GeCCo: Finger Gesture-based Command and Control for Touch Interfaces

Sriganesh Madhvanath, Dinesh Mandalapu, Tarun Madan, Naznin Rao, Ramesh Kozhissery

HP Laboratories
HPL-2012-238

Keyword(s):

gesture recognition; touch gestures; mode-switching; adaptation

Abstract:

With touch-based interfaces becoming commonplace on personal computing devices ranging from phones and slates to notebook and desktop PCs, a number of common tasks that were once performed using mouse or keyboard input now need to be performed using fingers on the touch surface. Finger-drawn gestures offer a viable alternative to desktop and keyboard shortcuts as shortcuts for common tasks such as launching of applications and navigation of large media collections. In order to be truly effective, the interface for definition, management and invocation of gestures should be highly intuitive, and optimized for the device. In particular, the process of invoking gestures should be seamless and natural. Further, the recognition of gestures needs to be robust for the specific user. In this paper, we describe GeCCo (Gesture Command and Control), a system for personalized finger gesture shortcuts for touch-enabled desktops and trackpad-enabled notebook PCs. One of the key issues addressed in the design of GeCCo is that of mode switching in the context of notebook PCs. We describe a user study to decide between different interactions for mode switching. The interactions are designed such that mode switch and gesture can be simultaneously indicated. Since new gestures may be defined by the user at any time, statistical pattern classification techniques which require large numbers of training samples for each gesture are not useful. Instead we use nearest-neighbor classification with Dynamic Time Warping (DTW) distance, and a writer adaptation scheme for improving accuracy to desired levels. We conclude the paper with experimental results and some thoughts on next steps.

External Posting Date: December 6, 2012 [Fulltext] Approved for External Publication
Internal Posting Date: December 6, 2012 [Fulltext]

GeCCo: Finger Gesture-based Command and Control for Touch Interfaces

Keep this place blank at the time of first submission

Keep this place blank at the time of first submission

***Abstract*—With touch-based interfaces becoming mainstream on personal computing devices ranging from phones and slates to even notebook and desktop PCs, a number of common tasks that were once performed using mouse or keyboard input now need to be performed using fingers on the touch surface. In this paper, we focus on the use of finger gestures as shortcuts for tasks such as command and control of applications, navigation of large collections. A keyboard shortcut or desktop icon for an application may be substituted with a finger gesture e.g. tracing a gesture 'O' launches MS Outlook. Such gestures may be used even on a conventional notebook PC using the trackpad. However in order to be truly effective, the interface for definition, management and invocation of gestures should be highly intuitive, and customized to the available device hardware. In particular, it should be possible for the user to define and modify gestures as and when required, and the process of invoking gestures should be seamless and natural. Further, the recognition of gestures needs to be robust for the specific user. We describe GeCCo (Gesture Command and Control), an application prototype we have created for touch-enabled devices that allows the user to define his or her own gesture shortcuts for applications. One of the key issues addressed in the design of GeCCo was that of mode switching in the context of notebook PCs. We describe a user study conducted to decide between different interactions for mode switching. The interactions were designed such that mode switch and gesture can be simultaneously indicated. Since new gestures may be defined by the user at any time, we cannot use statistical pattern classification techniques which require large numbers of training samples for each gesture. Instead we use nearest-neighbor classification with Dynamic Time Warping (DTW) distance, and a writer adaptation scheme for improving accuracy to desired levels. We conclude the paper with experimental results and some thoughts on next steps.**

***Index Terms*—gesture recognition, touch gestures, mode-switching, adaptation**

I. INTRODUCTION

Touch-based interfaces have come a long way in the last decade. Historically, the most common interaction with touch surfaces has been the pressing of soft buttons (e.g. at the ATM). However with the advent of highly sensitive and accurate capacitive touchscreen technology, devices such Apple iPhone and Microsoft's Surface [4] have introduced a large range of single and multi-touch gestures such as flicking, dragging, scrolling, pinching and so on, and these have made interaction with computing devices easier and more pleasurable than before. Consequently touch-based interfaces are becoming mainstream on a broad spectrum of personal computing devices ranging from phones and slates to notebooks and desktop PCs.

This shift has meant that a number of common tasks that were once performed using mouse or keyboard input now need to be performed using fingers on the touch surface, and many usability issues are yet to be addressed. In order to launch an application for instance, the user typically has to scroll or navigate through many screens or levels of icons. Such issues clearly detract from the fundamental value proposition and overall experience of using such touch-based interfaces, which are meant to improve ease of use and provide simpler interfaces to technology.

In this paper, we focus on the use of *finger gestures* – shapes drawn on the touch surface with a finger – where the shape has a certain meaning. Clearly this enables the use of a gesture to perform an "action" associated with a gesture, and in its simplest form, is a "shortcut". Navigation may also be supported by having gestures for specific folders, or by interpreting gestures as text characters and using the recognized text prefix to help navigate lists. In principle, the gesture shapes can be highly personalized, performed naturally, rapidly and conveniently using a finger or stylus, and allow random access - unlike explicit buttons, desktop shortcuts or keyboard shortcuts, which require seeking or scrolling to "find the shortcut". Further, since the gesture shapes and their mappings may be defined by the user and modified at will, the expressiveness of such gestures is potentially higher than what is possible with dedicated buttons. With the proliferation of touchscreens and touch sensors, such gestures can be used with a variety of devices such as cameras, printers, notebooks, touch-enabled PCs, point-of-sale terminals, phones and slates.

The idea of using gesture shapes for command and control is not new. StrokeIt[1] and OptiMoz[2] are examples of products or open-source projects that use pen or mouse

based gestures as shortcuts to system actions. In addition, several web browsers support the use of simple gestures for common browsing actions such as "back" and scrolling, via plug-ins. Some of these systems only allow limited gesture shapes, e.g. as sequences of up, down, left and right mouse movements [3]. Some have predefined mappings for a large number of applications and require the user to memorize them.

In order for gestures to be truly effective, there are several issues to be addressed: (i) The interface for definition, management and invocation of gestures should be highly intuitive, and customized to the available device hardware. In particular, it should be possible for the user to define and modify gestures as and when required, and the process of invoking gestures should be seamless and natural (ii) the recognition of gestures needs to be robust for the specific user.

In this paper we describe GeCCo (Gesture Command and Control), a system that we have built for finger gesture-based command and control, that attempts to address these issues. An overview of the functionality enabled by the system is presented in Section II. Some of the design issues such as gesture invocation and management are discussed in Sections III and IV. The approach used for robust recognition of finger gestures is outlined in Section V, and the system architecture of GeCCo discussed in Section VI. Conclusions and directions for future work are presented in the final section.

II. GeCCo Overview

GeCCo is a system for personalized finger gesture shortcuts for touch-enabled desktops and trackpad-enabled notebook PCs, implemented as a Windows application. Users may define their own gesture shapes and map them to actions. For instance, tracing a gesture 'm' can launch a media player (Figure 1). The actions may in general be any system actions such as launching applications, going to specific websites, or controlling the volume of the device without the use of the keyboard/mouse. Navigation may also be supported by having gestures for specific folders, or by interpreting gestures as text characters and using the recognized text prefix to help navigate lists. Gestures may also be context-dependent, i.e., mean different things in different contexts. In an example of navigation of a large music collection, the gesture "t" is mapped to the character "t" of the keyboard which enables the user to browse song titles in Windows Media Center software without a keyboard.

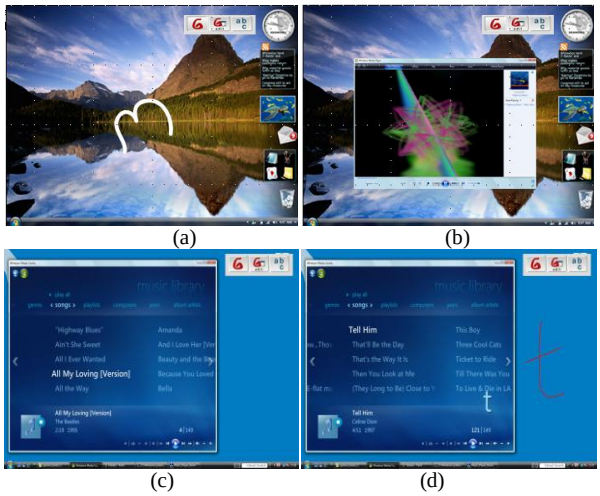

Fig. 1: (a) Enabling Gesture Mode by clicking the GeCCo button at top right and drawing the Gesture (b) application (media player) launched (c & d) drawing a character gesture to select a song to play

As mentioned in the introduction, the primary benefit of GeCCo is that gesture shapes can be highly personalized, performed naturally, rapidly and conveniently using a finger or stylus, and allow direct access to actions as opposed to scrolling through icons and desktop shortcuts. Further, the expressiveness of such user-defined gestures could exceed what is possible with dedicated buttons or keyboard shortcuts.

Although designed primarily for touch-enabled desktops and conventional notebooks, GeCCo could be used with a variety of touchscreen-enabled devices such as cameras, printers, point-of-sale terminals, mobile phones and slates.

A. Gesture management in GeCCo

GeCCo provides a management GUI with different tabs associated with different management tasks, such as adding new gestures, mapping gestures to commands, changing gestures, deleting gestures and so forth (Figure 2). The GUI can be accessed from a taskbar icon or floating bar (Figure 3). The interface is thus minimal and non-obtrusive.

The interface allows a new gesture to be defined by drawing it twice. GeCCo immediately notifies the user if the gesture is too "similar" to a gesture that is already defined.

B. Using Gestures

The user may draw a gesture anywhere on the screen or trackpad, and at any scale. GeCCo interprets the gesture and performs the corresponding action. When the gesture is not recognized with sufficient confidence, a menu of available gestures and mapped commands is presented to the user so that the user may explicitly select the intended command. In

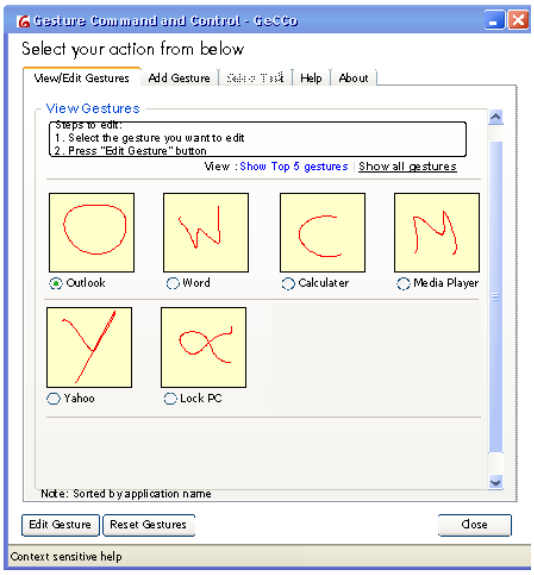

Fig. 2: GeCCo UI for Viewing/Editing Gestures

Fig. 3: Floating toolbar on Touch-based PC. The buttons correspond to Gesture Mode, Edit Gestures and Character Mode.

the rare event that a gesture is incorrectly recognized and GeCCo carries out an unintended action, the user can invoke a dialog to indicate the intended gesture. User feedback in both cases is used to adapt the recognizer to the user's writing style.

III. Gesture Invocation

Since in general the primary pointing device (such as the trackpad on a notebook, or the touchscreen on a PC or other device) needs to be "overloaded" for gesturing, one of the critical issues is the user effort to indicate gesturing (as distinguished from pointing or normal mouse movements) and to actually draw the gesture composed of one or more strokes. Such *mode switching* is a well-studied problem especially in the pen computing literature [9].

We have explored different schemes depending on the device and its touch hardware for triggering gesture mode. In the case of a touchscreen, GeCCo utilizes an onscreen widget that the user taps to invoke the gesture mode (Fig 3). At the end of the gesture (typically identified by a timeout), GeCCo automatically switches back to mouse mode.

However, there are fewer studies on the use of gestures within the context of devices such as conventional notebook PCs. We therefore conducted a study to examine mode switching from a conventional touch mode using trackpad of a standard notebook PC to gesture mode and back. Many of the brainstormed methods for mode switching, e.g. pressing a dedicated key(s) to get into gesture mode and pressing it again to get out of the gesture mode, required explicit effort on part of the user, were not intuitive and posed usability issues. Previous studies done around switching techniques in pen/stylus based interfaces have also proven that actions where the user must conventionally specify an intended mode before performing a task, end up being barriers to the usability of such systems [7, 8, 9]. In order to avoid explicit mode switching, we have come up with two different interactions, where in gesture indication and mode switching happen simultaneously. The first one uses a hardware key-trackpad combination and the second uses only the trackpad.

A. Interactions

The two methods of indicating a gesture were tested: (i) use of "Ctrl" key [left Ctrl key was used for the study] and single finger on the trackpad, and (ii) use of the index and middle finger in combination on the trackpad.

B. Apparatus

The experiment was conducted using a Windows notebook PC with a trackpad made by Synaptics. The trackpad supported only mouse movements; multi-touch gestures were not supported. It had a left, centre and right click buttons on the bottom and a scroll sensor on the right side. The scroll area was disabled during gesture mode to make the entire trackpad space available for making the gesture. The usable touch area on the trackpad including the scroll area was 3.3". The sessions were video recorded for post analysis. The use of two fingers to gesture was detected using the touch area reported by the Synaptics driver.

C. Participants

We recruited 28 participants from within the office environment for this study. Out of these, four participants were used for pilots and the remaining 24 for the study. The participants were notebook PC as well as desktop PC users from different domains such as sales, accounting, administration, design research, financial analysts and operations. We had 19 males and 5 females between the ages of 22 to 49 years. 12 participants said they always used an external mouse with a notebook PC, while 7 said they always used the trackpad. All participants received a gift voucher at the end of the study.

D. Process

The study sessions were 30 to 45 mins in duration. Each session included an introduction to the study, signing the informed consent form, collecting information about participant demographics and notebook PC usage, demonstration of mode switching mechanisms followed by practice of each, followed by the three actual tasks. At the end of the session, each participant was asked which method (s)he preferred and why.

Three tasks were designed for the study, each requiring six occurrences of switching to gesture mode. This resulted in 3 tasks x 6 mode switches per task x 24 participants = 432 instances of mode switching.

Table. 1. Gesture-to-action chart showing the gestures used in the study and the corresponding actions that would take place on making the gesture.

Gesture	Action assigned
←	Opens Firefox application with blank window
→	Opens Notepad application with new file
↑	Opens Media Player with a play list displayed
↓	Opens Yahoo messenger

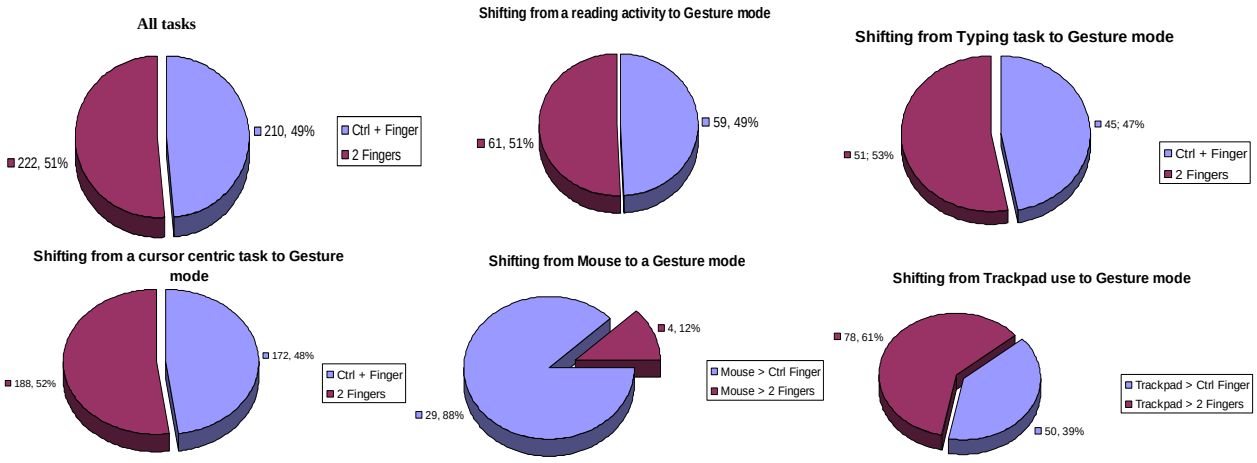

Fig. 4: User preference between mode switching methods for (a) all tasks (b) reading-centric tasks (c) typing-centric tasks (d) cursor-centric tasks (e) cursor-centric tasks when cursor control is using a mouse (f) cursor-centric tasks when cursor control is using the trackpad

Four gestures were used for testing in the study. These were kept simple and assigned simple actions as shown in Table 1. Each gesture was a straight line (vertical or horizontal) and the direction from which it began decided the corresponding action.

E. Task design and execution

The aim of task design was to keep the tasks natural and yet be able to identify any emerging pattern(s). Three types of tasks were designed – *typing centric*, *reading centric* and *cursor-centric*. The frequency of mode switching was balanced across the three. For example, a typical cursor-centric task would include: (i) One generic instance of gesture mode not preceded by any other action, (ii) One instance of gesture mode following a reading activity (iii) One instance of gesture mode following a typing activity (iv) Three instances of gesture mode following a cursor-centric (browse/select) activity. This structure was followed for the other two tasks as well. This allowed us to evaluate whether the preceding activity had any impact on the choice of mode switching mechanism.

A 3x3 Latin Square was used to counterbalance the order of the tasks. Participants were given the freedom to use any of the two methods (Ctrl+finger or two fingers) for gesture mode switching during the tasks. During the study, we noted the switching method used each time by the participant. We also noted the input interaction at each step of each task - whether the participant used the external mouse, keyboard shortcut, trackpad or joystick to perform an action.

At the end of each task, the participant was asked to comment on the method (s)he preferred for gesture mode switching. The stated preference was verified against the one that was empirically observed during the task. This also helped us understand the reasons for a participant preferring a particular method.

F. Results and Discussion

Overall, no clear polarity was found in preference of one mode switching method over the other (Figure 4a). It was also found that preference for a particular switching method did not depend on the type of activity preceding it (Figure 4b,4c,4d). However, in the case of cursor-centric activities, there was clear dependence on the cursor-control mode (i.e. mouse versus trackpad) in use preceding the mode switch (Figure 4e, 4f).

Considering that the use of mouse is preferred for most cursor centric tasks [10] [11], there are some important implications for GeCCo-like systems arising from this study. The proximity of the cursor-control device seems to play an important role in the work environment of notebook PC users. This is also apparent from the user comments stating their reason for preference of a mode switching method. Since mouse and trackpad are located at a distance fron one another in a notebook PC environment, the effort to move from mouse to trackpad to make a gesture might be a significant one for mouse users of the notebook PC population. This in turn may affect the usage of gestures by such users.

IV. ROBUST RECOGNITION OF GESTURES

Numerous studies have shown that near-perfect recognition is a prerequisite for user adoption of recognition-based interfaces. High recognition accuracy is a challenge in the GeCCo context since it is not reasonable to collect a large number of gesture samples in advance for training (especially when new gestures may be defined at will), and hence statistical pattern recognition techniques cannot be applied. On the other hand, GeCCo is meant to be personalized and we can hope to learn the user's style over time.

We have therefore used a prototype-based classification technique (k-nearest neighbors based on Dynamic Time

Warping distance), which computes the distance of the test sample to stored prototypes. In order to improve the accuracy of recognition and enable personalization, we use an adaptation strategy based on user feedback that involves adding prototypes and modifying existing prototypes based on different conditions. In essence, misrecognized gesture samples are added to the prototype set. In order to adapt the existing prototypes to the user's style of writing, we modify the existing prototype set using Learning Vector Quantization (LVQ) [5]. These techniques are available from the open source Lipi Toolkit [6], and are described briefly below.

A. Preprocessing and Feature Extraction

Each gesture is first preprocessed to address variability in sensor resolution and sampling rates. The sequence of (x,y) digitizer coordinates between "pen-down" and "pen-up" are typically uniformly sampled in time when they are produced by the digitizer. They are resampled using interpolation and represented as a sequence of points that are equidistant along the trajectory. The gesture is also scaled such that the maximum of its x and y-extents is equal to a predetermined box size, while its original aspect ratio is preserved. Each gesture is finally represented as a sequence of points, where each point contains not only the size-normalized x and y coordinates, but also additional features such as slope and curvature at that point [15].

B. Template matching

Each gesture class is modeled by a small set of prototypes or templates. The initial set is obtained from the samples provided by the user while defining a new gesture. Thereafter new samples are added or existing samples modified based on the adaptation strategy. Given a new gesture sample, its distance is computed from all of the stored prototypes of all defined gestures, using Dynamic Time Warping (DTW) distance. The distance computation uses Dynamic Programming to compute the lowest-cost alignment of the feature sequence computed from the test sample with the feature sequence corresponding to a stored prototype, and computing the normalized sum of the Euclidean distances between matched feature points. The k nearest prototypes to the test sample are then determined, and the majority class among them is declared to be the matched gesture class, with a certain confidence measure that reflects the distribution of the majority class among the k nearest neighbors.

C. Adaptation

GeCCo uses an adaptation strategy called Add + LVQ [5] which we empirically determined to equal or outperform other adaptation strategies, while limiting the number of stored prototypes per gesture class to a predefined maximum. This is an important consideration for real-time performance given that template matching is linear in the number of prototypes and quadratic in the length of the feature sequences.

When a test sample is misclassified by GeCCo and the user provides feedback about the intended class, the sample is added to the true class as an additional prototype. However, when a sample is not recognized with high confidence and the user provides feedback about the intended class using the context menu, the sample is added to the prototype set only if maximum prototype count for that class has not been reached. Otherwise it is used to modify its nearest sample among the existing prototypes [16]. In addition, successfully recognized samples are also used to modify the existing prototypes. Over a period of use, the prototypes start to accurately reflect the gesture shapes and style of the user.

D. Evaluation

We have performed an evaluation of recognition accuracy and the impact of writer adaptation. In a simulation involving 50 gesture classes extracted from a dataset of handwritten Tamil characters [17] (where we pretend that the characters are gestures) from 10 writers, we found that writer-specific accuracy increased from an average of 86% when only one sample of each gesture was available, to 94% when an additional sample was added as a prototype, to 96% after three samples. This suggests that very high accuracies can be achieved, considering that we can expect far fewer than 50 gesture classes in practice, and there is the possibility of rejecting ambiguous gestures both at the time of gesture definition and gesture invocation.

V. System Architecture

The architecture and components of GeCCo are shown in Figure 5. GeCCo runs as a background process and appears as an icon in the system tray and/or a floating toolbar. When in gesture mode, the system gets data from the touchscreen or trackpad driver when the user enters a gesture. The corresponding "digital ink" is passed to Gesture Shape Recognition Engine (SRE) which recognizes the gesture shape and returns the set of matching shape IDs with confidence values. The SRE is built using Lipi Toolkit [6] which implements the gesture recognition and adaptation schemes described above.

The GeCCo controller calls the application or the operating system to execute the command corresponding to the best matched gesture. When the gesture is not recognized with sufficient confidence, a menu of available gestures and mapped commands is presented to the user so that the user may explicitly select the intended command. This feedback is used to adapt the recognizer to the user's writing style.

VI. Conclusions and Next Steps

In this paper, we described GeCCo, a system for personalized finger gesture-based shortcuts suitable for a variety of devices with touch sensing capabilities. GeCCo allows the user to define arbitrary gesture shapes and map them any available command or action. While many such systems have been created, we believe GeCCo is significant among such systems in its ability to adapt to the way

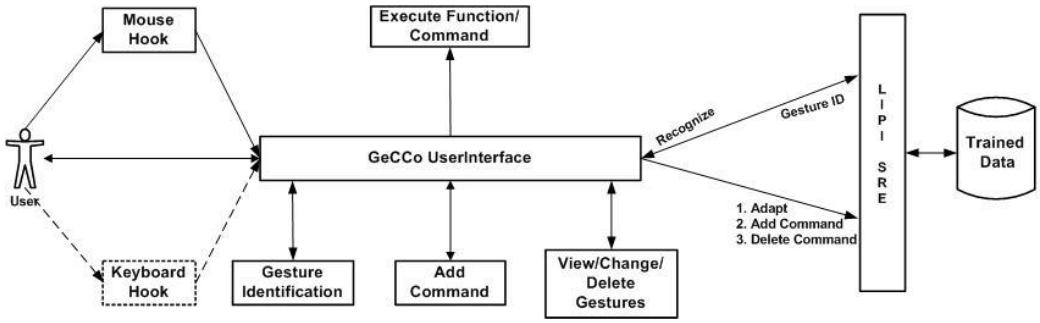

Fig. 5: GeCCo: Architecture and Components

gestures are written by the users. Another important focus of our work has been the ease of gesture invocation, especially with respect to mode switching in a notebook PC environment. We described a study that compares two different mechanisms for mode switching. Through the study did not yield any significant differences in preference of one method over the other, when taken in the context of cursor-centric activities, an important factor that impacts the preference is the proximity of the cursor-control device.

We believe there are a number of research questions to be answered before a system such as GeCCo can become truly mainstream. We would like to use the GeCCo prototype to conduct broad user studies to answer key questions such as: How many gestures can a user remember? What are the top few tasks/functions? What kinds of gestures do users select (simple vs. complex, single vs. multi-stroke)? Would users want gestures to be customizable or have predefined shapes? Would users like the frequently used commands for which gestures may be useful, to be recommended by the system based on observed usage? As touch screens proliferate, it will be interesting to investigate the role of GeCCo-like systems for touchscreen-enabled printers, point-of-sale terminals, as well as slates and mobile phones, where gesture-based shortcuts can reduce the burden of navigating deeply nested menus of options. Finally, we are continuing to explore alternative features and adaptation strategies to improve the recognition of gestures.

ACKNOWLEDGMENT

Omitted for blind review

REFERENCES

[1] Strokeit – Mouse Gestures for Windows, http://www.tcbmi.com/strokeit/

[2] Optimoz – Mouse Gestures for Mozilla Firefox, http://optimoz.mozdev.org/

[3] All in One Gestures – Plug in for Mozilla Firefox, https://addons.mozilla.org/en-US/firefox/addon/all-in-one-gestures/

[4] Microsoft Surface Computing - http://www.microsoft.com/surface/

[5] Adaptive Methods for On-Line Recognition of Isolated Handwritten Characters, Ph.D. Thesis, Vuokko Vuori, Helsinki University of Technology, 2002

[6] Lipi Toolkit – An Open Source Toolkit for Handwriting Recognition, http://lipitk.sourceforge.net

[7] Saund, E. and Lank, E., "Stylus Input and Editing Without Prior Selection of Mode", Proceedings of UIST'03, pp. 213 - 216.

[8] A. Sellen, G. Kurtenbach, and W. Buxton, "The Role of Visual and Kinesthetic Feedback in the Prevention of Mode Errors", Proceedings of Human-Computer Interaction, 1990, 667-673.

[9] Li, Y., Hinckley, K., Guan, Z. and Landay, J., "Experimental Analysis of Mode Switching Techniques in Pen-based User Interfaces", CHI 2005, April 2–7, 2005, Portland, Oregon, USA.

[10] Atkinson, S.,Woods, V., Haslam R.A., Buckle, P., "Using non-keyboard input devices: interviews with users in the workplace"

[11] Shanis, J., Hedge, A., "Comparison of 3 input technologies: Mouse, Trackpad and Multitouch"

[12] Hinckley, K., P. Baudisch, G. Ramos, and F. Guimbretiere. Design and Analysis of Delimiters for Selection-Action Pen Gesture Phrases in Scriboli. Proceedings of CHI'05, pp. 451 - 460.

[13] Alvarado, C. and R. Davis. SketchREAD: a multi-domain sketch recognition engine. Proceedings of UIST'04, pp. 23 - 32.

[14] Rubine, D. Specifying Gestures by Example. In Proceedings ACM SIGGRAPH'95 Conference on Computer Graphics. 1995. 329-337

[15] Stefan Jäger and Stefan Manke and Jürgen Reichert and Alex Waibel. Online handwriting recognition: the NPen++ recognizer, International Journal on Document Analysis and Recognition. 2001, Vol 3, pp. 169 - 180

[16] Vandana Roy, Sriganesh Madhvanath, Anand S, Ragunath R. Sharma. A Framework for Adaptation of the Active-DTW Classifier for Online Handwritten Character Recognition. ICDAR 2009, pp. 401 – 405

[17] Isolated Handwritten Tamil Character Dataset http://www.hpl.hp.com/india/research/penhw-resources/tamil-iso-char.html