Error-proof, High-performance, and Context-aware Gestures for Interactive Text Edition

Luis A. Leiva, **Vicent Alabau**, **Enrique Vidal**
ITI/DSIC, Universitat Politècnica de València
Camí de Vera, s/n – 46022 Valencia (Spain)
{luileito,valabau,evidal}@{iti,dsic}.upv.es

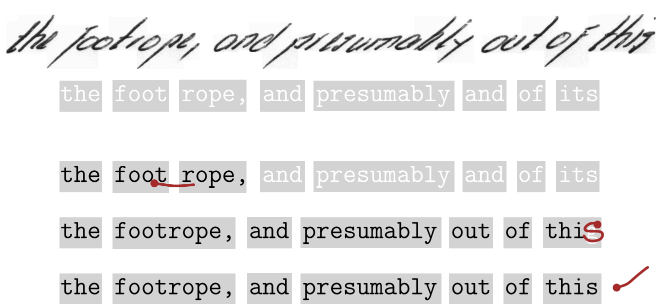


Figure 1: Transcribing handwritten text on a mixed-initiative UI, where the user and the system efficiently cooperate to produce an error-free result. As observed, only 2 operations are needed to achieve the correct transcription, as long as the submitted strokes are correctly being classified either as (the right) gestures or handwritten text when necessary.

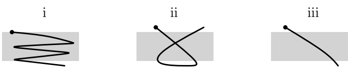
CHI 2013 Extended Abstracts, April 27–May 2, 2013, Paris, France.
ACM 978-1-4503-1952-2/13/04.

Abstract
We present a straightforward solution to incorporate text-editing gestures to mixed-initiative user interfaces (MIUIs). Our approach provides (1) disambiguation from handwritten text, (2) edition context, (3) virtually perfect accuracy, and (4) a trivial implementation. An evaluation study with 32 e-pen users showed that our approach is suitable to production-ready environments. In addition, performance tests on a desktop PC and on a mobile device revealed that gestures are really fast to recognize (0.1 ms on average). Taken together, these results suggest that our approach can help developers to deploy simple but effective, high-performance text-editing gestures.

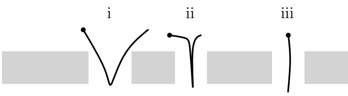
ACM Classification Keywords
H.5.2 [**Information Interfaces and Presentation**]: User interfaces—*Input devices and strategies, interaction styles*

Author Keywords
Text Editing; Gestures; UI Prototyping; Pointing Devices

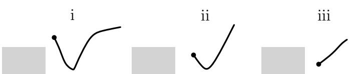
Introduction
Gesture-based interfaces provide the user with a direct, natural form of interaction. Together with the popularity of stroke-based devices (e.g., touchscreens, e-pens, styli, tabletops, or surfaces), accurate gesture recognition and suitable prototyping tools are becoming essential. Within



(a) Delete



(b) Insert



(c) Validate

Figure 2: One of the goals of this work is to simplify stroke candidates. Here, examples for deletion [2a], insertion [2b], and validation [2c] gestures are shown. Ours is the third option (minimum effort stroke).

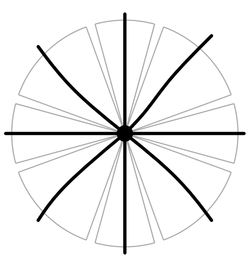


Figure 3: As a result of the aforementioned simplification goal, we ended up in reducing gestures to their minimal expression: 1D lines.

this context, text-editing applications are increasingly being enhanced with gestures, specially those applications that follow a mixed-initiative user interface (MIUI) principle [2], i.e., those in which the user and the system collaborate efficiently (Figure 1). For instance, CueTIP [8], CATTI [7], and IMT [6] are recent examples of text-editing applications with MIUIs partially commanded by gestures. In these systems, the user iteratively refines some automatic system output (or hypothesis), by providing corrective feedback that the system leverages to produce a better hypothesis.

Current Challenges

This work aims to solve three open problems when editing text on MIUIs. First, gestures and handwritten text must be unambiguously differentiated. Otherwise, if a gesture is misrecognized as text (or vice versa), cascading errors are likely to happen, so *a)* the actual user intention would be wrongly captured by the application; therefore *b)* it would not be possible for the system to derive a correct response; which *c)* would cause frustration, as *d)* the user would need to amend the erroneous response and resubmit the previously intended gesture (or text correction) again. Second, it is notably important to ensure both low recognition errors *and* low recognition times, since productivity is extremely mandatory when operating a text-editing MIUI. In this regard, users are typically willing to accept error rates up to about 3% or less, before deeming the technology as “too encumbering” [3]. Finally, it is mandatory for the system to know the *context* of an issued gesture, i.e., the application need to know information from the text itself which the user is interacting with, at the word (or even character) level, in order to provide the user with suitable corrections. Thus, on a text-editing MIUI, gestures must be performed over the text being edited.

The above-mentioned open problems constrain the design of the gesture vocabulary. Moreover, each type of application has unique operations and therefore requires specialized gestures. Even more, gestures are limited both by human memory and user performance, and hence they must remain simple. Figure 2 illustrates these ideas. Inspired in part by the Marking Menus techniques [1], our approach, named MinGestures, is based on the fact that drawing lines (1D gestures) with a pointer device is a very simple task and really easy for users to perform, but it also should be very efficient for computers to recognize, since the proposed gestures are linearly separable. This paper therefore provides a well-defined balance to deploy text-editing gestures on MIUIs.

Implementation

We first tried to implement our baseline set of eight 1D gestures (Figure 3) using state-of-the-art recognizers, among which we chose [5, 9, 10] for being easily customizable. Concretely, only \$1 [10] would partially fit our needs. Protractor [5] is a faster version of \$1, as a result of rotating gesture samples to their optimal indicative angle prior and during recognition, and therefore it is not appropriate to deal with our full set of 1D gestures. \$P [9] is another version of \$1 in which gestures are treated as clouds of points, so it cannot differentiate gestures on the basis of direction.

As reported in Pilot Study, after incorporating some tweaks to \$1, overall recognition error was around 2%, which was encouraging but perhaps still insufficient for editing text on MIUIs. Therefore, considering the simplicity of MinGestures, we opted for implementing a customized *parametric* recognizer, since target gestures must fit an assumed *model* (straight lines). Otherwise, the strokes should be identified as handwritten text.

Figure 4 provides a graphical overview of a proposed gesture set based on MinGestures, which, from our experience developing MIUIs (e.g., [6, 7]), incorporates essential operations to handwritten text and, at the same time, adequately solves the three open problems discussed in the previous section.

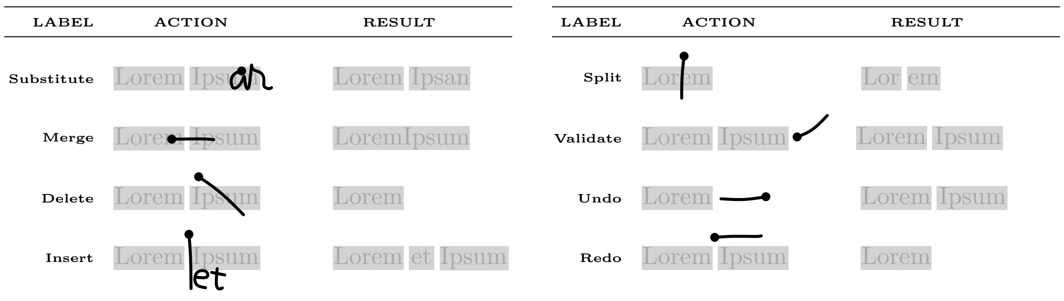


Figure 4: Sample set of interactive text-editing operations that can be developed with MinGestures.

Figure 5: Words' bounding boxes are normalized in height. This allows the user to easily select (or draw over) them, but also to find the gesture context.

Preliminaries: Contextualizing Gestures
For each text segment being edited, word bounding boxes are normalized in height (Figure 5). These "virtual" bounding boxes will be used to accurately detect the word(s) the user is interacting with.

Let $P = \{\boldsymbol{s}_1 \dots \boldsymbol{s}_p \dots \boldsymbol{s}_{|P|}\}$ be a sequence of $|P|$ strokes, where $\boldsymbol{s}_p = \{(x_1, y_1) \dots (x_n, y_n) \dots (x_{|\boldsymbol{s}_p|}, y_{|\boldsymbol{s}_p|})\}$ are sequences of $|\boldsymbol{s}_p|$ 2D points.

On the one hand, the centroid $\boldsymbol{c}_p = \frac{1}{|\boldsymbol{s}_p|} \sum_{n=1}^{|\boldsymbol{s}_p|} \boldsymbol{s}_p$ informs about the word being edited (Figure 5), by searching

$$j^* = \arg\min_j d(\boldsymbol{c}_p, \boldsymbol{c}_j) \tag{1}$$

where $d(\boldsymbol{c}_p, \boldsymbol{c}_j)$ is the distance between the stroke centroid and the j-th bounding box centroid. On the other hand, the angle $\theta_p = tan^{-1} \frac{y_{|\boldsymbol{s}_p|} - y_1}{x_{|\boldsymbol{s}_p|} - x_1}$ measures the slope of the fitted line (if any). As shown in Figure 6, MinGestures uses a tolerance of $\epsilon_1 = 35^\circ$ for diagonal lines and $\epsilon_2 = 10^\circ$ for horizontal/vertical lines, although both parameters are customizable.

Disambiguating Gestures & Handwritten Text
Using our parametric approach, we tried Pearson's ρ as a discriminative feature, as suggested by Li and Hammond [4]. This feature, albeit being intuitive for this task, did not work for us, as indicated in the next section. In contrast, we can use a couple of stroke features that fit better this task. These features rely on the gestures lying on the x-axis. Thus, each feature must be rotated by its indicative angle, as in [5, 9, 10]. First, the aspect ratio

$$\varphi = \frac{w}{h} \tag{2}$$

where w and h are, respectively, the width and height of the stroke bounding-box, informs about the shape of a stroke, therefore "thin" strokes are likely to be near-straight lines. Second, the cummulative horizontal negative derivative

$$\Delta_x^- = \sum_{n=2}^{|\boldsymbol{s}_p|} \max(x_{n-1} - x_n, 0) \tag{3}$$

informs about points being drawn backwards; therefore if a submitted stroke gives $\Delta_x^- \approx 0$, then it is monotonous in the (rotated) x-axis and therefore is likely to be a line.

Using these features, we found out that gestures can be easily disambiguated from handwritten text. Then, together with the taxonomy shown in Table 1, gestures are properly contextualized and potential collisions are solved. For instance, the character "1", a comma, or a dash,

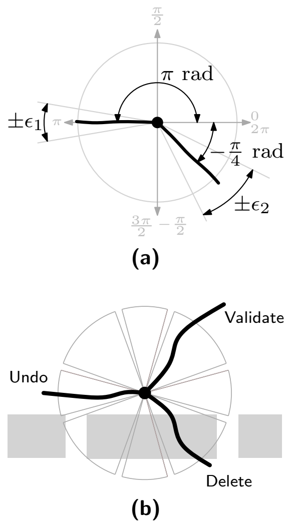


Figure 6: Accommodating gesture variability. [6a] The angular tolerances ϵ_1, ϵ_2 are user-customizable. [6b] Gestures are drawn on non-overlapping areas, so they can be robustly distinguished.

could be misrecognized as lines, for which the context of bounding boxes adequately solves these ambiguities.

Recapitulation: Recognizer Workflow
Whenever the user stops writing on the UI (e.g., after some milliseconds of inactivity), the submitted pen strokes are inspected according to Equations (2) and (3), together with the following taxonomy.

First	Last	Gesture labels
in	in	Substitute, Merge
in	out	Substitute
out	in	[unassigned]
out	out	Delete, Insert, Split, Validate, Undo, Redo

Table 1: Taxonomy of implemented gestures (Figure 4), based on the position of the first and last stroke points with respect to a word bounding box.

Then, if the first stroke is considered to be a line, the stroke angle is computed to classify the corresponding operation. Otherwise, the user would be substituting (handwriting) a word, in which case the strokes must be decoded by a handwriting recognition engine.

Pilot Study

We recruited 32 e-pen users (6 females) aged 26–36. All subjects were unpaid volunteers. A Wacom Bamboo 'Pen & Touch' digitizer tablet was used as input device in a regular PC (2 GHz CPU, 1 GB RAM) equipped with Ubuntu Linux. Participants were asked to perform each gesture up to 10 times, over the same mock-up sentence. Gestures were presented randomly, in order to avoid possible biases in learnability.

We also collected qualitative data about usage experience: **Q1:** Gestures are easy to perform; **Q2:** Gestures are easy to remember; **Q3:** Gestures do suffice for text-editing purposes; **Q4:** I am satisfied with this gesture recognizer. Questions were punctuated in a 1–5 Likert scale (1: completely disagree, 5: completely agree). Users were also encouraged to give free comments and ideas via an online survey at the end of the test.

Recognition Errors
We conveyed a series of experiments to assess how the recognizers performed in terms of accuracy and efficiency. The goal was to classify a given pen stroke into one of nine classes: each of the eight directions and handwriting text. As previously pointed out, first we should tell gestures and non-gestures apart.

Previously we have discussed some stroke features that can be used to detect lines. Nevertheless, we need to establish a threshold after which gestures and non-gestures can be discriminated. In order to identify the threshold without a bias, we decided to split the original corpus into two datasets. The training set consists of 5 samples per gesture plus text (9 classes) per user, with a total of 1440 samples. This set was used to obtain the threshold that minimizes the recognition error rate. The remaining samples (1351 in total) were used as an independent test set. The threshold used for these experiments was the one selected in the training phase. Finally, as the \$1 recognizer needs templates to operate, it was fed with a set of 'perfect' line samples in each of the eight directions.

Our initial approach was to use \$1 out-of-the-box. However, as this recognizer rotates all the gestures to its indicative angle, all lines were rotated to the vertical line. Moreover, \$1 scales gestures to a 1:1 aspect ratio, so lines become almost dots. Thus, results turned out to be random. Therefore, we decided to remove these limitations from \$1. We found that if a gesture was

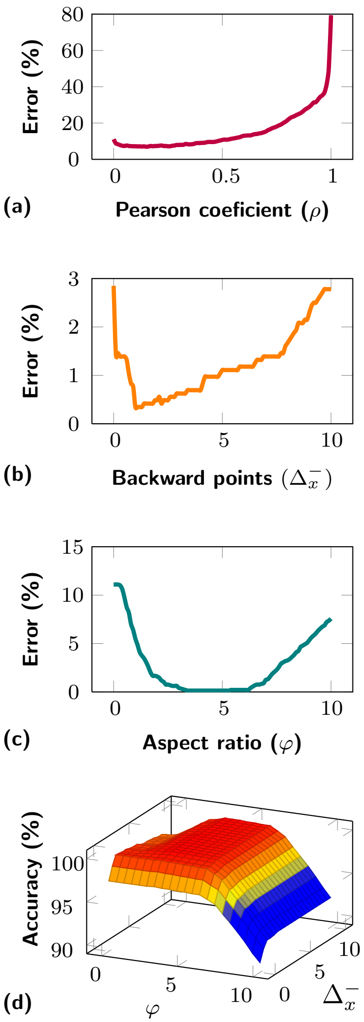


Figure 7: MG's threshold estimations. [7d] Vertical axis shows accuracy instead of error to ease the visualization.

recognized with less than 0.44 posterior probability, then an optimum was obtained with 2.22% error rate.

Next, we experimented with MinGestures (MG for short). Our first approach was to use Pearson's ρ. Unexpectedly, ρ proved to be not very robust for identifying lines, with an error of 6.94% with gestures being recognized as such if $\rho \geq 0.15$, which is very low (Figure 7a). Hence, we decided to rotate the gestures to lie down on the x-axis, and then compute the number of pixels drawn backwards (Equation 3). This resulted in a much better approach with an error of 0.35%, considering as non-gestures more than 1 pixels being drawn backwards (Figure 7b). Also, using the bounding box aspect ratio achieved very good results (0.14% with lines having an aspect ratio ≥ 3.4, Figure 7c). Finally, aiming at an error-free recognizer, we combined the last two techniques. Indeed, we achieved a perfect recognizer (both in training and test) when a gesture is at least 3:1 with no more than 6 pixels drawn backwards. A summary of the results for both training and test is shown in Table 2. As it can be seen, eventually MinGestures behaves as a deterministic error-free interface.

Performance Evaluation
Firstly, we analyzed the time that users invested to draw the gestures. On average, they took 800 ms (SD=610). However, notice that the *Substitute* gesture penalized slightly these results, since users submitted unconstrained handwriting words during the test. Secondly, we computed the average time required to recognize each gesture with MinGestures (Table 3). We ran our recognizer 10 times over all samples in an traditional PC (an i686 with 2 GHz CPU and 2 GB of RAM). The PC needed 0.1 ms on average (SD=0.01) to recognize all gestures. To better understand the impact of the time performance, we repeated the same experiment on an HTC Nexus One running Android 2.2. The mobile device needed on average 0.11 ms (SD=0.43) to recognize all submitted gestures. Regarding the PC performance, this difference of 0.01 ms could be considered statistically significant $[\chi^2_{(1,N=2791)} = 4.66, p = .03]$. Nonetheless, in practice users would not complain between using MinGestures on a mobile device or on a traditional PC in terms of performance, given the narrow margin of difference. These results concluded that the proposed set of gestures are effortless to draw and really fast to recognize.

Qualitative Results
Regarding the four qualitative questions asked at the end of the acquisition tests, we observed that people liked MinGestures overall (see Table 4). We concluded that our recognizer is a convenient approach to deploy text-editing gestures on MIUIs.

Limitations

The simplicity of our approach leads to a few inevitable drawbacks. First, MinGestures is suited to maximize accuracy and runtime efficiency. For that reason, this recognizer is domain-specific and could not fit a researcher's needs in other applications. Thus, text processing applications, such as post-editing interfaces, or transcription and translation systems, are our main and only (although relatively wide) target.

Second, MinGestures provides at most $8 \times 2 \times 4 = 64$ gestures [directions, (un)touching a word, and inside/outside words' bounding boxes], a set of actions that, however, should be enough for text-editing MIUIs. Some guidance to implement more gestures could be differentiating them on the basis of time or speed. If needed, multistrokes gestures could be implemented by

System	training	test
random	88.9	88.9
$1	88.9	88.9
modified $1	2.22	1.78
MG $+\rho$	6.94	7.99
MG $+\Delta_x^-$	0.35	0.52
MG $+\varphi$	0.14	0.22
MG $+\Delta_x^- + \varphi$	**0.0**	**0.0**

Table 2: Summary of recognition error rates (in %). Thresholds were optimized for training and used in test.

# Pts per gesture	24 (18)
Drawing time (s)	0.8 (0.6)

Table 3: Mean (and SD) values of user performance metrics.

Question	Score
Q1	4.69 (0.77)
Q2	4.45 (0.81)
Q3	4.42 (1.04)
Q4	4.66 (0.87)

Table 4: Mean (and SD) scores for the qualitative study, in a 1–5 Likert scale (5 is best).

combining the core set of MinGestures with finite state automata. In any case, there is an inherent limitation of all user-independent systems: creating custom gestures is restricted to the set of primitives used in MinGestures.

All in all, although a concise recognizer like ours may not rival other systems in terms of power, flexibility, or complexity, it is our belief that it may be well suited for a wide range of devices such as tablets, surfaces, or handhelds computers.

Conclusion and Future Work

Stroke-based MIUIs devoted to create or modify text can be easily enhanced with simple gestures, without resorting to complex techniques or using recognizers that are too general. We stressed this fact and developed a deterministic approach to disambiguate among simple gestures and handwritten text, with runtime efficiency as primary focus. This paper may thus serve as a reference guide prior to designing and evaluating text-editing MIUIs.

For future work, we have devised two research avenues. On the one hand, we plan to incorporate more expressivity to MIUIs that are driven by MinGestures. For instance, multiple gesture strokes could be submitted together with handwritten text at a time, speeding thus the cooperative (and interactive and predictive) workflow carried out by the user and the system on a text-editing MIUI.

We already have started working on the integration of MinGestures in a production-ready machine translation system. In the context of the CasMaCat project[1], our gesture set will assist professional translators to post-edit text interactively. The whole machine translation system is expected to be formally evaluated in a few months. It is our belief that MinGestures may enable a natural and accurate (interactive) text edition well beyond e-pen or touch devices.

[1] http://www.casmacat.eu

Acknowledgements

This work is supported by CasMaCat Project 287576 (FP7 ICT-2011.4.2).

References

[1] Hardock, G., Kurtenbach, G., and Buxton, W. A marking based interface for collaborative writing. In *Proc. UIST* (1993), 259–266.
[2] Horvitz, E. Principles of mixed-initiative user interfaces. In *Proc. CHI* (1999), 159–166.
[3] LaLomia, M. User acceptance of handwritten recognition accuracy. In *Proc. CHI EA* (1994), 107–108.
[4] Li, W., and Hammond, T. Using scribble gestures to enhance editing behaviors of sketch recognition systems. In *Proc. CHI EA* (2012), 2213–2218.
[5] Li, Y. Protractor: a fast and accurate gesture recognizer. In *Proc. CHI* (2010), 2169–2172.
[6] Ortiz-Martínez, D., Leiva, L. A., Alabau, V., García-Varea, I., and Casacuberta, F. An interactive machine translation system with online learning. In *Proc. ACL* (2011), 68–73.
[7] Romero, V., Leiva, L. A., Toselli, A. H., and Vidal, E. Interactive multimodal transcription of text images using a web-based demo system. In *Proc. IUI* (2009), 477–478.
[8] Shilman, M., Tan, D. S., and Simard, P. CueTIP: a mixed-initiative interface for correcting handwriting errors. In *Proc. UIST* (2006), 323–332.
[9] Vatavu, R.-D., Anthony, L., and Wobbrock, J. O. Gestures as point clouds: a $P recognizer for user interface prototypes. In *Proc. ICMI* (2012), 273–280.
[10] Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proc. UIST* (2007), 159–168.