doi:10.1093/iwcomp/10.1093/iwc/iwu019

Context-aware Gestures for Mixed-Initiative Text Editing UIs

Luis A. Leiva,* Vicent Alabau, Verónica Romero, Alejandro H. Toselli, Enrique Vidal

PRHLT Research Center, Universitat Politècnica de València
*Corresponding author: Tel.: +34 963878172
Email: `{luileito,valabau,vromero,ahector,evidal}@prhlt.upv.es`

This work is focused on enhancing highly interactive text-editing applications with gestures. Concretely, we study CATTI, a handwriting transcription system that follows a corrective feedback paradigm, where both the user and the system collaborate efficiently to produce a high-quality text transcription. CATTI-like applications demand fast and accurate gesture recognition, for which we observed that current gesture recognizers are not adequate enough. In response to this need we developed MinGestures, a parametric context-aware gesture recognizer. Our contributions include a number of stroke features for disambiguating copy-mark gestures from handwritten text, plus the integration of these gestures in a CATTI application. It becomes finally possible to create highly interactive stroke-based text-editing interfaces, without worrying to verify the user intent onscreen.

We performed a formal evaluation with 22 e-pen users and 32 mouse users using a gesture vocabulary of 10 symbols. MinGestures achieved an outstanding accuracy (less than 1% error rate) with very high performance (less than 1 ms of recognition times). We then integrated MinGestures in a CATTI prototype and tested the performance of the interactive handwriting system when it is driven by gestures. Our results show that using gestures in interactive handwriting applications is both advantageous and convenient when gestures are simple but context-aware. Taken together, this work suggests that text-editing interfaces not only can be easily augmented with simple gestures, but also may substantially improve user productivity.

Categories and subject descriptors: None

Keywords: Handwriting Transcription; Gesture Recognition; Interactive Text Edition

Responsible Editorial Board Member: Name

1. INTRODUCTION

For many of us, editing text on a computer has become a part of our daily lives; e.g., answering e-mails, blogging, chatting, filling web forms, or reviewing and proofreading documents. Text editors are thus central to everyday tasks in large or small measure, but unfortunately the foundations they are based on have barely improved over the last decade. The creation of text has traditionally been the user's exclusive responsibility, but now people often create text in collaboration with machines (MacKenzie and Tanaka-Ishii, 2007).

Interestingly, some text-editing applications have shifted from the above-mentioned traditional approach to a mixed-initiative user interface (MIUI) paradigm, where the user is not the only responsible of all decisions. Indeed, interactive-predictive techniques involved in MIUIs allow both the user and the system to collaborate efficiently and seamlessly. Within this paradigm, text editing goes well beyond basic text typing or correcting. The feedback provided by the user is leveraged by the system to immediately improve its current outcome and to retrain itself to further improve future results (Horvitz, 1999; Culotta et al., 2006; Toselli et al., 2010).

Text-editing applications that are being studied under the MIUI framework include computer-assisted machine translation (Vidal et al., 2008; Ortiz-Martínez et al., 2010, 2011), interactive transcription of handwritten texts (Shilman et al., 2006; Romero et al., 2009a) or spoken documents (Rodríguez et al., 2010b; Revuelta-Martínez et al., 2012), and interactive text generation (Rodríguez et al., 2010a; Revuelta-Martínez et al., 2013). In these applications, a system is trained with examples of the type of text to be produced and the user interactively guides the system to produce the desired output, often with a significant reduction in typing and thinking effort.

Toselli et al. (2011) found that *non-deterministic* feedback interfaces—i.e., those that do not rely on traditional keyboard or point-and-click actions—provide a unique opportunity for improving the user experience in MIUI-driven applications. Clearly, this also raises a number of challenges because non-deterministic feedback requires pattern recognition decoding procedures, which can be as complex as those needed to process other data signals considered in the application; e.g. the input image. Therefore, decoding this feedback is error-prone and typically will require further corrective user interactions.

Of course, advanced text-editing applications can rely on more deterministic feedback, based on keyboard input. However, under the MIUI framework, other modalities may be considered, potentially noisy but definitely more comfortable such as voice, touch, or e-pen (Alabau et al., 2014; Martín-Albo et al., 2012; Alabau et al., 2011a,b; Castro-Bleda et al., 2011). In addition to their potential for increased ergonomics, the interest in these interfaces is nowadays quickly increasing because of the recent growth in the use of mobile and handheld devices for an ever increasing number of both consumer and professional applications.

Of particular relevance here are stroke-based interfaces, where feedback, commands, and text-corrections are all entered by means of a tablet, a touchscreen, or any other input device that can produce a temporal sequence of spatial coordinates representing general *text* and/or specific *commands*. These emerging interfaces provide immediate electronic ink feedback of the digitized writing and mimic the familiar pen-and-paper paradigm to provide a paper-like experience (Tappert and Mosley, 2001). Suitable prototyping tools and techniques are thus essential for such emerging interfaces. Furthermore, with the increasing popularity of stroke-based applications, fast and accurate gesture recognition is becoming a critical part in the lifecycle of such systems. As compared to interacting with traditional devices, stroke-based applications provide the user with an easier and more comfortable interaction style, at the expense of the submitted feedback being less explicit for the system.

1.1. Contributions

In this paper, we contribute with the integration of a set of copy-mark gestures that are specifically adequate for interactive applications involving text editing with an e-pen. In the context of this paper, by e-pen we will refer to a stylus-like device that captures the handwriting or brush strokes of a user; i.e., a physical ink pen with electronic capabilities such as a digitizer tablet or a digitizer screen. However, we must remark that our approach is general enough as to be implemented in other stroke-based devices such as mice, tablets, or touchscreens.

We also contribute with a number of stroke features for disambiguating gestures from handwritten text. It is important to emphasize that disambiguation is tightly coupled to the recognition process itself, since the user can submit either a gesture, a handwritten word, or a combination of both, all at a time. This way, our work makes it finally possible to create highly interactive text-editing stroke-based interfaces, without worrying to verify the user's intent onscreen.

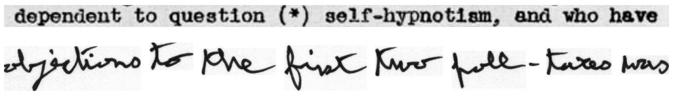
Finally, we should also mention that most of the ideas presented in this paper are a revisitation of previous concepts that have been thriving for decades but have had little success outside academia. In contrast, our results are of immediate application to all of the interactive applications mentioned previously and, to some extent, to non-interactive post-editing applications. Among these applications, for the sake of conciseness, we will focus on text-editing MIUIs as applied to the output of an interactive handwritten text image transcription system, also known as "Computer Assisted Transcription of Text Images" (CATTI) (Toselli et al., 2010; Romero et al., 2012).

1.2. Organization

The rest of the paper is structured as follows. In Section 2 we describe the background of this work and introduce the CATTI framework. In Section 3 we discuss the challenges of designing gestures for interactive text edition. Then, we define in Section 4 a set of marking gestures to enable natural interaction in CATTI applications, and the features that allow a computer to tell gestures and handwritten text apart. A number of user studies are performed in Section 5. Next, we integrate gestures in a CATTI prototype, as described in Section 6. Recapitulations of our general findings and comments on their implications for designing interactive text-editing interfaces are drawn in Section 7, followed by a brief discussion about the limitations of our approach. We then relate to previous work in Section 8. Finally, we end the paper in Section 9 with the main conclusions and provide directions for future work.

2. BACKGROUND

Handwritten text image transcription, mostly known as "off-line" handwriting text recognition (HTR), is the task of converting handwritten text images into an electronic text format. HTR refers to recognizing *cursive* handwriting (Figure 1), and is thus a task quite apart from Optical Character Recognition (OCR). In fact, there is no OCR system that supports cursive handwriting recognition as of today (Alabau and Leiva, 2012).


dependent to question (*) self-hypnotism, and who have

objections to the first two poll-taxes was

Figure 1: OCR deals with documents presenting predictable inter-word and inter-character spaces, consistent typesetting, etc. (top). On the contrary, HTR is suited to recognizing cursive, continuous handwriting, presenting skewed/slanted words, irregular calligraphy, and so on (bottom).

For some time in the past decades, the interest in HTR was diminishing, under the assumption that modern computer technologies would soon make paper-based text documents useless. However, more recently, the task has become an important research topic, especially because of the increasing number of online archives and digital libraries publishing huge quantities of digitized legacy documents. This fact has fostered the creation of digital libraries by public institutions[1,2,3,4] not only to preserve the cultural heritage, but also to make it possible to index, copy, edit, or translate the texts, search for words, etc.

Although many manuscripts have been transcribed as of today, most of the digital libraries host only scanned pages of the original books.[5] The vast majority of these documents, hundreds of terabytes worth of digital image data, remain waiting to be transcribed into an electronic format that would provide publishers, historians, demographers, and other researchers with new ways of typesetting, indexing, querying, and circulating these documents.

While the state of the art in HTR has dramatically advanced in the last few years, current technology is still far from providing sufficiently accurate transcripts for typical applications in a fully automated way. Therefore, rather than relying on inefficient manual procedures, interactive-predictive techniques such as CATTI are ideal candidates to achieve the required quality in a user-friendly way.

[1]http://www.bl.uk
[2]http://www.wdl.org
[3]http://www.europeana.eu
[4]http://www.cervantesvirtual.com
[5]http://www.cic.net/Home/Projects/Library/BookSearch/

2.1. The CATTI Framework

CATTI aims at assisting the user in the image transcription process; that is, the system eases and at the same time speeds up the task of transcribing text. CATTI presents automatic transcriptions of text images to the user, and progressively allows her to refine such transcriptions by performing a series of predefined editing actions; e.g., word substitution, insertion, validation of transcription parts, etc. When the user performs an editing action over a decoded text segment, the system immediately reacts and applies the corresponding editing operation along with other potentially useful changes, anticipating itself to future interactions. These changes are tailored to the text modified or revised so far by the user. Hence, this can be seen as an intelligent multi-word autocompletion procedure.

The basic protocol that rules this process can be formulated in the following steps, which are iterated until a high-quality transcription is obtained; see Figure 2:

(i) The system proposes a full transcription hypothesis $\hat{\mathbf{s}}$ of an input handwritten text line image $\mathbf{l}$, represented as a feature vector sequence.

(ii) The user amends the first error in such hypothesis, implicitly validating an error-free prefix $\mathbf{p}'$.

(iii) A consolidated prefix $\mathbf{p}$ is thus produced, based on the previously validated prefix $\mathbf{p}'$ and the corrections v submitted by the user.

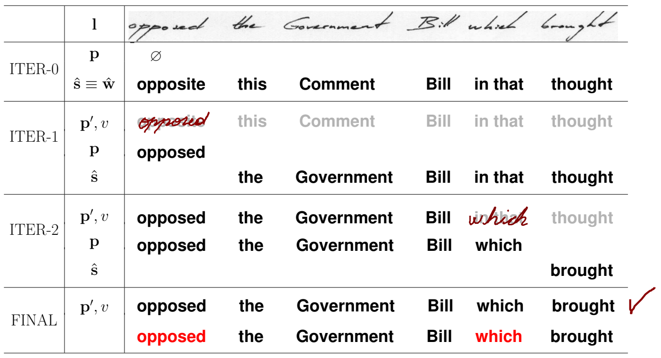
(iv) Using this new prefix $\mathbf{p}$, the system suggests the most suitable continuation $\hat{\mathbf{s}}$ of it, from a probabilistic point of view.

It is important to remark that the handwritten pen-stroke corrections v in Figure 2 are non-deterministic and so they could be misrecognized by an on-line HTR engine. However, for illustration purposes, in the figure we assume that these are correctly decoded.

As observed, a key point of this iterative process is that, within each user interaction, the system can take advantage of the consolidated prefix to produce improved text predictions. Formally, at each step of this process, both the line image representation $\mathbf{l}$, and a corrected transcription prefix $\mathbf{p}$ are available, so the system can autocomplete the full transcription by searching for the most likely suffix $\hat{\mathbf{s}}$, according to:

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}} \Pr(\mathbf{s} \mid \mathbf{l}, \mathbf{p}) = \arg\max_{\mathbf{s}} \Pr(\mathbf{l} \mid \mathbf{p}, \mathbf{s}) \cdot \Pr(\mathbf{s} \mid \mathbf{p}) \quad (1)$$

Since the concatenation of $\mathbf{p}$ and $\mathbf{s}$ constitutes a full transcription hypothesis, the term $\Pr(\mathbf{l} \mid \mathbf{p}, \mathbf{s})$ can be approximated by concatenated character-based Hidden Markov Models (Jelinek, 1998) as in conventional HTR. On the other hand, $\Pr(\mathbf{s} \mid \mathbf{p})$ is approximated by



	$\mathbf{l}$	opposed	the	Government	Bill	which	brought
ITER-0	$\mathbf{p}$	$\varnothing$					
	$\hat{\mathbf{s}} \equiv \hat{\mathbf{w}}$	opposite	this	Comment	Bill	in that	thought
ITER-1	$\mathbf{p}', v$	opposed	this	Comment	Bill	in that	thought
	$\mathbf{p}$	opposed					
	$\hat{\mathbf{s}}$		the	Government	Bill	in that	thought
ITER-2	$\mathbf{p}', v$	opposed	the	Government	Bill	which	thought
	$\mathbf{p}$	opposed	the	Government	Bill	which	
	$\hat{\mathbf{s}}$						brought
FINAL	$\mathbf{p}', v$	opposed	the	Government	Bill	which	brought ✓
		opposed	the	Government	Bill	which	brought

Figure 2: CATTI session example. The system proposes a complete transcription $\hat{\mathbf{s}} \equiv \hat{\mathbf{w}}$ of the input image $\mathbf{l}$, so the user prefix $\mathbf{p}$ is initially empty. In each interaction step the user implicitly validates the longest well-recognized prefix $\mathbf{p}'$ by correcting an erroneous word v, thereby generating a new prefix $\mathbf{p}$ ($\mathbf{p}'$ plus the word v). As a result, the system suggests a suitable continuation $\hat{\mathbf{s}}$ of this prefix $\mathbf{p}$. The process is iterated until a correct transcription $\mathbf{p} \equiv \hat{\mathbf{w}}$ is reached, which can be signaled by a special gesture. In this example, the estimated effort to post-edit the first hypothesis $\hat{\mathbf{w}}$ would be 5/7 words (71%), while the corresponding interactive estimate is $2/7 = 29\%$. This results in an estimated effort reduction of 59%.

modifying an n-gram language model at runtime, in order to cope with the increasingly consolidated prefixes. In a nutshell, by allowing users to dynamically be in command and use the preferred feedback modality, CATTI holds promise to be more acceptable to professional transcribers than manual transcription or plain post-editing, while improving transcriber productivity.

In early CATTI systems, only word substitution was considered as an allowed editing operation (Toselli et al., 2010). Later on, Romero et al. (2009b) incorporated “pointer actions”, allowing the user to indicate that a word was erroneous. This improved the expected user productivity under user-simulated studies. Then, for the sake of completeness, the other two classical editing operations—deletion and insertion—were introduced (Romero et al., 2009a). This way, the user could perform different types of error correction. Recently, with the goal of making the interaction process more productive, Romero et al. (2011) considered a new set of editing operations, such as word concatenation and word splitting, to be incorporated to CATTI systems. This new set was expected to contribute to a reduction of the number of user interactions with respect to using conventional CATTI operations.

Note that each interactive editing operation generates a different prefix for the system. Therefore, the suffix proposed by the system is tightly coupled to the previous editing operation. The following list enumerates the different actions that may be carried out in a CATTI system, together with a brief description of the prefix that is generated. We remark that such a prefix is implicitly validated at each interaction step, until the user decides to explicitly validate the full text transcription.

- *Substitute:* The first erroneous word is replaced with the correct word. The validated prefix consists of all words preceding the wrong word plus the corrected word.
- *Reject:* The user indicates an erroneous word in the proposed transcription and the system automatically proposes a new suffix, in which the first word is different from the erroneous word. All words that precede the incorrect word constitute the validated prefix.
- *Insert:* A new word is inserted between two words that are both assumed to be correct. The validated prefix is composed of all words preceding the inserted word, the inserted word itself, and the word that follows such a newly inserted word.
- *Delete:* An incorrect word between two correct words is removed. The validated prefix consists of all words preceding the deleted word, plus the word that follows such a deleted word.
- *Merge:* Two consecutive words are concatenated to generate a correct word. The validated prefix is composed of all the words that precede the merged words, plus the newly generated word.
- *Split:* A word is divided into two different (correct) words. The validated prefix consists of all words until the splitted word, followed by the new two words.
- *Validate:* The full transcription is accepted. The validated prefix contains all words in the current transcription.

As we have discussed in Section 1, these operations can be unambiguously provided by using a traditional keyboard-based UI by means of predefined shortcuts. Alternatively, all of these operations can be issued by means of stroke-based specialized gestures. A stroke-based UI is often preferred by users because, if adequately designed, gestures can be far more natural and easier to remember than keyboard shortcuts, and they can also be fast enough to produce (Anthony et al., 2013; Cao and Zhai, 2007; Long et al., 2000). Nevertheless, stroke-based UIs must deal with the challenge of recognizing hand-made input. This in turn challenges the design of the stroke-based text-editing UI and of course the underlying gesture recognizer.

3. DESIGNING GESTURES FOR INTERACTIVE TEXT EDITING

A pointer-based device operating a CATTI application should issue text-editing commands by means of gestures. In general, gestures can provide the user with a natural and engaging way of interacting with applications, whenever the underlying system can accurately understand the actions envisioned by the user. This section is devoted to solving the following three open problems that arise from editing text on MIUIs in general, and CATTI applications in particular.

First, gestures and handwritten text must be unambiguously differentiated (Huerst et al., 2010; Leiva et al., 2013). Otherwise, if a gesture is misrecognized as text or vice versa, cascading errors (Karat et al., 1999) 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, possibly issuing an *Undo* operation, and *e)* resubmit the previously intended gesture or text correction again. More specifically, the stream of cursor coordinates that is generated by the input device on the UI must be differentiated into: *1)* gesture strokes aimed at triggering editing actions, and *2)* strokes corresponding to handwritten text introduced by the user either to correct misrecognized words/characters or to enter unconstrained, raw text. In any case, strokes need to be *decoded*, thereby entailing a risk for the system to commit errors.

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" (LaLomia, 1994). Moreover, users tend to trust systems less if they appear to work mysteriously or behave unpredictably (Kristensson and Denby, 2011; Tseng and Fogg, 1999). Add to that the above-mentioned cascading errors, and the user almost surely will reject any interactive text-editing system that does not meet these strict requirements. In addition, gesture decoding must be fast enough to meet the real-time constraints entailed by interactive operations. MIUIs are very often implemented following a client-server architecture, where the user interface itself is generally developed as a light client, typically a mobile or web-based application (Ortiz-Martínez et al., 2010; Romero et al., 2009a; Sánchez-Sáez et al., 2010). In this kind of applications, time-response constraints often dictate that gesture decoding must be carried out on the client side. Correspondingly, gestures must allow for simple decoding with limited amount of computational resources.

The MIUI paradigm offers the possibility to adequately deal with the two above problems by taking advantage of the *context* of each issued gesture. To this end, the system needs 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. This is the third open problem addressed in this paper. As discussed later, on a text-editing MIUI, this can be successfully approached by letting gestures be performed over the text being edited. This way, the context—which mainly consists of the chunks of text interactively processed so far—may help predicting what text or gesture is most probably expected at each interaction step. In addition, the word the user is interacting with allows the system to establish spatial and temporal constraints, which helps disambiguating whether a stroke is aimed at specifying a command or at providing textual data.

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. Further, gestures are limited by user skill, in terms of both memory and performance, and hence they must remain simple. The process of designing gestures is therefore challenging. Ideally, gestures should be as distinct, usable, and quick to make as possible. Because MIUI-driven applications demand frequent interaction, applications should allow the user to do so with minimum effort. Alas, it is difficult to design gestures that are easy for computers to recognize and for humans to learn and remember (Long et al., 1999, 2000).

Inspired in part by the Marking Menus technique (Hardock et al., 1993; Kurtenbach and Buxton, 1994) and similar derivatives (Balakrishnan and Patel, 1998; Bailly et al., 2008; Moyle and Cockburn, 2002), our approach is based on the fact that drawing lines with a pointer device is a very simple task and really easy for users to perform, and it is also very efficient for computers to recognize, since the proposed gestures are linearly separable; see Figure 3. In general, stroke shortcuts can be as efficient as or more advantageous than keyboard shortcuts (Appert and Zhai, 2009) and may have a cognitive advantage because they are easy to memorize.

Marking Menus have been proved to be an extremely efficient menu selection technique (Kurtenbach and Buxton, 1991; Zhao and Balakrishnan, 2004; Delaye et al., 2011). They were designed to allow such selection by either popping-up a radial (or pie) menu, or by making a straight mark in the direction of the desired menu item without having to wait for the menu to pop up. That way, marks can be executed very quickly. In fact, the usage of this "mark mode" is very similar to flick-style finger motions that have become quite natural on touch-based devices for navigation (Raab, 2009). This should allow CATTI users to be more productive, since gestures could be invoked faster and with little effort.

To summarize, the driving idea here is that there is an opportunity to efficiently integrate a series of text editing actions using simple copy-mark gestures and taking advantage of the full application context. It seems intuitive that enhancing gestures with the application context would lead to better recognition outcomes. Hopefully, this would also lead to improvements in terms of system usability, effort reduction, and user acceptability. In the remainder of this paper, we will refer to our approach as MinGestures, a minimal and context-aware recognizer for interactive text editing.

4. IMPLEMENTATION

We tried a baseline set of eight gestures (Figure 3) using simple but reasonably accurate recognizers in the HCI literature, among which we chose the "$ family" for being easily customizable: $1 (Wobbrock et al., 2007), Protractor (Li, 2010), $P (Vatavu et al., 2012), and $N (Anthony and Wobbrock, 2010). Concretely, only $1 would partially fit our needs. On the one hand, Protractor rotates gesture samples to their optimal indicative angle prior and during recognition, therefore it is not appropriate to deal with the set of gestures we were testing, since all lines are misrecognized as horizontal lines. On the other hand, $P completely ignores stroke sequentiality, since gestures are treated as clouds of points, so it cannot differentiate gestures on the basis of direction. Finally, $N is the extension of $1 to multistroke gestures, where all possible stroke orders and directions are considered for each stored template. Therefore, it achieves exactly the same recognition error as $1 with our proposed gesture set.

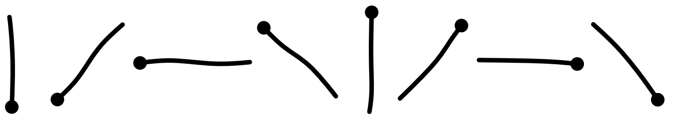


Figure 3: Minimal unistroke gesture vocabulary.

After incorporating some tweaks to $1, overall recognition error was between 16% in training and 7% in test (Section 5), which was definitely insufficient for editing text on MIUIs. We also tested the original Marking Menus algorithm (Kurtenbach, 1991) and the Rubine recognizer (Rubine, 1991b), and found that they both were still insufficient (Section 5.2.2). Therefore, considering the simplicity of the gestures we are targeting, we opted for implementing a customized *parametric* recognizer, since gestures must fit an assumed *model* (straight lines). Otherwise, the submitted strokes should be identified as regular text.

On the other hand, given that we are dealing with text-editing tasks, it would be interesting to incorporate contextual information, namely the words themselves the user is interacting with. This way, we could design a non-overlapped set of gestures with better disambiguation capabilities. In the following we briefly describe our approach, which is specifically tailored to interactive text-editing MIUIs in general and CATTI systems in particular.

4.1. Overview

MinGestures operates on a featurized representation of gestures and assume a parametric model that the target gestures have to fit. One of the appealing factors of this approach is that it is scale and position invariant, but also angular-tolerant. What is more, it does not need resampling stroke points, and does not require gesture templates. Our hypothesis is that this approach should achieve very low error rates, since handwriting can be rarely confused with straight lines, and gestures based on a set of straight lines can be easily differentiated according to their slope.

In this context, MinGestures could be defined as a line-fitting algorithm, but it certainly goes beyond this. First, it is able to disambiguate between gestures and handwritten text, by means of the unsupervised analysis of the user-submitted strokes. Second, MinGestures is context-aware, which implies that it "understands" the text the user is editing. This provides valuable information to build text-editing MIUIs. Finally, given the simplicity of copy-mark gestures, they are extremely fast in terms of both entry speed and recognition. These qualities clearly meet our strict requirements for highly interactive text-editing applications.

4.2. Gesture-based Commands

Figure 4 shows a graphical overview of the gesture set for the editing operations presented in Section 2.1, together with 3 additional operations that are worth including in every interactive application:

- *Undo:* The previous application state is restored; i.e., the previous command is undone.
- *Redo:* The last application state is restored, if available.
- *Help:* A contextual help (or another kind of resource) should be displayed to the user, if available.

Notice that these additional operations do not generate a prefix for a CATTI system, as they are solely

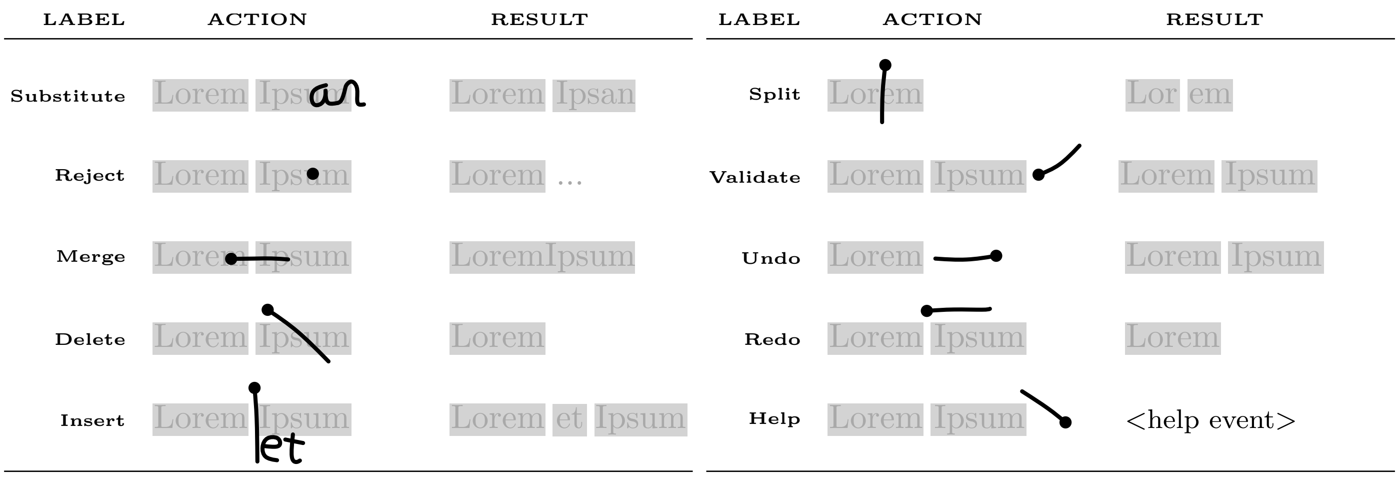


Figure 4: Set of interactive text-editing actions for MIUIs. The recognizer uses information from bounding boxes to construct the corrected prefix and disambiguate gestures; see, e.g., *Insert* and *Split*. The *Reject* gesture is performed as a single click. *Undo*, *Redo* and *Help* do not depend on the position of the bounding boxes, as these operations do not generate a prefix for a CATTI system.

introduced to ease user interaction within the application. Additionally, notice as well that *Insert* and *Substitute* gestures both comprise decoding either a word or a character from the submitted strokes. In these cases, the strokes belonging to handwriting are separated from the gesture stroke, and so they are sent to an underlying handwriting recognition system for decoding, as will be illustrated in Section 6. Finally, an important design decision was to implement the *Reject* operation as a single click, since this is the action that a user would perform before start correcting a word. In general, clicks are well understood by computer users and require little effort to be executed. However, while mouse clicks are really deterministic, composed of exactly one single-point stroke, e-pen clicks might comprise multiple points and thus may lead to an ambiguous recognition—although if points are very close to each other they can be easily told apart from other gestures.

4.3. Disambiguating Gestures and Text

Disambiguation is one of the fundamental capabilities of MinGestures, for which we first considered approaches based on properties of classical linear fitting. To begin, the least square error (LSE) is a measure of the "entropy" of a sequence of stroke points (Li and Hammond, 2012). However, LSE is not normalized by the number of points and it is thus very sensitive to the length of the strokes. To address this shortcoming, we decided to use the root mean square error (RMSE), which is a normalized version of LSE. This feature, albeit performing reasonably well for this task, did not completely meet our expectations. Therefore, we decided to use a couple of additional stroke features to improve the accuracy for this task. These features rely on the analysis of point sequences lying on the x-axis, for which the strokes must be rotated by its indicative angle, as described by Wobbrock et al. (2007); Li (2010); Vatavu et al. (2012). Actually, in our approach the original strokes are left untouched, in order to classify later each gesture on the basis of the 8 octant angles.

For disambiguating gestures and handwritten text we are only concerned about the first stroke submitted by the user; see Figure 4. Thus, let denote it as $\mathbf{t} = \{(x_1, y_1)\cdots(x_j, y_j)\cdots(x_N, y_N)\}$, with $N = |\mathbf{t}|$. First, the cumulative horizontal negative derivative for $\mathbf{t}$

$$\Delta_x^- = \sum_{j=2}^{N} \max(x_{j-1} - x_j, 0) \tag{2}$$

informs about points being drawn backwards; therefore if a stroke yields $\Delta_x^- \approx 0$, then it is monotonous in the (rotated) x-axis and thus is likely to be a line. Second, the aspect ratio of the stroke's bounding box

$$\varphi = \frac{\max(\mathbf{x}) - \min(\mathbf{x})}{\max(\mathbf{y}) - \min(\mathbf{y})} \tag{3}$$

where the point sequences $\mathbf{x} = \{x_1 \cdots x_j \cdots x_N\}$ and $\mathbf{y} = \{y_1 \cdots y_j \cdots y_N\}$ inform about the shape of a stroke. Hence, "thin" strokes are likely to be lines.

Using RMSE along with the additional two stroke features, we found out that gestures can be successfully disambiguated from handwritten text. Concretely, we assume that a stroke is handwritten text when at least one of these 3 features is above its respective threshold. Because only the first stroke must be featurized, this disambiguating operation requires minimal computation time. Then, together with the taxonomy shown in Table 1, gestures can be properly contextualized so that potential collisions can be successfully solved. For instance, the character "|", a comma, or a dash, could all be misrecognized as lines, however the context provided by the bounding boxes adequately solves these ambiguities; since, in our implementation, for character-level interactions the user must perform an insertion mark prior to entering the new character. Below we describe how context is used by MinGestures.

4.4. Contextualizing Gestures

For each text segment being edited, words' 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.

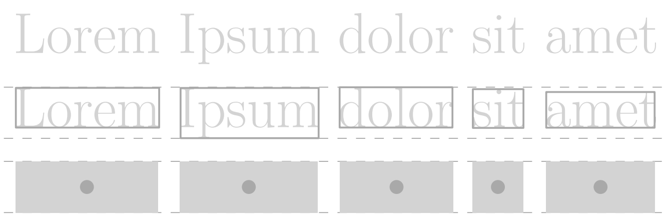


Figure 5: Words' bounding boxes are normalized in height. This allows the user to easily select (or draw over) them, but also to find the interaction context. In the bottom row, central dots represent word centroids.

On the one hand, the centroid $\mathbf{c}_1$ of the first stroke informs about the word being edited (Figure 5), according to the distance to the word centroid, i.e., the closest k-th bounding box: $k^* = \arg\min_k d(\mathbf{c}_1, \mathbf{c}_k)$.

On the other hand, if the first of the submitted strokes turns out to be a line, the angle of the *fitted* line $a = \frac{\hat{\mathbf{y}} - b}{\mathbf{x}}$ measures the slope of such stroke, where b is the intercept of the fit. In the past we tried the naïve approach to compute the angle using the starting and ending points of the stroke, i.e., $\theta = tan^{-1} \frac{y_N - y_1}{x_N - x_1}$ as in the original Marking Menus algorithm (Kurtenbach, 1991), but it did not yield robust results, specially when using an e-pen, as this device usually leads to substantial jittering effects; see Figure 6. Therefore, we found that computing the angle from the fit results in a stronger approach.

As shown in Figure 7, MinGestures uses an angular tolerance of $a \pm \epsilon_1$ for diagonal strokes and a slightly smaller tolerance $a \pm \epsilon_2$ for horizontal/vertical strokes, since diagonal lines are relatively harder to be accurately drawn. By default, $\epsilon_1 = 35°$ and $\epsilon_2 = 10°$. Nevertheless, both tolerances are user-customizable.

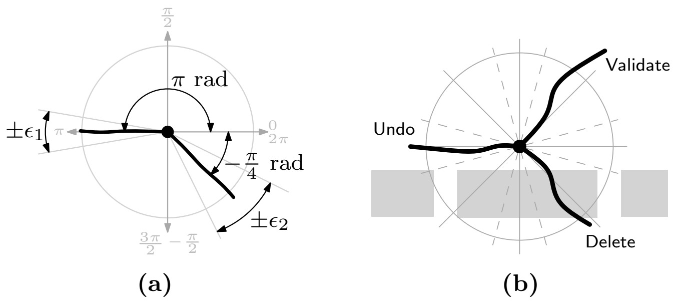


Figure 7: Accommodating gesture variability. The angular tolerances ϵ_1, ϵ_2 are user-customizable (7a). Gestures are drawn on non-overlapping areas, so they can be robustly distinguished (7b). See also Figure 10.

4.5. Recognizer Workflow

Whenever the user stops writing on the UI (e.g., when lifting the e-pen or after some milliseconds of inactivity), the submitted strokes are inspected according to RMSE and Equations (2) and (3). In case of gesture ambiguities, the following taxonomy is queried to solve them.

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.

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

Then, if the first stroke is considered to be a line, the stroke angle of the fitted line 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; see Section 6.1.

In CATTI, the user may submit an arbitrary number of strokes. Typically, this happens when handwriting words; i.e., when performing *Substitute* or *Insert* operations. On the contrary, when a single stroke is submitted, the system must decode the intention of such stroke, which can be either a gesture for a given editing operation or a (partially) handwritten word. In any case, being context-aware, each operation is assigned an operation code. Therefore, only when no operation code is detected, either because no label is assigned to the submitted gesture or because such gesture was not performed as it was designed, no CATTI prefix is generated.

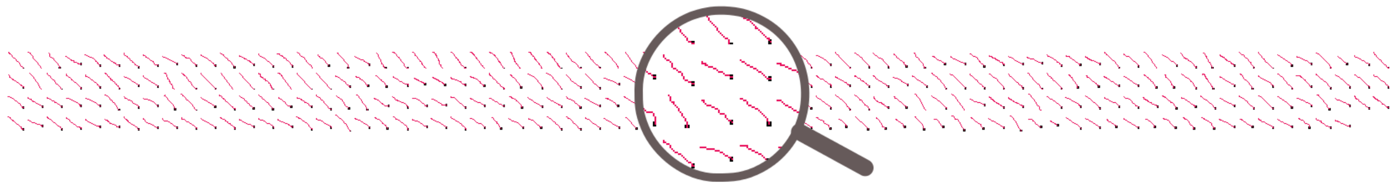


Figure 6: E-pen samples of the *Help* gesture. It can be observed that the pressure sensitivity of the e-pen may lead to substantial jittering effects, which might cause misrecognition results if stroke features are not computed robustly enough.

5. USER EVALUATION

We conducted a formal field study that was two-fold: to test MinGestures in a live setting and evaluate its performance as well as its acceptability by end users. Some of the questions we address in the following section are: Does MinGestures achieve its intended goals? Do users feel comfortable using this set of gestures? Is MinGestures easy to learn and remember? Is this set of gestures suitable enough for text-editing UIs? How well does it perform? Does the recognizer have strong limitations? If so, how can we deal with them? To our knowledge, this is the most comprehensive user study of a gesture-driven transcription application to date.

5.1. Methodology

We experimented with two input devices: a regular computer mouse and an e-pen on a digitizing tablet, both being used as blind-typing devices. Two groups of users were involved in the study, and each group tested only one device (between-subjects design). We gathered qualitative data in terms of subjective user ratings and think-aloud comments. Finally, performance was measured in terms of number of points per gesture, the time users needed to draw each gesture, and the time needed by the system to recognize them.

5.1.1. Subjects

On the one hand, 32 right-handed participants (8 females) aged 27–38 that could use the computer mouse were recruited via email, as they would participate *remotely*. On the other hand, 22 participants (5 females) aged 29–34 were personally recruited from our University's mailing lists. These participants would test the e-pen in the lab, and most of them were not accustomed to using this device. (This would allow us to compare best and worst case scenarios for recognizing gestures.) Only one user in the in-lab group was left-handed. In both groups, most of the users had technical degrees in Physics, Computer Science, or Engineering. All participants were given a gift voucher at the end of the study.

5.1.2. Apparatus

A web-based application was developed to capture gesture samples (Figure 8). This allowed us the flexibility to decentralize the remote (mouse) acquisitions and, at the same time, being able to run the application in a dedicated machine for acquiring the e-pen samples. The application was developed with the sole purpose of verifying if MinGestures could accomplish our research goals. Therefore, no handwriting text recognizer was behind the web UI. Similar to Balakrishnan and Patel (1998), in order to elicit expert behavior, we simply displayed the required gesture marks for the user to emulate, thus completely eliminating any need for familiarity with the gesture set or the web interface to begin with.

A series of traditional computers were used for mouse acquisitions, since users collaborated at their own working environments. For e-pen acquisitions, a Wacom Bamboo 'Pen & Touch' digitized tablet was used as input device in a regular PC (2 GHz CPU, 1 GB RAM) equipped with Linux Ubuntu 10.04. In all cases, gesture samples were registered at 30 fps and logged in XML format.

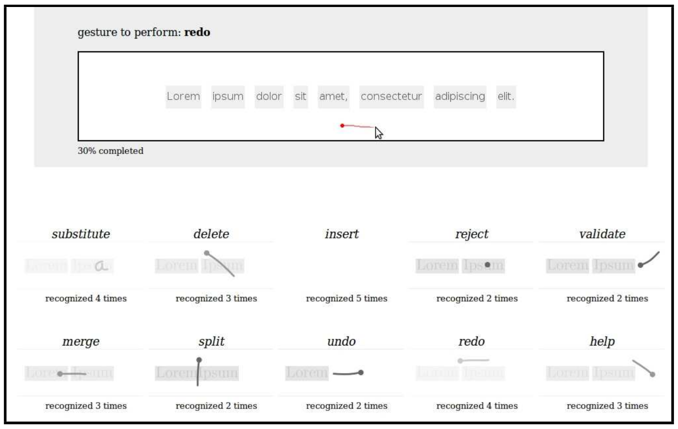


Figure 8: Screenshot of the acquisition application. The drawing area (top) was designed to resemble a real CATTI application; see Figure 12. During the evaluation, the opacity of each gesture image decreased by 20% every time it was performed, so that users would need to recall how gestures should be performed toward the end of the experiment.

5.1.3. Procedure

Before logging gesture samples, participants entered the application in 'test mode' and were told to get familiar with the set of gestures for about 5 minutes. Next, all subjects accessed the application in 'acquisition mode' and were asked to perform each operation up to 10 times—each user submitted 100 gesture samples. In each trial, a mock-up sentence was always presented to the user. The purpose of this choice was to keep participants focused on the gestures and not in the text. Gestures were presented randomly, in order to avoid possible biases in learnability. At the end of the evaluation 3,200 samples were collected from mouse users (32 users × 10 gestures × 10 attempts per user) and 2,200 samples from e-pen users.

We provided a feedforward mechanism in the form of cheat sheet, including a sample image of each gesture below the acquisition UI. We also were interested in testing whether users would be able to remember the proposed gesture set easily in the short term. To achieve this goal, every time an action was issued, the opacity of the corresponding gesture image decreased by 20%, so that, for each user, the second half of the trials (50 in total) were performed without visual help.

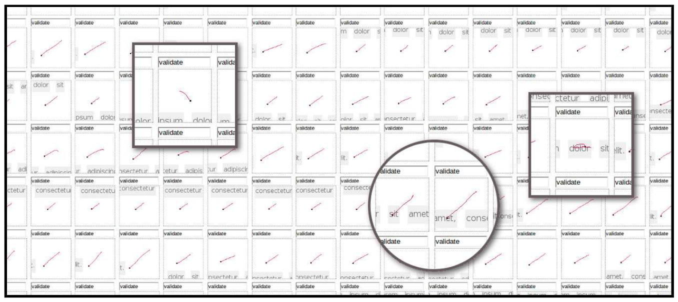


Figure 9: Screenshot of the application to visualize gesture samples, in this case for the *Validate* gesture. When possible, we corrected the labels of those gestures that were patently wrong (in squares) and kept all the rest, even if they were wrongly articulated (in circle).

Finally, we collected qualitative data about usage experience by means of a small survey. Users were asked to state how much they agreed with the following statements:

(i) Gestures are easy to perform.
(ii) Gestures are easy to remember.
(iii) These gestures are enough for text-editing purposes.
(iv) I am satisfied with this gesture recognizer.

Questions were scored in a 1-5 Likert scale (1: strongly disagree, 5: strongly agree). Users were also encouraged to provide free comments and suggestions via an online survey at the end of the test.

5.1.4. Design

Experiments were performed by using a between-subjects design. Among the relevant outcomes, the main one was mean error rate. As with other accurate gesture recognizers, errors were rare and hence data were skewed toward zero. This fact violated the normality assumption, even under usual transformations. Therefore, we used the non-parametric Kolmogorov-Smirnov (K-S) test to assess statistical significance of error rates. Unlike other non-parametric two-sample tests like Mann-Whitney's U or Wilcoxon's rank sum, Kolmogorov-Smirnov's D statistic is sensitive to differences in the general shapes of the distributions in the two samples; i.e., differences in dispersion, skewness, etc. For the rest of outcomes (e.g. gesture articulation time or recognition speed) we did not observe significant departures from normality (verified by Shapiro-Wilk tests, n.s.), so in those cases we used the Welch's Two Sample t-test to assess statistical significance between groups. Welch's t-test is similar to Student's t-test, but more apt for unpaired data groups, having possibly unequal variances. An alpha level of .05 was used for all statistical tests.

5.2. Results and Discussion

Since some samples would be submitted without visual help, we suspected that not all gesture acquisitions would be error-free. Therefore, we developed an application to visualize all samples for a given gesture that would allow us to adequately assign the right gesture label (Figure 9).

5.2.1. Gesture Recall

We used the aforementioned visualization application to analyze the mnemonic effect of gestures in the short term. We looked at 4 subsets: {mouse, e-pen} × {first half, second half} of the trials. Table 2 reports the results of this experiment. Recall errors were computed according to the output of MinGestures, so they include both user and system errors, i.e., in some cases users performed either a different gesture than the one the application was requesting, or the action was not performed as it was actually designed; e.g., stroke exceed angular restriction.

Table 2. Gesture recall results.

Device	Trial	Recall error (%)	95% CI[a]
mouse	1st half	1.65	[1.10 – 2.43]
e-pen	1st half	1.25	[0.71 – 2.13]
mouse	2nd half	1.20	[0.74 – 1.89]
e-pen	2nd half	0.53	[0.21 – 1.19]

[a]Wilson interval estimation for binomial distributions.

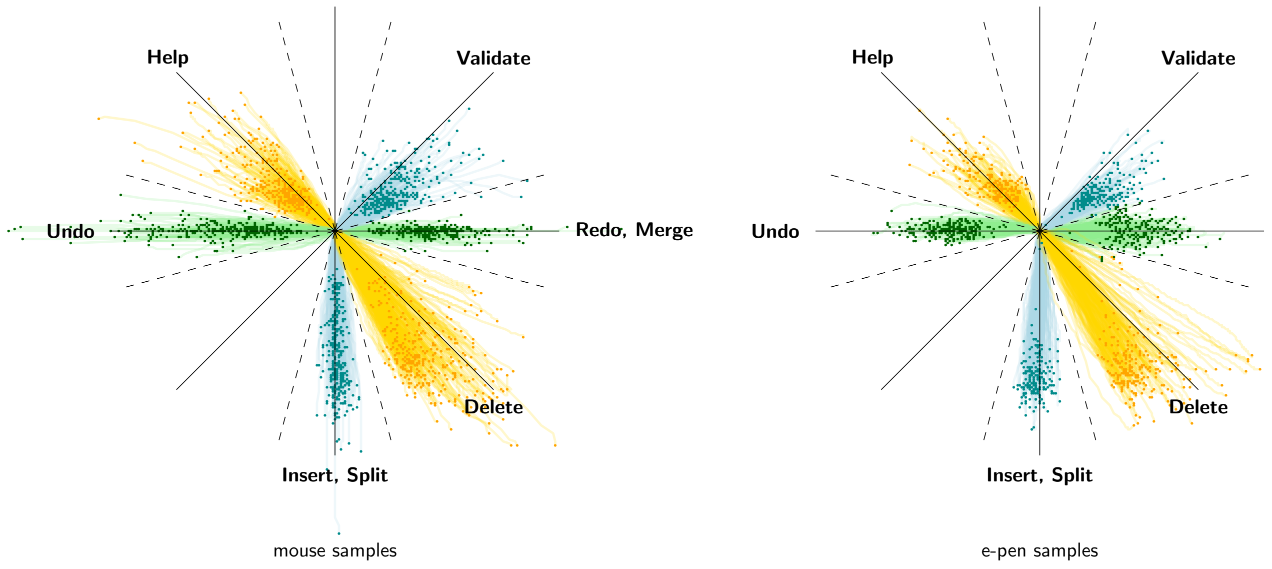


Figure 10: All samples of 6 octant-based gestures; see Figure 4. It can be observed that some gestures were executed very differently, mostly due to the application context; e.g. the *Delete* gesture required to cross over a word's bounding box, therefore these strokes are typically longer than, say, the *Insert* gesture.

Surprisingly, we observed that when gesture images were available as visual help (the first 50 trials out of 100 for each participant), mouse users executed the wrong gesture 1.6% of the time, while e-pen users did it 1.2% of the time. Then, for the last 50 trials, which were performed without visual help, recall error rates improved up to 1.2% for mouse users and 0.5% for e-pen users. These results reveal that people were able to learn the proposed gesture set successfully during the evaluation. In both cases, the K-S test revealed that differences between groups were not statistically significant [$D = 0.123$ for the first half and $D = 0.890$ for the second half, two-tailed hypotheses, $p > .05$ in both cases]. Overall, it can be said that MinGestures behaves almost as an error-free interface to issuing gestures. It remains unexplored, however, if gestures are equally memorable after days or weeks, though we suspect it may be the case.

5.2.2. *Recognition Errors*

We conducted a series of experiments to assess how MinGestures as well as the other recognizers performed in terms of accuracy and efficiency. The goal was to classify a given stroke into one of the gestures depicted in Figure 4, including handwritten text. A fundamental problem is thus how to tell gestures and non-gestures apart. Then, in case of a stroke being a gesture, we should exactly identify what is the gesture given. Ultimately, these steps must be performed into a single procedure, since it is the most intuitive way of interaction, i.e., the user does not need to explicitly state whether a gesture or a word correction is being submitted.

In Section 4.3 we have discussed some stroke features that can be used to detect pseudo-straight lines. Nevertheless, we need to establish the thresholds after which gestures and non-gestures can be discriminated. In order to identify such thresholds without a bias, we decided to split the original corpus into two datasets of two subsets each. The training subsets consisted of 1,032 e-pen samples plus 1,497 mouse samples. These samples were used to obtain the thresholds that minimized the recognition error rate. Conversely, the testing subsets were composed of the remaining samples (1,136 e-pen samples and 1,613 mouse samples). The proportion of gesture samples was balanced with the exception of some samples that were removed as a result of a bad acquisition (32 e-pen samples and 90 mouse samples). The thresholds used for the testing subsets were the ones selected in the training phase.

These experiments were user-independent, meaning that users in the testing subsets were different from those in the training subsets, for the thresholds estimation to be more generalizable. In addition, it is important to note that some of the gesture samples can only be differentiated when put into context. In particular, *Merge* and *Redo* are both straight lines that go in east direction, whereas *Insert* and *Split* both go south. Hence, these gestures were considered equivalent in the experiments where gestures were not contextualized. As a result, apparently counter-intuitive, it is possible for contextualized experiments to achieve less accuracy than the non-contextualized experiments, because the gesture vocabulary is bigger.

Our initial approach was to use \$1 out-of-the-box. Since this recognizer needs templates to operate, it was fed with a set of "perfect" line samples in each of the eight directions—each template had exactly 2 points, so that \$1 could perfectly interpolate intermediate points. 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 became almost dots. Thus, results turned out to be quite random, above 40%. Therefore, we decided to remove these limitations from \$1. The recognizer performance improved substantially, but still the error rates were not satisfactory, with values around 7–15%.

Then, we experimented with Marking Menus (Kurtenbach, 1991), which obtained an error rate above 10% for mouse and near 20% for e-pen. There are two main reasons for such high error rates. First, Marking Menus cannot distinguish straight lines from regular handwriting. Second, when the user operates over words' bounding boxes, diagonal gestures such as *Delete* or *Validate* tend to be less slanted, since bounding boxes are usually bigger in width than in height; see e.g. Figure 10. This leads to a number of misclassification errors due to angular restrictions.

Next, we used the Rubine recognizer (Rubine, 1991a), which is a parametric multi-class linear classifier. However, the computation of the classifier weights relies on stroke features for training that made it impossible to use the perfect templates we used in the previously tested \$1 versions. Thus, in this case we used the gesture samples from the training set. The initial results showed error rates around 15%. As these results seemed still high, we manually revised the results. We observed that some training samples were not good templates for this recognizer. By removing the training samples that caused low accuracy, the results could be improved to 7% for e-pen and 3% for mouse. However, this way of selecting templates is biased and, still, the accuracy was not sufficient enough according to our needs.

In consequence, we decided to experiment with MinGestures. Our first approach was to use the RMSE feature alone. Unexpectedly, RMSE proved to be not very robust for identifying lines, with error rates around 2% with gestures being recognized as such if RMSE < 1.5 (Figure 11a).

Using Equation 2 resulted in a much better approach, with an error rate between 0.68% and 1.76%, considering as non-gestures more than 2 pixels being drawn backwards (Figure 11b). Also, using the bounding box aspect ratio (Equation 3) achieved very good results, with error rates between 0.62% and 1.58% with lines having an aspect ratio ≥ 6; see Figure 11c. Finally, aiming at an error-free recognizer, we combined the three features. Indeed, we

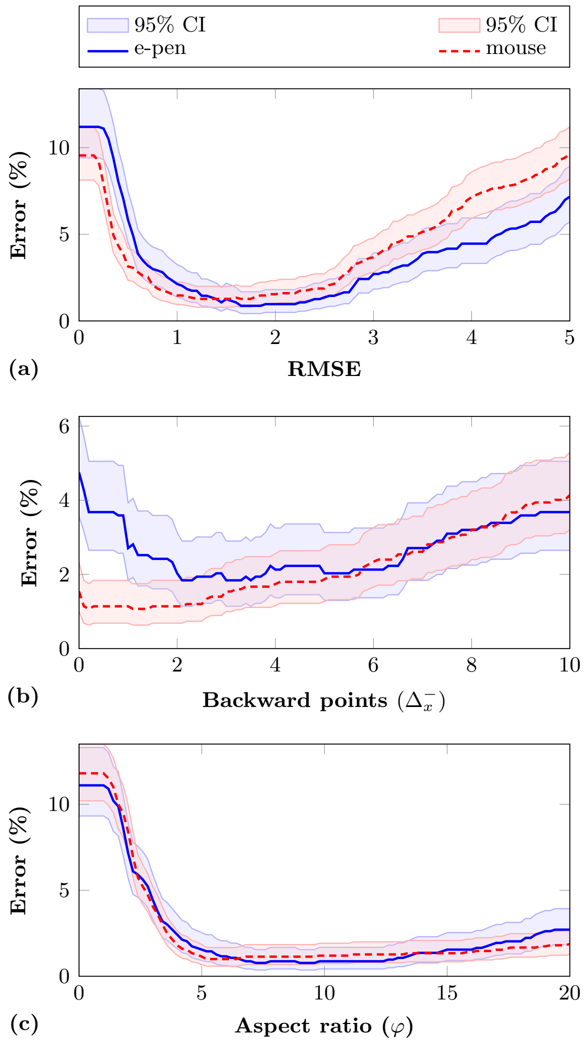


Figure 11: Threshold estimations in the training set for different features.

were able to further reduce the recognition error to 0.43% and 1.32% when a gesture is at least 5:1 with no more than 6 pixels drawn backwards and RMSE lesser than 3.

A summary of the results for both training and test is shown in Table 3 and Table 4. Differences between mouse and e-pen groups were not statistically significant [$D = 0.2834, p = .656$, n.s., two-tailed hypothesis]. For most of the participants (44 out of 52) MinGestures achieved error rates $\leq 1\%$ or, equivalently, accuracy rates ranged from 99% to 100%. The worst case was observed for one mouse user that failed to perform the *Insert* gesture 50% of the time, which could be considered an outlier given how well the other users did perform.

In light of these results, we conclude that our recognizer is suitable for text-editing applications that incorporate

Table 3. Summary of recognition error rates (in %) without context. Recognizer thresholds were optimized for training and used in test. Confidence Intervals are calculated according to the Wilson Method for binomial distributions.

	E-pen				Mouse			
System[a]	**training**	**95% CI**	**test**	**95% CI**	**training**	**95% CI**	**test**	**95% CI**
$1 recognizer	41.2	[40.3 – 42.2]	40.5	[39.6 – 41.4]	45.0	[44.3 – 45.6]	46.6	[46.0 – 47.2]
Marking Menus	18.5	[16.2 – 21.0]	19.5	[17.2 – 21.9]	12.8	[11.2 – 14.6]	13.1	[11.5 – 14.8]
Modified $1	16.0	[15.3 – 16.7]	15.6	[15.0 – 16.3]	7.48	[7.14 – 7.84]	7.56	[7.25 – 7.89]
Rubine	14.6	[14.0 – 15.3]	14.1	[13.5 – 14.7]	15.4	[14.9 – 15.9]	15.7	[15.2 – 16.1]
MG w/ RMSE	1.94	[1.22 – 3.01]	2.02	[1.79 – 2.29]	1.67	[1.11 – 2.48]	2.48	[2.30 – 2.68]
MG w/ Δ_x^-	2.13	[1.37 – 3.25]	1.76	[1.54 – 2.01]	0.53	[0.24 – 1.07]	0.68	[0.58 – 0.79]
MG w/ φ	1.26	[0.70 – 2.18]	1.58	[1.38 – 1.82]	0.40	[0.16 – 0.89]	0.62	[0.52 – 0.72]
MG w/ Δ_x^- $+\varphi$ +RMSE	**0.77**	[0.36 – 1.55]	**1.32**	[0.76 – 2.20]	**0.26**	[0.08 – 0.70]	**0.43**	[0.19 – 0.91]

[a]Since no context is considered in this experiment, the gesture vocabulary is composed of 8 symbols.

Table 4. Summary of recognition error rates for contextualized MinGestures (in %). Recognizer thresholds were optimized for training and used in test. Confidence Intervals are calculated according to the Wilson method for binomial distributions.

	E-pen				Mouse			
Contextualized MG[a]	**training**	**95% CI**	**test**	**95% CI**	**training**	**95% CI**	**test**	**95% CI**
w/ RMSE	0.87	[0.42 – 1.68]	1.67	[1.04 – 2.63]	1.27	[0.78 – 2.00]	1.67	[1.13 – 2.45]
w/ Δ_x^-	1.84	[1.14 – 2.90]	1.67	[1.04 – 2.63]	1.07	[0.63 – 1.75]	1.24	[0.78 – 1.93]
w/ φ	0.775	[0.36 – 1.55]	1.41	[0.83 – 2.31]	1.00	[0.58 – 1.67]	1.12	[0.68 – 1.78]
w/ RMSE $+\Delta_x^-$ $+\varphi$	**0.67**	[0.29 – 1.42]	**1.23**	[0.70 – 2.09]	**1.00**	[0.58 – 1.67]	**0.93**	[0.54 – 1.55]

[a]Using the full gesture vocabulary (10 symbols, see Figure 4).

gestures performed either by an e-pen or a computer mouse. What is more, regarding the type of input device, this study corroborates the findings drawn in earlier works (Kurtenbach et al., 1993; Zhao and Balakrishnan, 2004), showing that *1)* simple copy-mark gestures perform exceptionally well on different devices; and *2)* users quickly learn the associations between the gesture and its associated command. Therefore, we hypothesize that MinGestures should also be suitable for use in other kind of stroke-based devices, such as tabletops or touchscreens.

5.2.3. Performance Evaluation

We computed the average time required to recognize each gesture. Since mouse users performed the evaluation remotely, we reproduced the working settings in a traditional PC (2.8 GHz CPU, 2 GB RAM) running Linux Ubuntu 11.10. We ran our recognizer 10 times over all gesture samples, and the PC needed less than 0.5 ms on average to recognize each gesture. Table 6 summarizes these results, as well as the rest of evaluation metrics.

To better understand the impact of time performance, we repeated the experiment on an HTC Nexus One (1 GHz CPU and 512 MB RAM) running Android 2.2. The mobile device needed on average 1.14 ms ($SD = 0.71$) to recognize each submitted gesture. If compared to the performance of the PC, this difference of 0.5 ms is statistically significant [$t(62.97) = -5.64, p < .001$, two-tailed hypothesis]. Nonetheless, in practice, users would not complain regarding the use of MinGestures on a mobile device or on a traditional PC, given such an extremely narrow actual difference. These results concluded that MinGestures is a really fast recognizer, specifically one order of magnitude faster than other comparable recognizers.

In addition, we analyzed the time users invested to draw gestures, including also time spent in handwriting words. Mouse users needed 800 ms on average ($SD = 610$), while e-pen users took 970 ms ($SD = 740$). Differences were not found to be statistically significant [$t(46.90) = -0.33, p = .740$, n.s., two-tailed hypothesis]. As noticed, a high variability was introduced due to the fact that each participant submitted a very different number of strokes when handwriting words; i.e., the *Substitute* and *Insert* operations. If we consider just the time drawing gestures alone, then it is around 500 ms on average with standard errors below 100 ms in all cases.

5.2.4. Qualitative Study

Regarding the four qualitative statements asked at the end of the acquisition tests, we observed that people liked MinGestures in general terms; see Table 5.

Table 6. Average (and SD) performance metrics per gesture for each user group.

	Mouse users						E-pen users					
Label	**# Points**		**Draw time (s)**		**Recog. time (ms)**		**# Points**		**Draw time (s)**		**Recog. time (ms)**	
Substitute	55	(25)	1.80	(0.84)	0.73	(0.35)	64	(26)	2.10	(0.87)	0.83	(0.19)
Delete	34	(15)	1.80	(0.52)	0.44	(0.21)	37	(13)	1.20	(0.45)	0.62	(0.52)
Insert	31	(12)	1.00	(0.41)	0.41	(0.13)	48	(26)	1.60	(0.85)	0.56	(0.15)
Reject	1.1	(1.4)	0.03	(0.01)	0.00	(0.00)	1.9	(1.3)	0.06	(0.01)	0.19	(0.28)
Validate	18	(9.5)	0.59	(0.32)	0.29	(0.07)	14	(4.7)	0.48	(0.16)	0.26	(0.06)
Merge	25	(9.7)	0.84	(0.32)	0.36	(0.15)	27	(9.2)	0.91	(0.31)	0.47	(0.17)
Split	22	(10)	0.75	(0.34)	0.28	(0.11)	29	(13)	0.97	(0.42)	0.44	(0.12)
Undo	19	(9.8)	0.64	(0.33)	0.27	(0.08)	31	(16)	1.01	(0.52)	0.50	(0.28)
Redo	17	(7.4)	0.57	(0.25)	0.28	(0.08)	23	(11)	0.76	(0.38)	0.45	(0.32)
Help	18	(7.6)	0.59	(0.25)	0.32	(0.15)	20	(7.6)	0.66	(0.25)	0.32	(0.09)
Avg. Total	24	(18)	0.86	(0.61)	0.36	(0.21)	29	(22)	0.97	(0.74)	0.46	(0.30)

Table 5. Mean (and SD) scores for the qualitative study, in a 1–5 Likert scale (5 is best). No statistically significant differences were found between both groups.

Statement	**Mouse users**	**E-pen users**
1: Easy to perform	4.69 (0.77)	4.81 (0.38)
2: Easy to remember	4.45 (0.81)	4.81 (0.38)
3: Enough gestures	4.42 (1.04)	4.59 (0.83)
4: Satisfaction	4.66 (0.87)	4.90 (0.28)

Regarding the comments submitted by the participants, people were satisfied with MinGestures overall:

- *"I liked the system. I'm amazed how well it works. I think it's excellent for using with a drawing tablet."*
- *"Some actions like Merge, Undo, or Redo were easy to remember because they are quite intuitive."*
- *"I believe this recognizer can solve most of the necessities of today's pen UIs."*

Of course, some users also remarked a few drawbacks:

- *"I needed to consult the gesture images often".*
- *"There are gestures that are very dependent of the word boxes... It may be disturbing for the user having to find white spaces."*
- *"Although it performs quite well, I think there would be an inevitable learning curve before using this recognizer on a daily basis."*

To conclude this study, we are aware that our approach is not exempt of limitations, which are discussed later on in Section 7.2.

6. INTEGRATING GESTURES IN CATTI

In order to assess the utility of MinGestures in an interactive HTR scenario, we implemented the recognizer in an actual CATTI prototype; see Figure 12. This prototype has been used in a number of previous studies, including both researchers and professional paleographers (Leiva et al., 2011; Bosch et al., 2014). However, this time we aimed for a completely automated study; see Section 6.2. In the literature, other authors have conducted a similar setup with real users using the same CATTI prototype and a comparable dataset (Leiva et al., 2011), but no gestures were considered and only two handwritten pages were evaluated. In contrast, through user simulations, we can acquire a substantial number of observations. In sum, this study evaluated six of the operations implemented in MinGestures using a CATTI prototype through user simulations.

6.1. Prototype Overview

As introduced in Section 2.1, a CATTI system uses as input a handwritten text image and leverages a series of partially user-validated transcription (prefixes) to propose suitable continuations of the full sentence (suffixes), until a high-quality error-free output is achieved. Since the user iteratively and repeatedly interacts with the CATTI system, ensuring an accurate, fast, and easy-to-use process is a crucial endeavor for the success of the handwriting transcription tasks.

The aforementioned CATTI prototype follows a client-server communication model. The web interface (the client) is responsible for rendering the application and capturing user actions, which are performed by means of stroke-based gestures. A built-in CATTI engine (the server) loads language models of the text images and builds intelligent autocompletions.

The user transcribes the handwritten text images line by line, making corrections with some pointer-based input device, e.g., a stylus or a touchscreen. When strokes that represent words or characters are submitted for

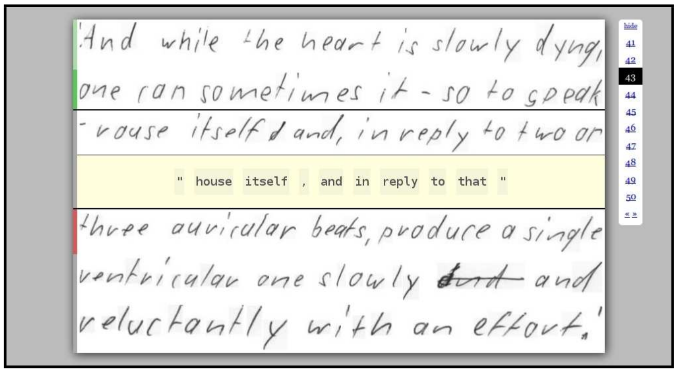

And while the heart is slowly dying,
one can sometimes it - so to speak
- rouse itself and, in reply to two or
" house itself , and in reply to that "
three auricular beats, produce a single
ventricular one slowly ~~and~~ and
reluctantly with an effort.'
hide
41
42
43
44
45
46
47
48
49
50

Figure 12: Screenshot of the CATTI prototype. On each page, users could select one line at a time. Once an image line is clicked, the main application loads the editing area. In the left margin of each page, lines are marked as validated (the full text segment has been reviewed), pending (only a fraction of the current text segment has been transcribed), or locked (someone else is working on the same text segment). More details about the UI or the application workflow can be found in (Romero et al., 2009a; Leiva et al., 2011).

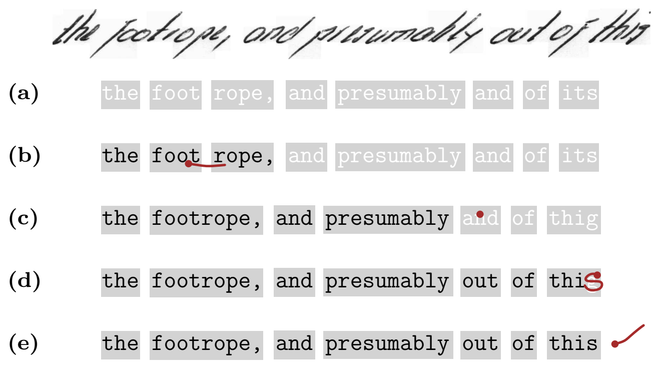

the footrope, and presumably out of this
(a) the foot rope, and presumably and of its
(b) the foot rope, and presumably and of its
(c) the footrope, and presumably and of thig
(d) the footrope, and presumably out of this
(e) the footrope, and presumably out of this

Figure 13: An example of MinGestures integrated in a CATTI system. An initial hypothesis for an input image (13a) is presented to the user. By merging words #2 and #3 (13b), a consolidated prefix (highlighted in black) is sent to the CATTI server with the associated operation code. The system generates a second alternative (13c), which still contains some errors. The user then rejects the first erroneous word in the received suffix (thereby implicitly consolidating a longer prefix), and the system provides a new text completion. The user still finds and corrects an error (13d), and the system finally provides a correct transcription, which is accepted by the user (13e).

decoding (i.e., with the *Insert* and *Substitute* gestures), the server uses an on-line HTR feedback subsystem. This on-line subsystem is based on Hidden Markov Models (HMMs) and n-gram language models, where on-line strokes are represented as *temporal* sequences of 6-dimensional feature vectors. These time-domain features are: stroke point coordinates, first and second derivatives of these coordinates, and stroke curvature.

The on-line subsystem acts in synergy with an off-line HTR subsystem, in order to improve the recognition of the (off-line) handwritten text image. Like the on-line subsystem, the off-line subsystem is based on HMMs and n-grams, although in this case off-line information is represented as *spatial* sequences of 60-dimensional feature vectors. These feature vectors account for sequential information about the amount of gray level and its distribution along the handwritten text in the input image.

For both on-line and off-line HTR subsystems, the decoding of their respective feature vectors is efficiently performed by the Viterbi algorithm (Jelinek, 1998). Further information about this built-in recognizer can be found in (Toselli et al., 2011), which is a very complete HTR guide in this regard.

Figure 13 shows a CATTI session example using MinGestures. For the sake of simplicity, let us assume that the time required to draw a gesture is the same as drawing one character with a pen-like device. Then, in Figure 13 the user would save $1 - \frac{4}{44} = 90\%$ of writing effort compared to manual transcription (44 characters would have been manually entered vs. just 4 gestures in a CATTI system) and $1 - \frac{4}{12} = 66\%$ compared to post-editing (12 characters would have been post-edited vs. just 4 gestures in a CATTI system). As observed, an important amount of effort can be saved when CATTI is driven by gestures.

6.2. Apparatus and Procedure

MinGestures was embedded in the application UI, and each gesture was associated to an operation code which unequivocally identified its corresponding editing operation. The operation code, together with the user-validated prefix—and optionally pen-based strokes to be decoded into words, if available—would be sent to the CATTI engine.

The interactive editing operations that we evaluated in this study were those that generate a different prefix for the system; i.e., *Substitute*, *Reject*, *Merge*, *Delete*, *Insert*, and *Split*. For instance, the *Validate* gesture normally issues an operation code to re-train language models, and therefore it does not take part in the interactive transcription process itself. The remaining gestures (*Undo*, *Redo*, and *Help*), as previously pointed out in Section 4.2, are completely managed by the application UI and hence are never sent to the CATTI engine. This way, we could automatically simulate user interactions under controlled conditions.

On the other hand, given the scope of this paper, we are interested in evaluating the effect of gesture interaction.

Therefore, we assumed that on-line handwriting recognition would work without errors.[6] As a result, for the purpose of this study, substitutions and insertions were both considered as a regular gesture.

6.3. Corpus

We used the IAM handwriting database (Marti and Bunke, 2002) for the experimentation. This corpus features gray-level images of unconstrained handwritten English texts, and is publicly available.[7] Such text images in turn correspond to handwritten sentences from the LOB corpus (Johansson et al., 1996), which encompasses around 500 printed electronic English texts of about 2K words each and about 1M total running words.

The IAMDB dataset is provided at different levels: sentences, lines, and isolated words. Here, in order to obtain comparable results with previous work, we used the sentence-level partition (Marti and Bunke, 2001; Toselli et al., 2010). This partition was split into a training set composed of 2,124 sentences handwritten by 448 different writers, and a user-independent test set composed of 200 sentences handwritten by 100 different writers. Table 7 summarizes this information.

Table 7. Basic statistics of the IAM database.

	Training	Test	Total	Lexicon
Writers	448	100	548	–
Sentences	2,124	200	2,324	–
Words	42,832	3,957	46,789	8,017
Characters	216,774	20,726	237,500	78

As stated in Equation 1, CATTI relies on linguistic and morphological information. We used a 2-gram language model for the former, trained with the whole LOB corpus (excepting the test sentences) and a HMM for the latter, using the handwritten text images in the training set.

6.4. Evaluation Metrics

Three evaluation measures were adopted to assess performance: word error rate (WER), word stroke ratio (WSR), and estimated effort reduction (EFR).

On the one hand, WER is the number of editing operations needed to correct an automatically generated transcription, normalized by the number of words in the reference transcription. As such, WER estimates the post-editing effort that a user would need to amend the errors produced by a (non-interactive) HTR system. On the other hand, WSR is the number of words that must be interactively corrected to achieve a perfect transcription, normalized by the number of words in the reference transcription. Hence, WSR estimates the effort that a user would need to produce completely correct transcriptions using a CATTI system. Finally, EFR is the relative difference between WER and WSR, and provides an estimate of the reduction—in terms of words to be corrected—that can be achieved by using CATTI with respect to using regular HTR post-editing.

Since WER and WSR are relative to the number of words in the reference transcription, the lower the better. On the contrary, the higher the EFR the better. It is worth pointing out that EFR may have negative values, i.e., when the effort required to obtain perfect transcriptions with a CATTI system is higher than the effort required with a regular post-editing HTR system.

6.5. Results and Discussion

State-of-the-art results reported an estimated performance of a post-editing HTR system (assessed by WER) of 25.8% and the performance of a baseline CATTI system[8] (assessed by WSR) of 21.8%. These account for an EFR of 15.5% (Toselli et al., 2010).

Table 8 depicts the results of incorporating the editing operations of MinGestures to CATTI. We considered that a single operation would be performed at a time to amend the system hypotheses. We also considered the simplified but reasonable assumption that the cost of performing each editing operation is the same; with the notable exception of the *Reject* operation, which is considered to have no cost. The reason is that the *Reject* operation is performed implicitly when positioning the cursor to introduce the corrected word in a baseline (keyboard-based) CATTI system. Then, WSR results for stroke-based devices such as mouse or e-pen were computed by counting each gesture error twice: one due to the failed gesture recognition and another due to the required error correction, similar to what the user would perform in a real setting.

According to the results, the number of user interactions needed to achieve perfect transcriptions by using a stroke-based device is comparable to using a deterministic input device (the keyboard), as differences in WSR and EFR are approximately equal; see Table 8. Nevertheless, stroke-based devices enable a more comfortable human-computer interaction, and therefore result in an easier and more effective transcription process. In addition, by using the different editing operations of MinGestures, user effort is substantially reduced with respect to using a baseline CATTI system. For example, using an e-pen the

[6]Decoding accuracy is typically around 97% (Toselli et al., 2010).
[7]Available at `http://iamwww.unibe.ch/~fki/iamDB/`

[8]Considering only the *Substitute* operation at the word level.

human effort needed to produce an error-free transcription is reduced by as much as 27.5%, whereas using a baseline CATTI the observed human effort reduction was 15.5%. This means that, from every 100 words that are misrecognized by a post-editing HTR system, a baseline CATTI user will have to correct 85 word-errors, while a CATTI user using gestures will have to correct only 72 words—the other 28 words will be automatically corrected by the CATTI system.

Table 8. Performance of gesture-driven CATTI, measured by the word stroke ratio (WSR) and the estimated effort reduction (EFR) regarding to using a non-interactive approach.

Device	**WSR** (%)	**EFR** (%)
keyboard	18.5	28.3
mouse	18.6	27.9
e-pen	18.7	27.5

A more detailed analysis of the contribution of each gesture to this experiment revealed that the *Merge* operation only proved to be helpful in one case for the IAM corpus. Further, we observed that the *Split* operation did not help to improve the proposed transcriptions in these experiments. On the other hand, *Insert* and *Delete* operations were the most frequent ones, and consequently they contributed with the most significant improvement with respect to the baseline case.

By way of conclusion, as reported in Section 5.2.1, gesture integration has been shown to be positively received by users. Taken together, our experiments suggest that incorporating gestures to interactive text-editing applications is an advantageous approach worth considering.

7. GENERAL DISCUSSION

The appealing feature of MinGestures as a straightforward and context-aware interface seems clear, although it goes beyond the reported evaluation results. Besides its demonstrated performance and accuracy, there are other interesting advantages in associating short straight marks as gestures. For instance, being so simple to perform, the frequent use of these gestures may reinforce the mapping to their corresponding commands. Also, as a consequence of its simplicity as well, the entry speed of gestures is extremely low. These attributes are expected to increase the productivity of regular CATTI users.

On the other hand, it has been shown that MinGestures can be conveniently integrated to command stroke-based text-editing UIs, since it takes advantage of the application context to disambiguate interaction intent; see Section 4.5. From above, it is clear that, in the design process of text-editing applications intended to be used with this kind of gestures, a series of opportunities are identified to enhance these applications further. We briefly discuss some of these opportunities below.

7.1. Design Implications

First and foremost, we found a number of stroke features that, when combined, enable a clear disambiguation between gestures and handwritten text. This allows MIUI developers and designers to implement a simple approach to deal with the uncertainty of 2D stroke sequences. Then, the proposed gesture set could serve as a design guide to determine an adequate number of parameterizable user commands. This way, applications could take the most advantage of gestures while reducing user's cognitive load. Second, MinGestures could help to develop similar-purpose pen-based UIs, for instance, facilitating the review of automatically generated translations. It also could be used to augment other modalities (e.g., speech-based) in interactive applications, as other authors have suggested in previous work; see Section 8. Finally, in the context of CATTI systems, it would be possible to delimit the set of gesture candidates to be performed according to the context of the words. For instance, the system could suggest to the user a list of suitable operations to be performed when positioning the pointer over a certain word, similar to a Marking Menu for novice users, where non-available options would be disabled.

7.2. Limitations

The simplicity of our approach leads to a few inevitable drawbacks. Firstly, 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.

Secondly, MinGestures provides at most $8 \times 2 \times 4 = 64$ gestures [directions × (un)touching a word × inside or 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 combining the core gesture set 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. This issue can be avoided by using a more complex recognizer, probably at the expense of recognition accuracy or real-time performance, or by resorting to more advanced pattern recognition techniques. However, it is currently outside the scope of this work.

All in all, although a concise recognizer like ours may not rival other systems in terms of 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.

8. RELATED WORK

Of all applications, perhaps the human factors of text editing have been the most studied (Buxton, 1988). In addition to CATTI, there is a substantial body of research in which stroke-based gestures are used for text-editing tasks. In the context of this paper, it is worth mentioning off-line techniques for editing text documents (Goldberg and Goodisman, 1991) and annotation (Hardock, 1991).

Previous work on the use of stroke-based devices has been most concerned with the capabilities of the pen for gestural input (MacKenzie and Chang, 1999). One of the first studies of hand-drawn gestures for simple editing tasks dates back to 1987, when Wolf and Morrel-Samuels (1987) reported that "the use of gestures is of particular interest in an interface which allows the user to write directly on the surface of a display with a stylus." Simple gestures led to good intra-subject consistency and there was consensus regarding gestures being perceived as easy to use and remember. Later on, Goldberg and Richardson (1993) introduced the general philosophy of simplifying gesture sets, motivated by the fact that simpler is faster to write, less prone to recognition error, and can be entered in an "eyes-free" manner, which requires little space onscreen.

8.1. Recognizing Gestures

Gesture recognition has its own roots in sketching and handwriting recognition. Classification algorithms for these topics include template matching (Connell and Jain, 2000; Nakayama, 1993), decision trees (Belaid and Haton, 1984; Kerrick and Bovik, 1988), neural networks (Dimitriadis and Coronado, 1995; Marzinkewitsch, 1991), hidden Markov models (Koschinski et al., 1995; Kosmala and Rigoll, 1998), parsing grammars (Costagliola et al., 2004; Mas et al., 2010), support vector machines (Bahlmann et al., 2001; El Meseery et al., 2009), or principal component analysis (Deepu et al., 2004; Zhang et al., 2010). Typically, gesture recognition takes place after a "pointer up" event, although it is possible to perform it continuously, in an incremental fashion (Bau and Mackay, 2008; Kristensson and Denby, 2011).

The most popular gesture recognizers for prototyping UIs are often based on the template-matching or instance-based approach: a query gesture is geometrically compared to a series of templates in a library of predefined gestures, using often Euclidean distance as dissimilarity measure or a Mean Square Error (MSE) score. Examples of state-of-the-art template-based recognizers among the HCI literature are \$1 (Wobbrock et al., 2007), \$N (Anthony and Wobbrock, 2010), and their newest versions Protractor (Li, 2010) and \$N-Protractor (Anthony and Wobbrock, 2012), respectively—the only difference with their previous versions is a closed-form algorithm to match gesture templates, which, in addition, provides better performance. More recently, Vatavu et al. (2012) introduced \$P, a sequential-agnostic recognizer where strokes are treated as a cloud of 2D points. Template matchers are a very viable and a relatively simple solution for recognizing gestures, and can be adapted to personalized user gestures. Nonetheless, they can be both time and space consuming on the computational side, given the size of the gesture vocabulary and the number of stored templates to define each gesture.

One of the first ideas to recognize symbols was using direction sequences from stroke motion alongside with lookup tables (Groner, 1968; Powers, 1973). Later, Rubine (1991a) and Smithies et al. (1999) introduced a variety of statistical and geometric features (aspect ratio, stroke length, etc.) as input to a simple linear classifier. Algorithms based on this parametric fashion have been implemented in a variety of recognizers, e.g., SATIN (Hong and Landay, 2000) and iGesture (Signer et al., 2007). Other toolkits for developing gestures are readily available for Java[9] or C#.[10] Parametric recognizers have been shown to perform excellently when the target gestures fit the assumed model (Li, 2010). This is one of the main reasons for adopting this approach in MinGestures.

8.2. Gestures for Text-Editing Applications

One of the pioneer products in the market implementing stroke-based gestures for text editing is the Microsoft Tablet PC,[11] whose handwriting panel provides the user with basic editing support, by pointing one or more words (by crossing them out) and then rewriting the misrecognized text. Nevertheless, in this case editing operations are limited to the context of the on-line handwritten text introduced by the user, whereas in CATTI it is also taken into account the contextual

[9]http://sourceforge.net/projects/mousegestures
[10]http://codeproject.com/KB/recipes/cmgblade.aspx
[11]http://msdn.microsoft.com/en-us/library/ms840465.aspx

information of the off-line handwritten text as well as the transcription to be supervised on the UI. Furthermore, in both cases the set of gestures is very limited, with regard to text editing, when compared to MinGestures.

On a related research line, Kristensson and Zhai (2004) described SHARK^2, an effective and fast shapewriting system based on what is known as *sokgraphs* (shorthand on keyboard as graphs). This aimed for seamless transition from visually guided tracing to recall-based gesturing. However, a shortcoming of this approach is that users must spend time learning the *sokgraph*-specific gesture for each word, which is also highly dependent of the keyboard layout employed (e.g., QWERTY, AZERTY). As in MinGestures, this approach may employ copy-mark gestures for signaling e.g. a word deletion in what is called a "stream editor", although SHARK^2 does not make use of contextual information to recognize gestures.

More closely related to the idea described in this paper, but without fully exploiting contextual information, Kristensson (2007) illustrated the use of intuitive stroke-based pen gestures for editing incorrect words outputted from a continuous shapewriting recognizer in a text-editing UI. Specifically, any sequence of words could be deleted by a crossing action, which resembles very much the MinGestures gesture for deletion. Aside from that, Kristensson (2007) relied on virtual or phantom keyboard gestures for text input. Similarly, Wang et al. (2006) considered gestures for reviewing the output of a continuous handwriting recognition system through a multimodal error correction mechanism that allowed the user to correct handwriting errors by simultaneously using pen and speech input. Gestures were used to adjust the segmentation of Chinese characters by splitting and merging, and to select text to be replaced by means of speech recognition. Therefore, in this case gesture recognition was rather simple, as it was not possible to distinguish between gestures and handwriting.

In a similar vein, several works have also been carried out in the speech transcription field. For instance, Ogata and Goto (2005) and Vertanen and Kristensson (2009) describe speech transcription systems whose UIs display the recognition results along with other competitive candidates. This way, the user who finds a word recognition error can simply select the correct word from such candidates. There are more elaborated approaches in this line, such as the one introduced by Suhm et al. (2001), which uses gestures for non-interactive post-edition of the output of a speech recognizer, or the ones presented by Wang et al. (2008) and Liu and Soong (2006), which are aimed for interactive speech error correction. As usual, only substitution, deletion, and insertion operations could be submitted by using stroke-based gestures. Then, additional information obtained from pen-based interaction was fed back to the speech recognizer; for instance, error location, error type, and so on. Kristensson and Denby (2009) reported on a longitudinal study of unconstrained handwriting input that handwriting entry as well as error rates were about the same as for QWERTY software-based keyboards. This suggested that handwriting input methods could be augmented with a complete set of gestures to provide full text editing, as MinGestures does.

On another line, many solutions have been proposed to distinguish text (or annotations) and gestures (or commands) with explicit approaches. For instance, Zeleznik and Miller (2006) devised a method in which the computer was able to distinguish gestures from free-form ink by means of "terminal punctuation", a tap issued after each gesture. On the other hand, Hinckley et al. (2005); Liao et al. (2008) proposed an approach where users could clearly indicate the current stroke type; e.g. by pressing a button or a foot pedal. Approaches in which the computer and the user collaborate to resolve ambiguous input is also possible (Saund et al., 2003). Among these, we believe that an implicit approach, as the one we use in MinGestures, is the most promising direction to tell gestures and text apart, since it does not place unnecessary burdens on the user.

9. CONCLUSION

Stroke-based applications devoted to create or modify text such as CATTI 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 MinGestures, a context-aware approach that is able to disambiguate among simple copy-mark gestures and handwritten text, with runtime efficiency as primary focus. This paper may thus serve as a reference guide prior to designing and evaluating CATTI-like applications and related text-editing MIUIs.

We have tested MinGestures in a series of scenarios and user studies, including quantitative and qualitative results, reporting on accuracy, recognition speed, and drawing performance; using both an e-pen and a computer mouse as input devices. We should mention that MinGestures alone is really straightforward but put in the context of CATTI applications is an intelligent solution. Ultimately, it can be easily integrated into prototypes that are written in virtually any programming language. Nevertheless, it is our belief that MinGestures may enable a natural and accurate interactive text edition well beyond stroke-based devices. For instance, we have already deployed MinGestures in a production-ready machine translation system. In the context of the CASMACAT project,[12] MinGestures is assisting

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

professional translators to review machine-translated text interactively. Finally, we want to mention that the reader can try the standalone version of MinGestures at http://cat.prhlt.upv.es/mg/, as well as the CATTI prototype at http://cat.prhlt.upv.es/iht/.

9.1. Future Work

From the user's perspective, we plan to speed up gesture learning by incorporating some interactive feedback mechanism to MinGestures; for instance, providing incremental information as to the current state of the recognition algorithm. In this regard, Bau and Mackay (2008); Kristensson and Denby (2011) have proposed interesting approaches that are worth pursuing. First-time users would be the most benefited target of this enhancement, who could remove the need of having to use such an interactive mechanism as they continue to work with the gesture-driven application.

From a technical perspective, we plan to enhance MinGestures with the possibility of extending its context-aware capabilities. For instance, a gesture that is driven by mouse clicks alone could have different meanings when performed outside word bounding boxes. This way, the expressiveness of gestures could be further expanded. In addition, doing so would also enable either the assignment of different gestures for issuing a given command or the widening of the gesture vocabulary.

ACKNOWLEDGEMENTS

We thank the anonymous *IwC* reviewers for the feedback provided to strengthen this manuscript. This work is supported by the 7th Framework Program of the European Union under grant agreements 600707 (tranScriptorium) and 287576 (CASMACAT), the Spanish MINECO under grant TIN2012-37475-C02-01 (STraDa), and the Generalitat Valenciana under grant ISIC/2012/004 (AMIIS).

REFERENCES

Alabau, V., Leiva, L. A., 2012. Transcribing handwritten text images with a word soup game. In: Proceedings of Extended Abstracts on Human factors in computing systems (CHI EA).

Alabau, V., Rodríguez-Ruiz, L., Sanchis, A., Martínez-Gómez, P., Casacuberta, F., 2011a. On multimodal interactive machine translation using speech recognition. In: Proceedings of the international conference on multimodal interfaces (ICMI).

Alabau, V., Sanchis, A., Casacuberta, F., 2011b. Improving on-line handwritten recognition using translation models in multimodal interactive machine translation. In: Proceedings of the Association for Computational Linguistics (ACL).

Alabau, V., Sanchis, A., Casacuberta, F., 2014. Improving on-line handwritten recognition in interactive machine translation. Pattern Recognition 47 (3), 1217–1228.

Anthony, L., Vatavu, R.-D., Wobbrock, J. O., 2013. Understanding the consistency of users' pen and finger stroke gesture articulation. In: Proceedings of the conference on Graphics interface (GI).

Anthony, L., Wobbrock, J. O., 2010. A lightweight multistroke recognizer for user interface prototypes. In: Proceedings of the conference on Graphics interface (GI).

Anthony, L., Wobbrock, J. O., 2012. $N-protractor: a fast and accurate multistroke recognizer. In: Proceedings of the conference on Graphics interface (GI).

Appert, C., Zhai, S., 2009. Using strokes as command shortcuts: Cognitive benefits and toolkit support. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Bahlmann, C., Haasdonk, B., Burkhardt, H., 2001. On-line handwriting recognition with support vector machines: A kernel approach. In: Proceedings of the International Workshop on Frontiers in Handwriting Recognition (IWFHR).

Bailly, G., Lecolinet, E., Nigay, L., 2008. Flower menus: a new type of marking menu with large menu breadth, within groups and efficient expert mode memorization. In: Proceedings of the working conference on Advanced visual interfaces (AVI).

Balakrishnan, R., Patel, P., 1998. The PadMouse: facilitating selection and spatial positioning for the non-dominant hand. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Bau, O., Mackay, W. E., 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Belaid, A., Haton, J., 1984. A syntactic approach for handwritten formula recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (1), 105–111.

Bosch, V., Bordes-Cabrera, I., Munoz, P. C., Hernández-Tornero, C., Leiva, L. A., Pastor, M., Romero, V., Toselli, A. H., Vidal, E., 2014. Computer-assisted transcription of a historical botanical specimen book: Organization and process overview. In: Proceedings of the international conference on Digital Access to Textual Cultural Heritage (DATeCH).

Buxton, W., 1988. The natural language of interaction: A perspective on non-verbal dialogues. INFOR: Canadian Journal of Operations Research and Information Processing 26 (4), 428–438.

Cao, X., Zhai, S., 2007. Modeling human performance of pen stroke gestures. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Castro-Bleda, M. J., España-Boquera, S., Llorens, D., Marzal, A., Prat, F., Vilar, J. M., Zamora-Martinez, F., 2011. Speech interaction in a multimodal tool for handwritten text transcription. In: Proceedings of the international conference on multimodal interfaces (ICMI).

Connell, S. D., Jain, A. K., 2000. Template-based on-line character recognition. Pattern Recognition 34 (1), 1–14.

Costagliola, G., Deufemia, V., Polese, G., Risi, M., 2004. A parsing technique for sketch recognition systems. In: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC).

Culotta, A., Kristjansson, T., McCallum, A., Viola, P., 2006. Corrective feedback and persistent learning for information extraction. Artificial Intelligence 170 (14–15), 1101–1122.

Deepu, V., Madhvanath, S., Ramakrishnan, A. G., 2004. Principal component analysis for online handwritten character recognition. In: Proceedings of the International Conference on Pattern Recognition (ICPR).

Delaye, A., Sekkal, R., Anquetil, E., 2011. Continuous marking menus for learning cursive pen-based gestures. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI).

Dimitriadis, Y., Coronado, J., 1995. Towards an art-based mathematical editor that uses on-line handwritten symbol recognition. Pattern Recognition 8 (6), 807–822.

El Meseery, M., El Din, M. F., Mashali, S., Fayek, M., Darwish, N., 2009. Sketch recognition using particle swarm algorithms. In: Proceedings of the 16th IEEE international conference on Image processing (ICIP).

Goldberg, D., Goodisman, A., 1991. Stylus user interfaces for manipulating text. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Goldberg, D., Richardson, C., 1993. Touch-typing with a stylus. In: Proceedings of the INTERCHI'93 Conference on Human Factors in Computing Systems.

Groner, G. F., 1968. Real-time recognition of hand-printed symbols. Pattern Recognition 1 (1), 103–108.

Hardock, G., 1991. Design issues for line driven text editing/annotation systems. In: Proceedings of the conference on Graphics interface (GI).

Hardock, G., Kurtenbach, G., Buxton, W., 1993. A marking based interface for collaborative writing. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Hinckley, K., Baudisch, P., Ramos, G., Guimbretiere, F., 2005. Design and analysis of delimiters for selection-action pen gesture phrases in scriboli. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Hong, J. I., Landay, J. A., 2000. SATIN: a toolkit for informal ink-based applications. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Horvitz, E., 1999. Principles of mixed-initiative user interfaces. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Huerst, W., Yang, J., Waibel, A., 2010. Interactive error repair for an online handwriting interface. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Jelinek, F., 1998. Statistical Methods for Speech Recognition. MIT Press, Cambridge, Massachusetts.

Johansson, S., Atwell, E., Garside, R., Leech, G., 1996. The Tagged LOB Corpus, User's Manual. Norwegian Computing Center for the Humanities.

Karat, C.-M., Halverson, C., Horn, D., Karat, J., 1999. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Kerrick, D., Bovik, A., 1988. Microprocessor-based recognition of hand-printed characters from a tablet input. Pattern Recognition 21 (5), 525–537.

Koschinski, M., Winkler, H. J., Lang, M., 1995. Segmentation and recognition of symbols within handwritten mathematical expressions. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP).

Kosmala, A., Rigoll, G., 1998. On-line handwritten formula recognition using statistical methods. In: Proceedings of the International Conference on Pattern Recognition (ICPR).

Kristensson, P. O., 2007. Discrete and continuous shape writing for text entry and control. Ph.D. thesis, Linköping University, Sweden.

Kristensson, P. O., Denby, L. C., 2009. Text entry performance of state of the art unconstrained handwriting recognition: a longitudinal user study. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Kristensson, P. O., Denby, L. C., 2011. Continuous recognition and visualization of pen strokes and touch-screen gestures. In: Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling (SBIM).

Kristensson, P. O., Zhai, S., 2004. SHARK^2: A large vocabulary shorthand writing system for pen-based computers. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Kurtenbach, G., Buxton, W., 1994. User learning and performance with marking menus. In: Proceedings of Extended Abstracts on Human factors in computing systems (CHI EA).

Kurtenbach, G. P., 1991. The design and evaluation of marking menus. Ph.D. thesis, University of Toronto.

Kurtenbach, G. P., Buxton, W., 1991. Issues in combining marking and direct manipulation techniques. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Kurtenbach, G. P., Sellen, A. J., Buxton, W., 1993. An empirical evaluation of some articulatory and cognitive aspects of marking menus. Human-Computer Interaction 8 (1), 1–23.

LaLomia, M., 1994. User acceptance of handwritten recognition accuracy. In: Proceedings of Extended Abstracts on Human factors in computing systems (CHI EA).

Leiva, L. A., Alabau, V., Vidal, E., 2013. Error-proof, high-performance, and context-aware gestures for interactive text edition. In: Proceedings of Extended Abstracts on Human factors in computing systems (CHI EA).

Leiva, L. A., Romero, V., Toselli, A. H., Vidal, E., 2011. Evaluating an interactive-predictive paradigm on handwriting transcription: A case study and lessons learned. In: Proceedings of the 35th Annual IEEE Computer Software and Applications Conference (COMPSAC).

Li, W., Hammond, T., 2012. Using scribble gestures to enhance editing behaviors of sketch recognition systems. In: Proceedings of Extended Abstracts on Human factors in computing systems (CHI EA).

Li, Y., 2010. Protractor: a fast and accurate gesture recognizer. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Liao, C., Guimbretière, F., Hinckley, K., Hollan, J., 2008. Papiercraft: A gesture-based command system for interactive paper. ACM Transactions on Computer-Human Interaction (TOCHI) 14 (4), 18:1–18:27.

Liu, P., Soong, F. K., 2006. Word graph based speech rcognition error correction by handwriting input. In: Proceedings of the international conference on multimodal interfaces (ICMI).

Long, A. C., Landay, J. A., Rowe, L. A., 1999. Implications for a gesture design tool. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

Long, Jr., A. C., Landay, J. A., Rowe, L. A., Michiels, J., 2000. Visual similarity of pen gestures. In: Proceedings of the SIGCHI conference on Human factors in computing systems (CHI).

MacKenzie, I. S., Chang, L., 1999. A performance comparison of two handwriting recognizers. Interacting with Computers 11 (3), 283–297.

MacKenzie, I. S., Tanaka-Ishii, K., 2007. Text Entry Systems: Mobility, Accessibility, Universality. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Marti, U., Bunke, H., 2001. Using a statistical language model to improve the preformance of an HMM-based cursive handwriting recognition system. International Journal of Pattern Recognition and Artificial Intelligence 15 (1), 65–90.

Marti, U., Bunke, H., 2002. The IAM-database: an English sentence database for off-line handwriting recognition. International Journal on Document Analysis and Recognition 5 (1), 39–46.

Martín-Albo, D., Romero, V., Toselli, A. H., Vidal, E., 2012. Multimodal computer-assisted transcription of text images at character-level interaction. International Journal of Pattern Recognition and Artificial Intelligence 26 (5), 1263003 (19 pages).

Marzinkewitsch, R., 1991. Operating computer algebra systems by hand-printed input. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC).

Mas, J., Llados, J., Sanchez, G., Jorge, J. A. P., 2010. A syntactic approach based on distortion-tolerant adjacency grammars and a spatial-directed parser to interpret sketched diagrams. Pattern Recognition 43 (12), 4148–4164.

Moyle, M., Cockburn, A., 2002. Analysing mouse and pen flick gestures. In: Proceedings of the SIGCHI-NZ Symposium on Computer-Human Interaction (CHINZ).

Nakayama, Y., 1993. A prototype pen-input mathematical formula editor. In: Proceedings of AACE EdMedia.

Ogata, J., Goto, M., 2005. Speech repair: Quick error correction just by using selection operation for speech input interface. In: Proceedings of Eurospeech.

Ortiz-Martínez, D., Leiva, L. A., Alabau, V., Casacuberta, F., 2010. Interactive machine translation using a web-based architecture. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI).

Ortiz-Martínez, D., Leiva, L. A., Alabau, V., García-Varea, I., Casacuberta, F., 2011. An interactive machine translation system with online learning. In: Proceedings of the Association for Computational Linguistics (ACL).

Powers, V. M., 1973. Pen direction sequences in character recognition. Pattern Recognition 5 (1), 291–302.

Raab, F., 2009. Extremely efficient menu selection: Marking menus for the Flash platform. Available at http://www.betriebsraum.de/blog/2009/, retrieved on May 2012.

Revuelta-Martínez, A., Rodríguez, L., García-Varea, I., 2012. A computer assisted speech transcription system. In: Proceedings of the European Chapter of the Association for Computational Linguistics (EACL).

Revuelta-Martínez, A., Rodríguez, L., García-Varea, I., Montero, F., 2013. Multimodal interaction for information retrieval using natural language. Computer Standards & Interfaces 35 (5), 428–441.

Rodríguez, L., García-Varea, I., Revuelta-Martínez, A., Vidal, E., 2010a. A multimodal interactive text generation system. In: Proceedings ofthe International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction (ICMI-MLMI).

Rodríguez, L., García-Varea, I., Vidal, E., 2010b. Multi-modal computer assisted speech transcription. In: Proceedings ofthe International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction (ICMI-MLMI).

Romero, V., Leiva, L. A., Toselli, A. H., Vidal, E., 2009a.

Interactive multimodal transcription of text images using a web-based demo system. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI).

Romero, V., Toselli, A. H., Vidal, E., 2009b. Using mouse feedback in computer assisted transcription of handwritten text images. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR).

Romero, V., Toselli, A. H., Vidal, E., 2011. Study of different interactive editing operations in an assisted transcription system. In: Proceedings of the international conference on multimodal interfaces (ICMI).

Romero, V., Toselli, A. H., Vidal, E., 2012. Multimodal interactive handwritten text transcription. Vol. 80. World Scientific Publishing Co. Pte. Ltd., Singapore.

Rubine, D., 1991a. Specifying gestures by example. Computer Graphics 25 (4), 329–337.

Rubine, D. H., 1991b. The automatic recognition of gestures. Ph.D. thesis, Carnegie Mellon University.

Sánchez-Sáez, R., Leiva, L. A., Sánchez, J. A., Benedí, J. M., 2010. Interactive predictive parsing using a web-based architecture. In: Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT).

Saund, E., Fleet, D., Larner, D., Mahoney, J., 2003. Perceptually-supported image editing of text and graphics. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Shilman, M., Tan, D. S., Simard, P., 2006. CueTIP: a mixed-initiative interface for correcting handwriting errors. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Signer, B., Kurmann, U., Norrie, M. C., 2007. iGesture: A general gesture recognition framework. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR).

Smithies, S., Novins, K., Arvo, J., 1999. A handwriting-based equation editor. In: Proceedings of the conference on Graphics interface (GI).

Suhm, B., Myers, B., Waibel, A., 2001. Multimodal error correction for speech user interfaces. ACM Transactions on Computer-Human Interaction (TOCHI) 8 (1), 60–98.

Tappert, C. C., Mosley, P. H., 2001. Recent advances in pen computing. Tech. Rep. 166, Pace University, available: http://support.csis.pace.edu.

Toselli, A. H., Romero, V., Pastor, M., Vidal, E., 2010. Multimodal interactive transcription of text images. Pattern Recognition 43 (5), 1814–1825.

Toselli, A. H., Vidal, E., Casacuberta, F. (Eds.), 2011. Multimodal-Interactive Pattern Recognition and Applications. Springer.

Tseng, S., Fogg, B., 1999. Credibility and computing technology. Communications of the ACM 42 (1), 39–44.

Vatavu, R.-D., Anthony, L., Wobbrock, J. O., 2012. Gestures as point clouds: a $P recognizer for user interface prototypes. In: Proceedings of the international conference on multimodal interfaces (ICMI).

Vertanen, K., Kristensson, P. O., 2009. Parakeet: A continuous speech recognition system for mobile touch-screen devices. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI).

Vidal, E., Rodríguez, L., Casacuberta, F., García-Varea, I., 2008. Interactive pattern recognition. In: Machine Learning for Multimodal Interaction. Vol. 4892 of Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Wang, L., Hu, T., Liu, P., Soong, F. K., 2008. Efficient handwriting correction of speech recognition errors with template constrained posterior (tcp). In: Proceedings of INTERSPEECH.

Wang, X., Li, J., Ao, X., Wang, G., Dai, G., 2006. Multimodal error correction for continuous handwriting recognition in pen-based user interfaces. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI).

Wobbrock, J. O., Wilson, A. D., Li, Y., 2007. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In: Proceedings of the ACM symposium on User interface software and technology (UIST).

Wolf, C. G., Morrel-Samuels, P., 1987. The use of hand-drawn gestures for text editing. International Journal of Man-Machine Studies 27 (1), 91–102.

Zeleznik, R., Miller, T., 2006. Fluid inking: augmenting the medium of free-form inking with gestures. In: Proceedings of the conference on Graphics interface (GI).

Zhang, Y., McCullough, C., Sullins, J. R., Ross, C. R., 2010. Hand-drawn face sketch recognition by humans and a PCA-based algorithm for forensic applications. Transactions on Systems, Man, and Cybernetics, Part A 40 (3), 475–485.

Zhao, S., Balakrishnan, R., 2004. Simple vs. compound mark hierarchical marking menus. In: Proceedings of the ACM symposium on User interface software and technology (UIST).