performed through sketch-based techniques.

In this paper we present the sketch-based technology related to the interpretation of the gestures needed to enter text through KeyScretch, a novel text entry method which has been proven to be more efficient than the traditional tapping-based method. The method enables text entry through the input of gestures associated to word chunks. The gestures are guided by a menu which appears around a key of the keyboard as soon as the user presses it.

The proposed procedures improve the interpretation of the gestures and consequently increase the accuracy of the method. This improvement is measured through a simulation in which the sketch-based recognition is compared to the target-based menu item selection through an estimation of the error rate.

1 Introduction

The efficiency of text entry on mobile devices is limited due to the reduced dimension of these devices. In most cases, palmtop and smart phones equipped with touch screens require the users to interact with a *soft keyboard*, a keyboard drawn on the screen. The main method of interacting with these keyboards is *tapping*. Each *tap* corresponds to a single character input. The latest research trend is to abandon such an interaction in favor of a more natural gesture-based interaction performed on these keyboards. As an example of this, we can cite *Shapewriter* [15], in which the user draws over a keyboard, i.e. with a *QWERTY* layout, to link up the letters of a word s/he wishes to write.

Recently, Costagliola et al. [2] introduced a novel text entry method, called *KeyScretch*. The method uses a combination of soft keyboards and menus: a menu is shown around the pressed key enabling the input of a text unit through a gesture touching the menu items associated to the characters. *KeyScretch* is based on previous works: the use of menu-augmented keyboards has been studied by Isokoski [5]. In its basic form, the use of the menu enables the input of digraphs through a single interaction, called a *flick*. The improvement introduced with *KeyScretch* is the support of multiple-selection of menu items with a pointer gesture. The use of these gestures allows the user to enter particularly frequent text chunks instead of simple digraphs. *KeyScretch* outperforms the traditional tapping-based method and significantly improves the previous menu-based one too.

The interpretation of the gestures is performed through sketch-based techniques. In this paper we present the sketch-based technology related through the interpretation of the gestures needed to enter text through *KeyScretch*. These techniques provide a better interpretation of the user gestures, compared to the target-based menu item selection, and consequently relax the error rate. The error reduction has been estimated by running the sketch-based recognizer on the tracks produced by the user gestures during a longitudinal usability study. For both the sketch-based and the target-based methods, the error rate is calculated and reported for each session of the study. In this paper we give details of the procedures employed by the sketch-based recognizer, based on geometrical sketch recognition techniques.

The rest of the paper is organized as follows: the next section contains a brief survey of the main text entry methods related to touch screens; section 3 describes the text entry method and its performances; section 4 discusses the proposed recognition model in detail; lastly, the evaluation of the model is described in section 5.

2 Text Entry Methods: A Brief Survey

Several methods for accelerating the text entry task have been proposed. In this brief survey we only consider text entry methods related to touch screens. For a more comprehensive survey on text entry on mobile devices, the reader should refer to [10]. Some of the main directions followed by researchers are:

- handwriting and shorthand writing recognition;

- proposal of *more efficient* keyboard layouts;

- prediction of text;

Handwriting recognition is not an easy task and raises several challenges, such as segmentation [10]. Furthermore, handwriting is rather slow: its speeds are commonly in the 15 - 25 WPM range. To ease the recognition and avoid segmentation, the methods for handwriting on mobile devices have been limited to the recognition of one character at a time. Examples of these methods are *Unistrokes* [3] and *Graffiti* [1]. As for *shorthand*, it allows users to reach high performances in terms of speed (up to 100 WPM [9]). Nevertheless it appears unlikely that common users could adopt such a difficult to learn method. A method allowing common users to easily learn the symbols is that of using schemes indicating the path the pointer should follow, such as the *pentagrid* used in the *VirHKey* method [13].

The *QWERTY* layout is the most familiar to the users. Unfortunately, it is not the most efficient with *tapping*. An efficient keyboard layout should minimize the distance between characters with a high probability of being consecutive in the words of target languages. To this aim, alternative layouts have been proposed, such as *OPTI* [11], *Metropolis* [16] and others. These layouts enable a faster and more accurate entry (more than 40 WPM with expert users), compared to the traditional *QWERTY* layout.

Predictive input methods reduce the effort required to enter text by predicting what the user is entering. A typical predictive handwriting system presents possible next words as a list and allows the user to select one to skip manual writing. As argued in [8], predictive text entry with English language is not necessarily faster than just simply finishing typing the words, since it requires the cognitive load of selecting the candidates from a list.

A noteworthy attempt to improve the interaction with soft keyboards through gestures, implemented by the *SHARK2* system [6], is to use a single gesture to produce an entire word. Instead of *tapping*, the user draws over a keyboard, i.e. with a *QWERTY* layout, to link up the letters of a word s/he wishes to write. This idea has been also exploited in commercial initiatives [15]. This approach let the user learn the gestures for common words and draw them without looking at the keyboard. The gesture is initially visually guided (for novice users): the pointer must touch the characters located on the keyboard in order to enter a word. Then, when learned, it is recognized through the comparison of the drawn shape (pattern recognition) to a vocabulary of gestures. The limit of this approach is that only a small set of frequent gestures can be remembered by the user.

3 The *KeyScretch* Text Entry Method

The *KeyScretch* method enables text entry through the input of gestures associated to word chunks instead of entire words. The gestures are guided by a menu which appears around a key of the keyboard as soon as the user presses the key and disappears as soon as the user releases the pointer.

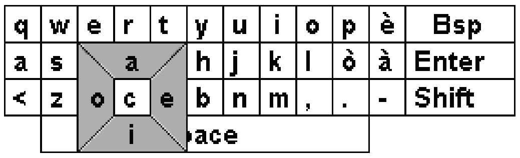

Figure 1. The QWERTY keyboard layout augmented with a menu.

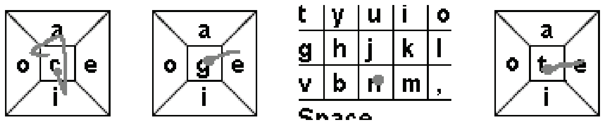

Figure 2. The 4 strokes needed to enter the text 'ciao gente'.

While the menu is shown, it is possible to sweep out a gesture that touches only the desired characters in succession by dragging the pointer, without lifting it (this gesture recalls the one used for the *Cirrin* method [12]). A space character can be inserted by releasing the pointer after returning inside the character key area. If the pointer is lifted in the menu area, instead, the text unit is terminated with the last selected character. Note that in all the other cases the space bar is directly used.

KeyScretch can be instantiated differently for different languages. As a first instance, we have associated each of the vowels 'a', 'e', 'i' and 'o' to each of the sides of the squared character keys. As shown in [2] this choice is particularly suited for the Italian language since these vowels are the most frequent letters in Italian and, furthermore, they are more easily remembered by users. Different menu characters may be needed by different languages. By arranging the menu items as shown in Figure 1, the interaction sequence necessary to enter the Italian text *ciao gente* (*hello folks*, in English) is shown in Figure 2. The string is ten characters long but it can be entered with a sequence of four strokes (*taps* or *gestures*). The strokes correspond to the input of the following sequence of text units {*ciao* }{*ge*}{*n*}{*te*}. Three text units out of four are entered through a gesture and only one through a *tap*. With the first stroke we can enter up to five characters.

3.1 Performances

A theoretical analysis on the expected performances in terms of speed obtainable by an ideal user has been performed in [2]. Furthermore, a longitudinal usability study with 6 users has also been recently performed. The methodology of this experiment follows the standard procedures used in the field [11]: the users entered text on a touchscreen with both methods for 20 sessions, for 15 minutes with each method. During the sessions the users entered as fast and as accurate as possible short text phrases, selected in a random order from an 80 Italian phrases database representative of the language.

The study allowed us to evaluate the learning curve of both the traditional and the *KeyScretch* method. The scatter diagrams of the two metrics, speed and accuracy, are shown Figure 3. As for the former, we can point out that the performance crossover can be reached rather soon (close to the 10th session, after about 2 hours and a half of use) across the sessions and the *KeyScretch* method can outperform the traditional tapping-based method in the subsequent sessions and, all the more so, in the estimation of future sessions. The best average speed with the *KeyScretch* method has been reached at the 19th session, with 42.7 WPM with a moderate error rate of 3.35%. The fastest user has exceeded the the considerable speed of 52 WPM at the last session with an error rate of 3.31%. For each method, we derived standard regression models in the form of the power law of learning. The prediction equations and the squared correlation coefficients are illustrated the figure.

As for the accuracy, we point out that, preserving the status quo, our method has a lower accuracy, on average. In fact, the *KeyScretch* method has an error rate of 3.18%, versus the 1.93% of the traditional method. Nevertheless, the difference is more marked in the earlier sessions and tends to diminish with practice. Since this paper is more focused on the methods for sketch recognition, more details on the usability study are going to be published in an upcoming paper.

4 The Sketch-Based Recognition Model

In the user study described in the previous section, the recognition of user gestures has been performed through a target-based recognizer: the initial character of the text unit is selected by detecting the position where the pointer is pressed; the following characters are selected when the pointer exactly enters a menu item area; the possible final space is selected by detecting the position where the pointer is released.

An exploratory visual analysis of the errors committed by the users during the use of the menu has been carried out by comparing the track of the gestures on the keyboard and the corresponding produced text. From such an analysis, we have realized that:

- Most of the errors are committed in the use of the menu (73.5%) rather than in tapping interactions. Furthermore, the greatest part of them is due to articulatory concerns (43.3% of all errors) instead of cognitive ones. That is, the errors are primarily due to inaccuracy in the physical action necessary to select an item from the menu and not to mistakes of the user in remembering the position of a character in the menu.
- It is possible to recover from most of the articulatory errors by better interpreting the user gestures. In particular, a geometrical recognition of the basic movements intended by the users can have the best performances over other recognition methods.

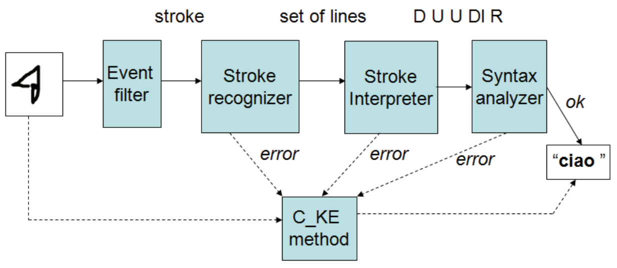

Figure 4. The data flow.

In this section we describe the sketch-based procedures carried out for the interpretation of the gestures. The sketch-based recognizer basically works as follows: the recognizer tries to obtain the text chunk. If it fails (it can fail because the stroke cannot be recognized as a polyline, or the lines obtained do not represent a legal movement inside the menu), the stroke is recognized through an optimized version of the target-based C_KE method, where C stands for *Centering* while KE stands for *Key Enlargement*. In fact, the optimization lies in translating the gesture such that its starting point is centered in the key. Furthermore, the area available to the user to end the stroke inside the squared key (to enter a space) is enlarged of a margin in both its width and height.

To elaborate, the architecture of the recognizer is shown in Figure 4. The data flows through the following modules:

1. The *Event Filter*: captures the input events of the device and constructs the stroke;
2. The *Stroke Recognizer*: transforms the stroke to a polyline;
3. The *Stroke Interpreter*: transforms the polyline to a set of movements among the menu items;
4. The *Syntax Analyzer*: transforms the movements to a text chunk.

4.1 Event Filter

This module captures the input events of the device (pointer *press*, *drag* and *release*) and constructs the stroke. Following a common approach in sketch recognition, the stroke is represented through an ordered sequence of *TimedPoints*: triples in the form (t, x, y), where t is the timestamp, and x and y are the x-coordinate and the y-coordinate of the point on the screen, respectively. In the simplest scenario, the stroke can be easily constructed by adding a *TimedPoint* for all of the *drag* events between a starting *press* and a final *release* events. This layer also includes methods which cope with defects of the device: in particular, the device used during the experiment outlined in section 3.1 (A *Sympodium ID250* Interactive Pen Display), sometimes generated undesired additional events during the use of the pointer. Thus, the events had to be suitably filtered.

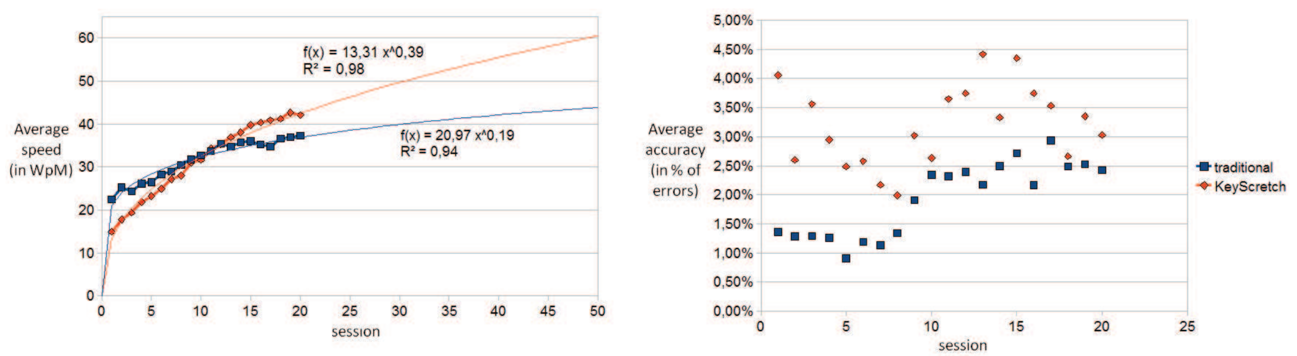

Figure 3. The text entry speed (left) and accuracy (right) measured across the 20 sessions.

4.2 Stroke Recognizer

Firstly, the stroke recognizer takes the strokes from the *Event Filter* and produces a polyline. A polyline is a continuous line composed of one or more line segments. The *Ladder* framework [4] has been used for recognizing the polylines. Ideally, each segment should represent the movement of the pointer to reach the desired menu item. Without firstly cleaning the stroke, this rarely occurs: we then implemented the *Line Fusion* and *Tail Removal* steps to refine the polyline before passing it to the *Stroke Interpreter*. An example of the recognition of a polyline from a stroke is shown in Figure 5(a) and (b) where the menus are not shown. The original stroke is represented in light gray color, the polyline in black. The resulting polyline as recognized by *Ladder* is composed of 6 segments: $b_1..b_6$.

The *Line Fusion* step finds segments which cannot be considered complete movements to reach a menu item, but parts of a unique movement (they are recognized as separate segments due to curvature). The fusion is always performed on the segment of the polyline whose angle with the previous segment is the minimum on the whole polyline. The fusion is performed only if such an angle amplitude is less than a threshold value. The application of the *Line Fusion* procedure on the sample stroke in Figure 5(b) is shown in Figure 5(c): starting from the $b_1..b_6$ polyline, we obtain a simplified polyline with only 4 segments $c_1..c_4$. In two consecutive iterations the procedure fuses in a single segment c_3 segments b_3, b_4 and b_5 while the tails c_1 and c_4 are still to be processed.

The tail removal is a typical procedure in sketch recognition. Paulson and Hammond [14] propose a procedure based on the length and the curvature of the first and last 20% of the stroke. In our procedure we have not considered the percentage length of the segment over the stroke at all, since even a simple tap can result in an unintended snatch on a key, which can be erroneously interpreted as a menu item selection (see Figure 8(c)). We based our procedure on the absolute length of the segment and the difference between the timestamps of the segment endpoints. If the segment length is less than a threshold *l* or the time difference is less than a treshold *t*, the segment is regarded as a tail and removed. This is the case of the segments c_1 and c_4 shown in Figure 5(c). Their removal leads to the resulting clean polyline in Figure 5(d).

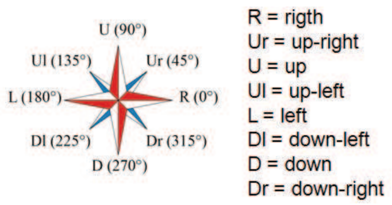

Figure 6. The directions attributed to the polyline segments.

4.3 Stroke Interpreter

The *Interpretation* step transforms the clean polyline representing the calculated user intended character selections in a sequence of movements among menu items. The sequence is represented through a string composed of a concatenation of strings representing possible movements in the 8 main directions shown in Figure 6. The slope of a segment is approximated to the nearest direction. The slopes of the directions are shown in the figure. As for the sample in Figure 5(d), the two segments are attributed the directions 'R' and 'Dl'; thus, the final string representing the interpretation of the stroke is 'RDl'.

If a horizontal segment is approximately (a multiplying factor must be set as a threshold) twice the first segment of the polyline, the movement is doubled (i.e. 'R' becomes 'RR'), in order to interpret the movement from a menu item to the opposite (left-to-right, right-to-left, top-to-bottom or bottom-to-top).

4.4 Syntax Analyzer

In order to formally specify the language describing the set of legal movement strings we define the automaton state diagram represented in Figure 7. Here the states represent

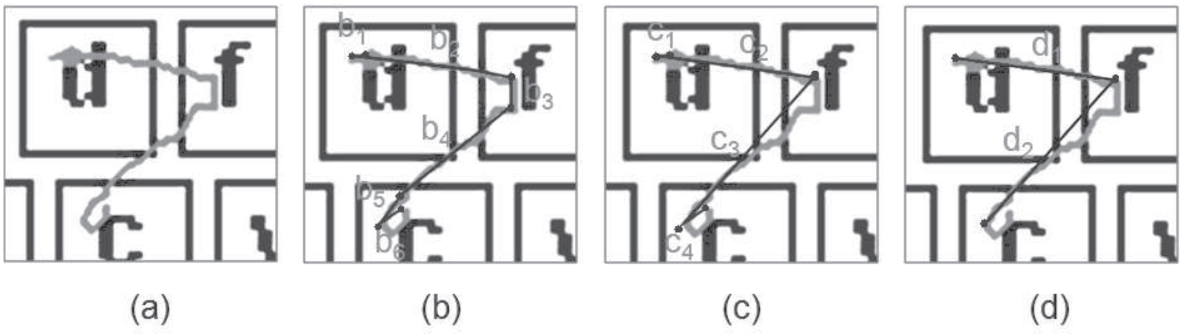

Figure 5. A sample of the Stroke Recognition process.

the 4 menu items (q_1, q_2, q_3, q_4) and the key (q_0), while the transitions represent the legal movements that may be produced from each state. All the states are final and q_0 is also the initial state. In order to formally specify the translation from movement strings into text chunks we build a Moore machine starting from this automaton. The machine associates each of the $c_1..c_4$ states to the output of a character. In particular, in the tested instance it associates q_1 to the output 'a', q_2 to the output 'e', q_3 to the output 'i', and q_4 to the output 'o'. To reflect our semantics we need to split state q_0 in three states q_{01}, q_{02}, and q_{03} where:

1. q_{01} is both the initial and a final state associated to the key character as output, and with only the 4 q_0 outgoing transitions;
2. q_{02} is a non final state associated to no output (empty word) and with both the 4 outgoing and 4 incoming transitions of q_0;
3. q_{03} is a final state associated to the 'space' character as output and with only the 4 q_0 incoming transitions.

As an example, by producing the legal movement string 'DUUDlR' starting from the key 'c', we obtain the sequence of Moore machine states q_{01}, q_3, q_{02}, q_1, q_4, q_{03} producing the output text chunk 'ciao '.

5 Evaluation

The sketch-based recognizer described in the previous section can translate a gesture into a legal sequence of movements among the menu items in 95.09% of the cases (from the data of the longitudinal study outlined in section 3.1). In the remaining 4.91% cases, the recognition must be performed through the *C_KE* target-based method, described in the previous section.

Some cases of better interpretation of the user's will are shown in Figure 8. A tail removal is shown in Figure 8(a) the tail caused a re-enter in the key area, interpreted as a trailing space by the previous target-based recognizer. The sketch-based recognizer has correctly recognized the movement as composed of a successive selection of the bottom and the right menu items ('DUr'). The movements shown in Figure 8(b) were interpreted as a sequence of a movement to the right followed by a movement to the left menu item, due to an excessive amplitude of the return movement. The comparable length of the two lines has been correctly interpreted by the sketch-based recognizer as a movement to the right menu item followed by a return to the key area ('RL' instead of 'RLL'). Lastly, the gesture in Figure 8(c) is a simple tap. Nevertheless, the presence of a tail caused an erroneous interpretation of the gesture as a selection of the right menu item. The tail has been removed and the tap has been correctly recognized.

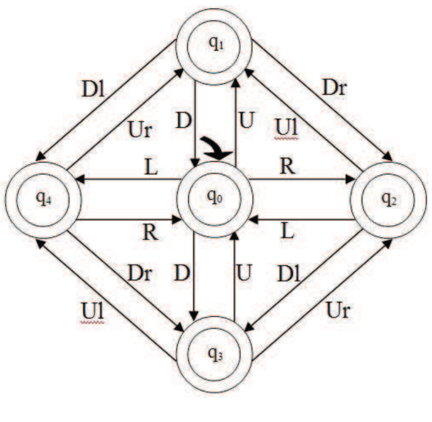

Figure 7. The automaton state diagram used to recognize the movement string.

The translation of a gesture in movement among menu items does not necessarily produce a correct text chunk, that is, the text can contain typing errors, due to user incorrect typing or to incorrect recognition. The effectiveness of the sketch-based recognizer has been measured by comparing the error rate obtained by the target-based recognizer in the longitudinal study, to the one that could have been obtained by the sketch-based recognizer. For the latter, the text entry has been simulated through a suitable software module which implements the recognizer: it takes as input the logged tracks and generates the corresponding phrases.

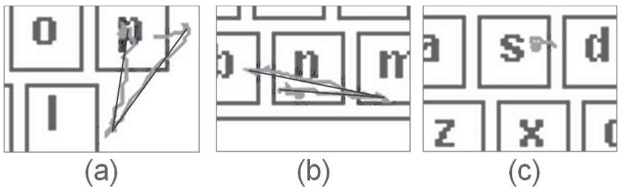

Figure 8. Some cases of correct interpretation of the sketch-based recognizer.

This approach (the use of a simulation run on the data previously gathered in an experiment) is based on previous work: Kristensson and Zhai [7] run a simulation on pen traces recorded from participants in order to show accuracy improvement on *SHARK2* [6] text entry method.

The recognizer has been tuned by setting its parameters on 4 out of the 20 sessions (the 3rd, 8th, 13th and 18th) and tested on the remaining 16 sessions. The improvement obtained on the training data, that is the percentage of error reduction on the 3rd, 8th, 13th and 18th sessions was 32.59%, 8.55%, 35.03%, 15.08%, respectively, with an average of 22.81%. The error reduction on the test set was even better, with an average of 24.75%. Overall, the use of the sketch-based recognizer redured the error rate from 3.18% to 2.20%, thus making the accuracy of *KeyScretch* comparable to the one measured for the traditional tapping-based method from the 9th/10th session on. An analysis of variance revealed a significant difference in error rates between the two recognizers ($F_{1,5} = 68.1, p < .0001$).

To be sure that the sketch-based technique is effective also with other users than those with which it has been tuned, we ran another user study. In this experiment, 6 users performed text entry only with the *KeyScretch* method and for just one session of 15 minutes. 3 users were completely novices, while 3 of them had some previous experience with the method occurred about 3 months before this experiment, quantifiable in about 2 hours of practice. The recognizer, applied on the novice users, produced an error reduction of 10.46%, and a reduction of 13.07% on the 3 lightly experienced users. These values are in the range obtained on the tracks of the longitudinal study, even if they are under their average values.

6 Conclusion

In a previous research we have defined the new text entry method *KeyScretch*. Both theoretical and usability studies show that *KeyScretch* has good performances from the point of view of the speed, outperforming the tapping-based method.

In this paper we presented the sketch-based technology related to the interpretation of the gestures needed to enter text through *KeyScretch*. The proposed procedures improve the interpretation of the gestures and consequently increase the accuracy of the method. The evaluation shows that geometric sketch recognition techniques, associated to other calibrations can improve the accuracy of the method and make it comparable to the one measured for the traditional method just after a few hours of use. Future work include: further studies aimed at investigating the possibility of achieving improvements on the recognizer, in order to further tune the accuracy; the comparison of the presented recognizer with other recognizers based on different techniques (i.e. pattern matching); the exploitation of other methods (i.e. dictionary-besed ones) to correct errors. A demo of *KeyScretch* can be run at
`http://keyscretch.altervista.org/KeyScretch.html`

References

[1] C. H. Blickenstorfer. Graffiti: Wow!!!! *Pen Computing Magazine*, pages 30–31, 1995.
[2] G. Costagliola, V. Fuccella, M. Di Capua, and G. Guardi. Performances of multiple-selection enabled menus in soft keyboards. In *Proceedings of DMS'09 (VLC Workshop)*, pages 359–364, 2009.
[3] D. Goldberg and C. Richardson. Touch-typing with a stylus. In *Proceedings of INTERCHI '93*, pages 80–87, Amsterdam, The Netherlands, The Netherlands, 1993. IOS Press.
[4] T. Hammond and R. Davis. Ladder, a sketching language for user interface developers. *Comput. Graph.*, 29(4):518–532, 2005.
[5] P. Isokoski. Performance of menu-augmented soft keyboards. In *Proc. of CHI '04*, pages 423–430, New York, NY, USA, 2004. ACM.
[6] P.-O. Kristensson and S. Zhai. Shark2: a large vocabulary shorthand writing system for pen-based computers. In *Proc. of UIST '04*, pages 43–52, NY, USA, 2004. ACM.
[7] P. O. Kristensson and S. Zhai. Improving word-recognizers using an interactive lexicon with active and passive words. In *Proceedings of IUI '08*, pages 353–356, New York, NY, USA, 2008. ACM.
[8] K. Kurihara, M. Goto, J. Ogata, and T. Igarashi. Speech pen: predictive handwriting based on ambient multimodal recognition. In *Proceedings of CHI '06*, pages 851–860, New York, NY, USA, 2006. ACM.
[9] Y. Ma, G. Leedham, C. Higgins, and S. M. Htwe. Segmentation and recognition of phonetic features in handwritten pitman shorthand. *Pattern Recogn.*, 41(4):1280–1294, 2008.
[10] 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.
[11] I. S. MacKenzie and S. X. Zhang. The design and evaluation of a high-performance soft keyboard. In *Proc. of CHI '99*, pages 25–31, New York, NY, USA, 1999. ACM.
[12] J. Mankoff and G. D. Abowd. Cirrin: a word-level unistroke keyboard for pen input. In *Proc. of the ACM UIST 98*, pages 213 – 214. ACM, 1998.
[13] B. Martin. Virhkey: a virtual hyperbolic keyboard with gesture interaction and visual feedback for mobile devices. In *Proceedings of MobileHCI'05*, pages 99–106, New York, NY, USA, 2005. ACM.
[14] B. Paulson and T. Hammond. Paleosketch: accurate primitive sketch recognition and beautification. In *Proceedings of IUI '08*, pages 1–10, New York, NY, USA, 2008. ACM.
[15] Shapewriter. http://www.shapewriter.com/, 2010.
[16] S. Zhai, M. Hunter, and B. A. Smith. The metropolis keyboard - an exploration of quantitative techniques for virtual keyboard design. In *Proc. of UIST '00*, pages 119–128, New York, NY, USA, 2000. ACM.