Separating and Recognizing Gestural Strokes for Sketch-Based Interfaces

Yougen Zhang, Hanchen Song, Wei Deng and Lingda Wu

Abstract Gestures are widely used as shortcuts to invoke commands in pen-enabled systems. The pen mode problem needs to be addressed when integrating gestures into sketching applications. Traditionally, explicit mode switching methods are widely employed to separate gestural strokes from normal inking strokes. However, explicit methods may have availability constraints or be inefficient under many circumstances. In this work, a new gesture interaction paradigm for sketch-based interfaces was proposed. The lasso stroke, which is widely used for selection, was modified and used to indicate the entrance of gesture mode. Our approach is intuitive and efficient; it could be a useful addition to pen-based user interfaces.

Keywords Sketch-based interface · Pen gesture · Mode switching · Gesture recognition

1 Introduction

Pen-based user interface is a primary kind of post-WIMP (window icon menu pointer) interface. It allows for fluid and expressive input based on the pen-and-paper metaphor in tasks such as design sketching, note taking, and computer

Y. Zhang (✉) · H. Song
Science and Technology on Information Systems Engineering Laboratory,
National University of Defense Technology, 410073 Changsha, China
e-mail: zhangyougen@nudt.edu.cn

H. Song
e-mail: songhanchen@hotmail.com

W. Deng · L. Wu
Key Laboratory, Academy of Equipment, 101416 Beijing, China
e-mail: dengw_021@163.com

L. Wu
e-mail: wulingda@139.com

F. Sun et al. (eds.), *Foundations and Applications of Intelligent Systems*,
Advances in Intelligent Systems and Computing 213,
DOI: 10.1007/978-3-642-37829-4_16,

operating. In recent years, with the fast development and growing popularity of electronic whiteboard, smart phones, PDAs, and Tablet PCs, pen-based interfaces are becoming more and more prevalent [1]. Gesturing is an important means of interaction in pen-enabled interfaces. The gesture strokes are widely used as shortcuts to invoke commands. Compared with traditional GUI elements (menu, toolbar button, and keyboard shortcut), pen gestures are efficient yet cheap, since they require little additional hardware resources, like screen space or hardware buttons.

This work aims at integrating pen gestures into sketching applications. There are several issues that need to be addressed. Firstly, a set of gestures should be designed according to the desired functionalities, which is specific to application. Secondly, the gesture recognizer that distinguish input gestural strokes need to be constructed. This problem has been extensively studied, and relatively robust solutions are widely available [2]. Another important issue is the so-called stroke mode problem.

Since both sketching and gesturing are done stroke by stroke with the pen, input strokes could be either ink strokes or gestural strokes. The former are data that should be stored for later processing; the latter are commands that intended for immediate interpretation and execution by the computer [2]. Therefore, it is essential to determine whether the system is in ink mode or in gesture mode for each input stroke, so as to decide how this stroke should be processed. For example, a design-by-sketch system allows users to draw sketch strokes naturally and to edit the sketch by issuing editing gestures (copy, paste, delete, etc.). In the designing process, users may switch between ink mode and gesture mode frequently and irregularly. Obviously, an ineffectual ink/gesture mode switching technique may become the bottleneck in the system usability.

The mode problem faced by these systems is a classic problem [3]. It has long been considered as an important source of errors, confusion, unnecessary restrictions, and complexity [4]. Traditional solutions to this problem simply require the user to switch modes explicitly. For example, a toolbar with icons may be employed, on which the user taps to enter the intended mode. However, the resulting round-trip time interrupts the user's attention from his/her work [5]. Pen barrel button and tablet bezel button are also often used for explicit mode switching [4] in many existing pen-based applications, usually pressing button for gesturing. However, these supplementary physical buttons are not always available. Even if a button is available, mode errors occur if the user forgets to press the button prior to inputting his/her sketch or command stroke, because the mode switching action is mentally separated with the gesture actions. That will result in a spurious ink stroke or an unexpected command recognized and executed, depending on the error type. To recover from the error, the user has to disrupt his/her task, check and modify the digital ink content, switch to the right mode, and then repeat the intended input stroke [3].

In this paper, we proposed a solution of gesture interaction for sketching interfaces, aiming at reducing the burden of mode switching on users. The main idea of our design is to use a detectable lasso stroke for indicating the entrance of

gesture mode in addition to target selection. The remainder of this paper is organized as follows: Sect. 2 briefly introduces the previous research on ink/gesture mode switching problem. In Sect. 3, we present the design of our generally applicable gesture interface; its key design concepts are discussed. Section 4 describes some issues on recognition. In Sect. 5, we conduct a preliminary evaluation. Conclusion and future study are discussed in Sect. 6.

2 Related Work

Various methods have been investigated for performing stroke mode switching. As discussed in the previous section, explicit mode switching methods are simple and widely employed, but they may have availability constraints or be inefficient under many circumstances. Therefore, efforts have also been made to the research of implicit mode switching techniques in recent years.

Implicit mode switching aims at releasing user's physical and mental burden of switching between modes manually. Strictly speaking, without knowing the mental state of the user, the problem of implicitly distinguishing strokes intended to be gestures from those intended to be ink is not computable; however, by making assumptions about the likelihood of certain interaction sequences, it is possible to build actually effective systems [6].

Nijboer et al. [7] explored using frame gestures to control rotation, translation, and scale of the drawing canvas and of stroke selections. By mapping canonical transformations (translation, rotation, scaling) of the canvas or of stroke selections to contextual gestures that are started from the canvas border or the selection frame, frame gestures enable a fluid switching between normal drawing, interaction with the drawn strokes, and interaction with the canvas, without having to switch between dedicated operating modes. A limitation of this design is the restriction by the actual interface border. Furthermore, frame gestures are just applicable to transformations but not other gesture commands.

Li et al. [4] explored using pen pressure that is available on many tablet devices to achieve an implicit mode switching. They leave the heavy spectrum of the pressure space for gesturing and preserve the normal (middle) pressure space for inking. Since users have differences in their inherent pressure spaces, personalized pressure spaces are needed. Analogously, 3D orientation of the pen was also studied to address the mode problem [8].

Saund and Lank [3] present a solution to the mode problem in pen-based programs. They offered an inferred-mode interaction protocol that avoids the prior selection of mode during inking. The system tried to infer the user's intent, if possible, from the properties and context of the pen trajectory. When the intent is ambiguous, a choice mediator for the user is offered, which can also be ignored so as to maximize the fluidity of drawing. A similar design called "Handle Flags" was proposed by Grossman et al. [5]. When the user positions the pen near an ink stroke, Handle Flags are displayed for the user to perform potential selections.

However, their techniques only avoid prior mode selection for the selection subtask. Another limitation of Handle Flags is that they rely heavily on the result of the stroke grouping algorithm, which can be complicated in unstructured diagrams.

Zeleznik and Miller proposed a design, which is called "Fluid Inking" [6], for disambiguating gestures from regular inking. This design uses two simple patterns: the prefix flicks (fast straight lines) and post-fix terminal punctuation (fast taps or short pauses). We also use a lasso stroke as gesture prefix, but it is integrated with the selection operation, and it is consistent among all gestures.

Guo and Chen investigated the recognition of handwriting-editing gestures and alphanumeric in a mixed recognition mode [9]. Gestures are mixed directly with handwriting strokes. As a result, the recognition system is prone to be confused.

3 Our Design of Gesture Interaction

Our goal is to develop a gesture interface for a sketch-based military situation marking system. As discussed in previous sections, the essence of the stroke mode problem is a compromise between the freedom of user operation and the complexity of mode inference/recognition for the system. In order to provide users with an approach of switching stroke modes quickly and conveniently, we focus on the design of fluid mode transitions according to sketching interaction patterns.

In this section, a simple analysis of gestures in the sketching interface is provided. We propose to use the lasso stroke, which is widely used as the selection gesture, to indicate the entrance of gesture mode. The new gesture issuing method is introduced.

3.1 Gesture Set Analysis

Empirical analysis shows that, besides normal inking of text and graphics, three major types of operations are desired and frequently performed in the process of military situation marking: (1)editing objects, including copy, paste, delete, move, rotate resize, or label objects. Object here may be either raw strokes or recognized text/symbols; (2) manipulating the background map, such as zoom in/out or translate the map. Map is often used as the reference in situation plotting; (3)other commands provided by the system, for example, save, redo, undo, last/next view, and so on.

We design a set of pen gestures for the sketch-based military situation marking system, as listed in Fig. 1. The red dot of each stroke is the starting point of the gesture trajectory.

Selection is one of the most elementary operations in gesture interactions. Most command involves a selection task explicitly or implicitly. For example, the target object should be specified when performing editing operations such as copy,

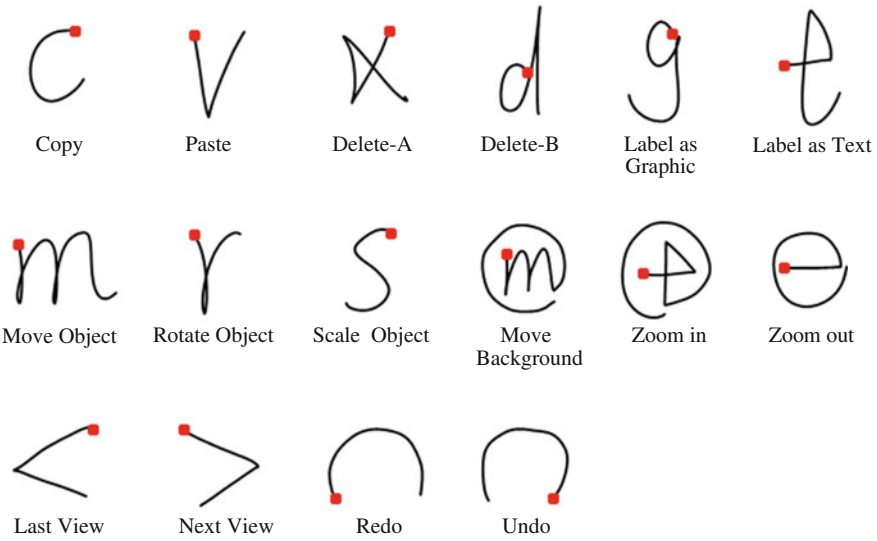

Fig. 1 Gestures designed for the sketching system

delete, etc. When pasting an object, the target location should be specified. As long as most gestures begin with selection, it guided us to the idea of entering gesture mode by detecting selections.

3.2 Gesture Issuing Method

Lassoing is widely used for selection both in pen-based interfaces and in our daily life. It allows users to select by drawing an enclosing stroke around target objects. Lassoing works well especially for small scattered targets [10], which is often the case in sketched diagrams that contain numerous short strokes.

We hereby set several conventions for issuing gestures. By default, the sketching system works in ink mode. When a gesture task is required, users can take the following steps: Firstly, draw a lasso stroke and enclose target objects or location if necessary. Secondly, draw the gesture command stroke inside the lasso. This stroke would be fed to the gesture recognizer which outputs the class label of the gesture. For most gestures, the corresponding command would be executed then. Besides instantly executable commands, there are also some gestures suitable for interactive operation. These mainly include object transformation gestures (move, resize, rotate) and background manipulation gestures (move, zoom in, zoom out). In this case, users should draw an additional stroke to specify further parameters of the command. For example, when moving an object, this stroke is used to drag the object to somewhere interactively. Analogously, the degree of

Fig. 2 Examples of lasso stroke

other object transformation gestures and background manipulation gestures could also be determined in this way.

Based on the above interaction conventions, the system can be aware of the stroke mode transitions. Once a lasso stroke is detected, the system enters gesture mode from normal ink mode and parses the following one stroke as a gesture command. Then it executes the command either instantly or interactively in response to the additional stroke. After that, the system returns to ink mode, waiting for new ink strokes or gesture interaction sessions.

So far, the problem of detecting gesture strokes comes down to detection of the lasso stroke. The usual lasso stroke we used is simply a closed stroke, which can be easily confused with ordinary circle in ink of drawing. Therefore, we use a modified lasso stroke that is reliably distinguishable from normal inking strokes. As shown in Fig. 2, our lasso stroke is extended with a straight segment inside the original closed stroke. This minor modification eliminates most confusion. The lasso stroke can be drawn either clockwise or counterclockwise, of deformable shape according to the shape/distribution of the target object(s). The next section will discuss the detection of the lasso stroke, as well as the recognition of gesture command strokes.

4 Recognition

Gesture recognition is the process of parsing a hand-drawn stroke as being one of the predefined gesture types. According to our gesture issuing method described in Sect. 3, a gesture consists of one lasso stroke and one or two command strokes.

4.1 Lasso Stroke Recognition

The lasso stroke acts as the sign of entering gesture mode and also specifies the target of the gesture in most cases. Therefore, it is critical to detect the lasso stroke in the sequence of input strokes quickly and accurately. In order to accommodate target object(s) and user habits, the lasso stroke is allowed to be deformable and be either clockwise or counterclockwise. So it is not feasible to detect the lasso stroke

by matching it to predefined templates. We treat this as a binary classification problem and used a feature-based method to test whether an input stroke s is a lasso stroke.

Several features of s were extracted as follows:

SL Stroke length of s

DE Distance between end points of s

LB Diagonal length of the bounding box of s

MR Minimum of the ratio of distance/length. Here, both the distance and the length are measured from a sample point to the start point. When s is a lasso stroke, MR locates the turning point between the sub-stroke of circle and the sub-stroke of straight segment. Thus, we denote the point that takes MR by tp; denote the sub-strokes before and after tp by s_{circle} and s_{segment}, respectively.

PI Percentage of s_{segment} that lie inside the bounding box of s_{circle}

LT Location of tp, that is, the index of tp among sample points, normalized to the range of [0,1].

SS Straightness of s_{segment}, defined as the ratio of the distance between the end points to the length of s_{segment}

We collected a set of training strokes. It consists of lasso strokes under different circumstances, normal inking strokes, and inking strokes that may confuse with the lasso stroke (for example, circle, symbol θ and φ). Then a decision tree classifier was trained. This classifier works well with precision and recalls of 99.4 and 97.2 %, respectively, in 10-fold cross-validation, consuming less than one millisecond per stroke (Fig. 3).

Once an input stroke was classified as lasso stroke, it would be highlighted in different color and stroke width. This provides users with instant feedback on the state of stroke mode switching, allowing users to recover immediately from possible false rejection or false acceptance of lasso stroke classification. The lasso stroke is generally distinguishable from normal inking strokes. In rare cases when a lasso-like inking stroke needs to be drawn, users can draw a check "$\surd$" on that stroke (yet highlighted) immediately after it was drawn (within the time-out period). Then it would be stored as an inking stroke and displayed in normal color.

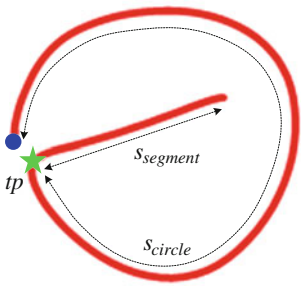

Fig. 3 Illustration of feature extraction

For a lasso stroke, we compute its bounding box and convex hull. Then the enclosed objects would be identified and highlighted, indicating the scope of selection. The bounding box and convex hull would also be used later to determine the size and location of the gesture, as well as the relative position between the lasso stroke and its succeeding command stroke.

4.2 Command Stroke Recognition

Since the work of Rubine, numerous pen gesture recognition methods have been proposed, falling into one of two main categories: template based and feature based. Some of the famous methods are Dollar 1, Protractor, and 1¢. The performance of existing methods is reasonably acceptable; therefore, we did not focus on the recognition of unistroke gesture commands. The method we adopted is similar to Dollar 1 developed by Wobbrock et al. except for some modifications in scaling and rotation.

4.3 Gesture Execution

When defining a gesture, each command stroke is mapped to a command. Once the command stroke is recognized, and the corresponding command could be executed immediately in case of instant gesture. If the command stroke corresponds to an interactive gesture, the system waits for the additional stroke, which drives the gesture execution interactively, providing users with feedback of the operation result. The way of parsing the additional stroke is gesture specific.

Figure 4 gives two examples of gesture interactions using the proposed method. The first one is a delete gesture containing a lasso stroke and the command stroke. It is executed directly. Figure 4b illustrates an interactive gesture. When the command stroke is recognized as the "Move," the system moves the selected strokes following the pen tip in real time when the succeeding stroke is drawn.

5 Evaluation

A preliminary user study was conducted to evaluate the performance of our proposed gesture interface and gain user feedback. We implemented a prototype sketch-based military situation marking system with our gesture interface. A traditional gesture interface was also implemented and tested for comparison. It employs two toolbar buttons for explicit mode switching, and it uses the common circling stroke for object selection. A set of 16 gesture commands (as listed in Fig. 1) were tested. However, several of them were not executable. In other words,

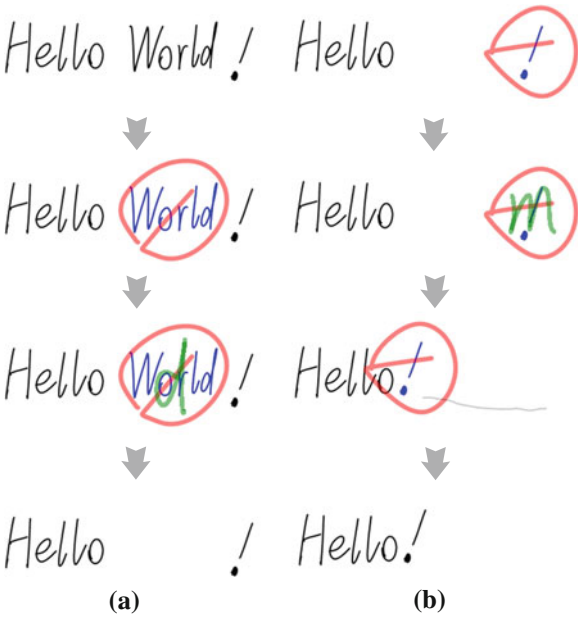

Fig. 4 Two examples of gesture interactions. **a** Delete. **b** Move

the recognition result of these commands were displayed but not executed, because the implementation of them is rather tedious and beyond the scope of this paper.

Six participants took part in the evaluation; all of them were experienced in computer operation, but had little or no experience of using pen-enabled computers except for smart phones. A Lenovo ThinkPad X200 tablet was used as sketching input device. Participants were instructed about the use of our sketch-based marking system on the tablet for several minutes. Then they were asked to perform two gesturing tasks. The first task is to test the supported gestures separately on a given sketched diagram. In the second task, participants were asked to sketch a diagram under textual guidance on operations of inking (draw graphics and write text labels) and gesturing. Upon completion of the tasks, participants were encouraged to make comments and suggestions to obtain additional feedback.

We observed a total of about 400 gesture interactions using each method. When using the proposed method, twelve lasso strokes were misclassified as ink strokes (false negatives). They were mainly careless lasso strokes when users had not yet adapted. There were also three misclassified ink strokes (false positives); they were all corrected automatically since their subsequent strokes were drawn outside of them, thus no accidental gesture was triggered. As for the traditional method, seven mode switching operations were neglected. Thirty-eight gestures were recognized incorrectly, mainly due to errors of our simplified command stroke recognition algorithm, which adopts nearest neighbor matching.

According to the comments and suggestions, participants stated that the proposed gesture issuing method is indeed more complex, but it is still quite intuitive and very easy to memorize and use. Nevertheless, the proposed approach is more efficient because it saves the round-trip time of mode switching. Moreover, the traditional method is error prone since mode switching may be neglected, while in the proposed approach mode, transitions are automatic, allowing users to concentrate on inking and gesturing.

The user study also helped us in identifying some limitations and possible improvements of our approach. A limitation of our lasso-based selection is its poor performance under extreme conditions. For example, it is difficult to select individual objects using a lasso in densely crowded conditions. Furthermore, one participant stated that it is inefficient to draw a lasso stroke that encloses a large-size object (like a long curving stroke). In these cases, the tapping-based selection method is more desirable. We consider adding a select command to our gesturing framework, so as to support more flexible selection. Moreover, we can identify a large-size object as selected if it is covered by the lasso stroke in its center over a certain area. Another suggested improvement is to allow for manual explicit mode switching as a complement to the proposed approach, so as to address the case of consecutive gesture interactions.

6 Conclusions and Future Work

We have presented a novel gesturing approach for sketch-based interfaces. The main advantage over traditional explicit mode switching approaches is that it supports fluid mode switching without extra hardware requirements. This makes it applicable to various pen-enabled systems ranging from tablets and smart phones to electronic whiteboards. Moreover, our gesture issuing method is consistent among gestures, thus lighten the burden of memory load compared to the method of [6]. The preliminary evaluation results indicate that participants learned to use our interface in a short period of time.

Our future work will focus on the personalization of command strokes. Since we have used template-based approach for recognizing these unistroke gestures, it is straightforward to define personalized gestures by storing user-specific templates. However, some issues remain to be studied, for example, how to help user choosing gestures so as to avoid possible conflict with existing ones and how to organize templates efficiently. Furthermore, the presented user study was informal so far, and further study is necessary in order to gain more useful feedback.

Acknowledgments This work is supported by the National Science Foundation of China under Grant No. 61103081.

References

1. Johnson G, Gross MD, Hong J, Do EY-L (2009) Computational support for sketching in design: a review. Found Trends Human Comput Interact 2:1–93
2. Appert C, Zhai S (2009) Using strokes as command shortcuts: cognitive benefits and toolkit support. 27th international conference on Human factors in computing systems. ACM, Boston, pp 2289–2298
3. Saund E, Lank E (2003) Stylus input and editing without prior selection of mode. 16th annual ACM symposium on user interface software and technology, Vancouver, Canada. ACM, pp 213–216
4. Li Y, Hinckley K, Guan Z, Landay JA (2005) Experimental analysis of mode switching techniques in pen-based user interfaces. SIGCHI conference on human factors in computing systems, Portland, OR, USA, pp 461–470
5. Grossman T, Baudisch P, Hinckley K (2009) Handle flags: efficient and flexible selections for inking applications. Graphics interface 2009, Canadian information processing society, Kelowna, British Columbia, Canada, pp 167–174
6. Zeleznik R, Miller T (2006) Fluid inking: augmenting the medium of free-form inking with gestures. Graphics interface 2006. Canadian information processing society, Quebec, Canada, pp 155–162
7. Nijboer M, Gerl M, Isenberg T (2010) Exploring frame gestures for fluid freehand sketching. Seventh sketch-based interfaces and modeling symposium. Eurographics Association, Annecy, France, pp. 57–62
8. Tian F, Jiang Y, Dai G, Wang H (2009) Instruction method based on stroke tail gesture. In: CAS, I.o.S. CN 200910080176
9. Guo F, Chen S (2010) Gesture recognition techniques in handwriting recognition application. 12th international conference on frontiers in handwriting recognition, Kolkata, pp 142–147
10. Mizobuchi S, Yasumura M (2004) Tapping vs. circling selections on pen-based devices: evidence for different performance-shaping factors. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, Vienna, Austria, pp 607–614