

Linköping Studies in Science and Technology

Dissertation No. 1106

# **Discrete and Continuous Shape Writing for Text Entry and Control**

**Per Ola Kristensson**

Department of Computer and Information Science  
Linköping University  
581 83 Linköping, Sweden

Linköping 2007

ISBN 978-91-85831-77-7  
ISSN 0345-7524

Printed by LiU-Tryck, Linköping 2007

# Abstract

Mobile devices gain increasing computational power and storage capabilities, and there are already mobile phones that can show movies, act as digital music players and offer full-scale web browsing. The bottleneck for information flow is however limited by the inefficient communication channel between the user and the small device. The small mobile phone form factor has proven to be surprisingly difficult to overcome and limited text entry capabilities are in effect crippling mobile devices' use experience. The desktop keyboard is too large for mobile phones, and the keypad too limited. In recent years, advanced mobile phones have come equipped with touch-screens that enable new text entry solutions. This dissertation explores how software keyboards on touch-screens can be improved to provide an efficient and practical text and command entry experience on mobile devices. The central hypothesis is that it is possible to combine three elements: software keyboard, language redundancy and pattern recognition, and create new effective interfaces for text entry and control. These are collectively called "shape writing" interfaces. Words form shapes on the software keyboard layout. Users write words by articulating the shapes for words on the software keyboard. Two classes of shape writing interfaces are developed and analyzed: discrete and continuous shape writing. The former recognizes users' pen or finger tapping motion as discrete patterns on the touch-screen. The latter recognizes users' continuous motion patterns. Experimental results show that novice users can write text with an average entry rate of 25 wpm and an error rate of 1% after 35 minutes of practice. An accelerated novice learning experiment shows that users can exactly copy a single well-practiced phrase with an average entry rate of 46.5 wpm, with individual phrase entry rate measurements up to 99 wpm. When used as a control interface, users can send commands to applications 1.6 times faster than using de-facto standard linear pull-down menus. Visual command preview leads to significantly less errors and shorter gestures for unpracticed commands. Taken together, the quantitative results show that shape writing is among the fastest mobile interfaces for text entry and control, both initially and after practice, that are currently known.



# Acknowledgements

---

Although my name stands alone on this dissertation, it would never have been completed without the support of many other people.

In particular, I would like to express my gratitude to Dr. Shumin Zhai for being an outstanding advisor, collaborator and friend. I could write pages on why Shumin is an excellent thesis advisor. In fact I did exactly that in 2005 for a nomination of best technical co-op advisor at IBM Almaden Research Center (he won). Shumin not only taught me the principles of human-computer interaction research, he taught me the importance of focus, logical thinking and setting priorities straight. I can see now that I matured tremendously after these years of research. I am at this point in time Shumin's only graduate student and as a result I have benefited from essentially an unlimited stream of advice, ideas and inspiration from him, both when I was working across the hallway from his office at IBM Almaden and at Linköping University across half of the globe. The opportunity to become a human-computer interaction scientist through his expert guidance is a once in a lifetime opportunity and something I will be eternally grateful for.

I also want thank Prof. Sture Hägglund for his enthusiasm, advice and his generous funding of my doctoral research tenure at Linköping University. Without Sture's support this dissertation would not have been a reality. I also want to thank my research group's leader Prof. Henrik Eriksson for his invaluable advice. I also really appreciate Prof. Erik Sandewall at the artificial intelligence and integrated computer systems division, who generously offered financial support for the first semester before I was admitted into the graduate program in computer science. Erik was one of the first who came to my "office" (bunker!) after I finished my undergraduate thesis (*D-uppsats*) and explicitly congratulated me on having written an excellent thesis. I am very thankful for his truly unselfish initiative to support my research at that initial stage when the technology was yet unproven.

I'd like to thank, in advance, Prof. Bill Buxton to agree to serve as my thesis opponent, and Prof. Lars Ahrenberg, Prof. Jan Gulliksen and Docent Mikael Goldstein to serve as my thesis examination committee members. Thank you for not only taking your valuable time to read and examine my thesis but also travel all the way to Linköping from various parts of the world.

At Linköping University I particularly want to thank Prof. Lars Ahrenberg, Prof. Nils Dahlbäck, Maria Holmqvist, Magnus Ingmarsson, Magnus Rimbark and Dr. Pernilla Qvarfordt who offered advice that directly helped my research. In addition I thank Imad-Eldin Ali Abugessaisa, Lise-Lott Andersson, Dr. Mattias Arvola, Dr. Erik Berglund, Prof. Patrick Doherty, Arne Fäldt, Fredrik Heintz, Dr. Stefan Holmlid, Dr. Björn Johansson, Prof. Arne Jönsson, Britt-Inger Karlsson, Stefan Karlsson, Anders Larsson, Ola Leifler, Dr. Hanjing Li, Dr. Jonas Lundberg, Jalal Maleki, Docent Magnus Merkel, Rolf Nilsson, Piotr Rudol, Sonia Sangari, Dr. Annika Silvervarg, Mustapha Skhiri, Sara Stymne, Per Sökjer, Eva Ragnemalm, Jiri Trnka, Lillemor Wallgren, Mariusz Wzorek and the faculty, technical and administrative staff, and my fellow graduate students for providing a vibrant work environment.

During my four years of research I was offered the fantastic opportunity to work for almost two years at IBM Almaden Research Center in sunny San Jose, California, USA. I had a wonderful time in California and my graduate internship at IBM Research gave me the opportunity to interact with many world class scientists. The following researchers there directly affected my research direction and generously shared their experiences and offered invaluable advice: Dr. Johnny Accot, Dr. Arnon Amir, Dr. Tue Haste Andersen, Dr. Sreeram Balakrishnan, Dr. John Barton, Dr. Allen Cypher, Dr. Ronald Fagin, Jon Graham, John Karidis, Min Lin, Dr. Paul Maglio, Dr. Robert Morris, Dr. Wayne Niblack, Dr. John Pitrelli, Dr. Barton Smith, Dr. Jayashree Subrahmonia and Alison Sue. I also want to thank Dr. David Beymer, Dr. Christopher Campbell, Dr. Steve Cousins, Dr. Alex Cozzi, Dr. Andreas Dieberger, Clemens Drews, Stephen Farrell, Dr. Myron Flickner, Dr. Eben Haber, Dr. Beverly Harrison, Dr. Eser Kandogan, Dr. Stina Nylander, Dr. Daniel Russel, Dr. John Tang, Vladimir Zbarsky and everyone else I have interacted with at the research center.

California and especially San Francisco is an amazing place and I got the opportunity to hang out with some really cool people such as Kristoffer Hallgren, Dr. Tue Haste Andersen, Dr. Jhilmil Jain, Sophia Liu, Eric Perlman and Lubomira Stoilova, and all you others.

I also want to thank the generous support of the ACM SIGCHI community particularly by way of admitting and sponsoring me to the doctoral symposia at UIST 2004 and CHI 2005. At UIST 2004 I was given invaluable advice from a panel consisting of Dr. Ken Hinckley, Prof. Scott Hudson and Prof. Beth Mynatt, and the other graduate students that participated. At CHI 2005 the panel consisted of Prof. Gilbert Cockton, Prof. Joëlle Coutaz, Dr. Mary Czerwinski Prof. Jan Gulliksen, Dr. Henry Lieberman, Prof. Philippe Palanque, Prof. Fabio Paternò, Prof. Raquel Prates and Prof. Janet Wesson. Needless to say, the panel members and participating graduate students were invaluable to my continued research. I was very fortunate to receive a “Best doctoral consortium contribution” award at CHI 2005 that certainly boosted my confidence when pursuing my research direction.

At the CHI, IUI and UIST conferences I have attended I have had the pleasure to interact with many insightful researchers. In particular I would like to thank Prof. James Fogarty, Dr. Antti Oulasvirta, Jingtao Wang and Prof. Kari-Jouko Rähkä (who also was my opponent for the licentiate thesis).

The Gesture & Dynamics workshop at Glasgow University in 2006 was very inspiring. I am grateful to Prof. Roderick Murray-Smith who arranged the workshop, invited me to attend and managed to fund my expenses. The talks at the workshop were excellent and the conversations afterwards even better. I thank Prof. Poika Isokoski, Prof. David MacKay, Prof. Benoît Martin, Prof. Roderick Murray-Smith, Morten Proschowsky and Dr. John Williamson for stimulating conversations.

I also want to take the opportunity to thank some of my fellow text entry researchers: Prof. Scott MacKenzie, William Soukoreff, Prof. Poika Isokoski, Prof. Brad Myers and Prof. Jacob Wobbrock. Your research papers have been inspiring and I hope you feel the same of my own and my co-authors work. Unfortunately I have only had the opportunity to interact briefly with some of you as of yet.

I want to thank all participants that volunteered to participate in my experiments. Aside from providing me with invaluable experimental data, many of you generously provided insightful comments that affected my research direction.

I thank my parents Eva and Sven Kristensson and my brother Lars Johan Kristensson, my family and friends in Malmö and everywhere else, for your support, understanding and patience with me when I have insisted on traveling back and forth between the USA and Sweden for the last four years. I also thank Julia Rolf for being there.

Last, I want to express my gratitude to IBM Almaden Research Center and the Swedish Institute of Computer Science in Linköping for sponsoring parts of my research.





# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Discrete Shape Writing</b>	<b>7</b>
<b>3 Continuous Shape Writing</b>	<b>25</b>
<b>4 Recognizing Continuous Shape Writing</b>	<b>83</b>
<b>5 Continuous Shape Writing for Control</b>	<b>99</b>
<b>6 Design Dimensions of Mobile Text Entry</b>	<b>127</b>
<b>7 Conclusions</b>	<b>181</b>
<b>References</b>	<b>185</b>



# Chapter 1

## Introduction

---

The main contribution in this dissertation is the introduction, development, and analysis of shape writing as a text entry and control interface.

Why do we need another text and control entry interface? The short answer is that non-disabled users do not need a new text interface for desktop computing. The desktop keyboard works just fine. However, when the form factor of computing devices is too small to host a desktop keyboard, or when the desktop keyboard and mouse setup does not apply (e.g. wall-sized displays), new effective interaction interfaces are required.

Why do we need to write text on mobile devices? First, communication with family, friends, colleagues and others have evolved beyond speech and transcended into email and instant messaging communication. Second, one of the fundamental culture-preserving activities our modern civilization relies on is the art of writing. Therefore effective text entry inventions play an important historical role. Orr [1987] captures the importance of fast text entry via shorthand in the following passage: “Thanks to stenography, no gap would exist, nothing would be lost. At last history would be both significant and faithful, loyal to its objects”.

In the last centuries writing technology has undergone many changes throughout history. In the 19<sup>th</sup> century typewriting technology rapidly evolved to be practical and replaced pens and quills (to a great extent). In the 20<sup>th</sup> century the typewriter was replaced by the desktop computer and the keyboard. In the late 20<sup>th</sup> century and in the beginning of the 21<sup>st</sup> century the mobile phone evolution created demand for efficient mobile text entry methods. As the mobile phone becomes ubiquitous in society, users expect the power of the desktop in a handheld device. Mobile phones gain increasing computational power and storage capabilities and there are already mobile phones that can show movies, act as a digital music player and offer full-scale web browsing. The bottleneck for information flow is however limited by the inefficient communication channel between the user and the small device. The small mobile phone form factor has proven to be surprisingly difficult to overcome and limited text entry capabilities are in effect crippling mobile devices’ use experience. The systems in this dissertation are intended to offer effective alternatives.

## 1.1 Background

---

This dissertation is in the field of human-computer interaction. However, human-computer interaction is a broad field ranging from ethnography and interaction design to input device engineering and human performance modeling. My specific dissertation topic is within the subfield of input devices and text entry.

Text entry research has a long and fascinating history. In fact, text entry research is much older than computer science, and certainly older than human-computer interaction. The first real text entry innovation was the alphabet and “longhand” writing. The letters in the alphabet have evolved tremendously and become increasingly efficient over time [Sacks, 2003]. The second innovation was founded on the realization that the alphabet was inefficient for rapid transcription and that some form of shorthand writing system was needed. The earliest such system was in use already 600-500 B.C. in ancient Greece. The Romans took over the tradition of shorthand writing and the most famous early shorthand system is probably *notæ Tironianæ* (the “Tironian notes”) invented by Marcus Tullius Tiro. Tiro’s shorthand system was much more effective than the Roman equivalent of long hand writing. Tiro’s system became widely used by secretaries in Rome and the number of shorthand “notes” grew to around 13,000 [Melin, 1927]. The writing tablet was covered with wax and a wooden pen, and the stylus was used to carve symbols into the wax layer. The opposite end of the stylus was used as an eraser [Melin, 1927].

Amazingly, Tiro’s notes were less efficient than our modern alphabet. Like all other shorthand systems Tiro’s notes also needed to be transcribed to proper script in order to be readable afterwards. However this transcription was difficult and tedious. The Byzantine emperor Justinian I even went so far as to ban shorthand usage because of the high risk of incorrectly transcribed notes [Martinville, 1849]. Here is an early example of the importance of not only consider entry rate – raw speed – in text writing systems. The error rate is of equal importance.

Shorthand systems continued to evolve. The first true stenography system created in Europe was probably *Nova Ars Notaria* (“the new note art”), invented at the end of the 14<sup>th</sup> century, possibly by the monk John of Tilbury [Melin, 1927]. It had three new inventions. First, the individual letters in the alphabet were dramatically simplified to simple line marks [Melin, 1927]. In fact, almost the entire alphabet in *Nova Ars Notaria* is identical to the Unistrokes alphabet proposed by Goldberg and Richardson [1993] for handheld computers. Second, word stems were encoded as line marks for the first letter and complemented with dots and lines to distinguish different word stems from each other. In essence, this is an early example of text compression. Third, frequency analysis was used to find the most common word stems from the psalms in the Book of Psalms. These word stems were given unique encodings, while the rest were written as semi-shorthand where the consonants were usually written as traditional Latin letters [Melin, 1927]. Apparently three corner stones of text entry

research: 1) minimization of users' articulations, 2) language modeling, and 3) optimization based on frequency analysis, were already explored in 13<sup>th</sup> century medieval Europe.

Another breakthrough in text entry was the realization that words can be encoded based on sounds rather than how they are written in longhand. John Willis published a shorthand writing system in 1602 where letters that are silent and repeated letters are omitted, and only the dominating sound in a diphthong is written. The letters in the alphabet are also simplified into as simple line marks as possible to improve efficiency. Figure 1.1 shows how to write the word *are* in Willis's system [Melin, 1927]. Rules defined when vowels could be omitted or written as dots along an imaginary vertical axis orthogonal to the base line. Abbreviation rules were also employed. Furthermore, Willis attempted to define the shorthand symbols such that shorthand symbols that were frequently written in succession could be easily connected [Melin, 1927] (as in Figure 1.1).



Figure 1.1. An example of Willis's shorthand. Left: The stroke for *a*. Center: The stroke for *r*. Right: The stroke for *are*.

The idea that words can be encoded based on sounds were adopted in the internationally best known shorthand systems, Pitman and Gregg shorthand, introduced in 1837 and 1888 respectively [Melin, 1927].

However, in the 19<sup>th</sup> century the typewriter was invented [Yamada, 1980]. Slowly the typewriter, and later the desktop computer became the dominating text entry method in practically all text composition tasks. With a desktop keyboard there is no need for transcription, and using a computer the text can be efficiently edited with a word processing application.

In the late 20<sup>th</sup> century and the beginning of the 21<sup>st</sup> century mobile phones became increasingly popular and transcended into all-purpose handheld computing devices. The desktop keyboard is too large for mobile phones and the keypad too limited. In recent years advanced mobile phones have come equipped with touch-screens which enable new text entry solutions. This dissertation explores how software keyboards on touch-screens can be improved to provide an efficient and practical text entry experience on mobile devices.

## 1.2 Central Hypothesis and Research Questions

---

The work in this dissertation is primarily based on three intertwined concepts: the software keyboard, language redundancy and pattern recognition.

A software keyboard is a keyboard displayed to the user on a touch-sensitive screen. Software keyboards are usually operated with a finger or a pen. Software keyboards are common in mobile devices and ubiquitous in pen-based operating systems.

Language redundancy is a feature of natural languages that is necessary for practical communication. Without some redundancy human-human communication would be almost impossible due to noise, ambiguities and other factors in verbal and non-verbal communication. A side-effect of language redundancy is that not all letter key combinations on a software keyboard form valid words in a language. There is an infinite number of letter key combinations, but only a finite set of words in any human language. Valid words can be captured in language model. A simple form of a language model is a lexicon.

Pattern recognition is the process of recognizing regularities. In a software keyboard the user's pen or finger contact positions form high resolution spatial patterns. Words captured in a lexicon can also be represented as high resolution spatial patterns by mapping the letters in the words to the letter key positions on the software keyboard layout. Using pattern recognition the user's motion pattern can be matched against a set of such word patterns.

The central hypothesis of this dissertation is that it is possible to combine these three elements: software keyboard, language redundancy and pattern recognition, and create new effective interfaces for text entry and control. I call these collectively "shape writing" interfaces.

Shape writing can be broken down into two fundamentally different input methods. With the first method the user writes a word by serially tapping the letter keys of the software keyboard with a pen or a finger. Due to the partitioning of the user's articulation for a word into discrete steps, I call this input method "discrete shape writing". With the second method the user articulates a trace that connects the intended letter keys without lifting the pen or finger. The user simply slides a finger or pen along a path resembling the geometric trace that connects all the letter keys of the intended word in sequence. The user's articulation for a word is no longer partitioned by lifting up and pressing down a pen or finger. Rather the user's entire continuous high-resolution articulation of the shape of a word is considered. I call this latter method "continuous shape writing".

If it is possible to devise discrete and continuous shape writing systems, a set of research questions emerge:

1. How are effective discrete and continuous shape writing systems engineered – from the user interface to the recognition algorithms?
2. How effective are the discrete and continuous shape writing interfaces for text entry?
3. Can shape writing also be used as a control interface, and if so, how effective is it?

Research question 1 is answered in Chapters 2-4. Research question 2 is answered in Chapter 3 and Chapter 6. Research question 3 is answered in Chapter 5.

In addition to answering these specific research questions it is important to frame the work in relation to previous research. Chapter 6 proposes a set of the 22 most important design dimensions of mobile text entry. In the process, Chapter 6 relates the text entry methods proposed in this dissertation to the large body of prior but most recent research on text entry.

## 1.3 Definitions and Accuracy of Measurements

---

### 1.3.1 Entry and Error Rate Definitions

The text entry research community has not come to a consensus regarding entry and error rate measurements. In this dissertation entry rate and error rates will, unless otherwise noted, be defined as follows.

Entry rate is measured in words per minute (wpm). A word is defined as five consecutive characters. A character is defined as any valid letter, number, punctuation, white space or control code that a user would want to enter to the computer system. For example, the letter *a*, the number 2, the punctuation symbol ; , the TAB and ENTER key presses are all characters. If entry rate is known in cps (characters per second) the entry rate in wpm is:

$$\left(\frac{60}{5}\right)cps = 12cps \quad (1.1)$$

Error rate is defined as the minimum number of Morgan [Morgan, 1970] editing operations required to transform the user's written text into the user's intended text, divided by the number of characters in the user's intended text. Error rate is measured in percent. If the written text is identical to the intended text the error rate is 0%. If the written text is completely different from the intended text the error rate is 100%.

### 1.3.2 Accuracy of Measurements

The timing measurements from the experiments reported in this dissertation have been recorded with the help of operating system functions that uses the most accurate hardware timer available, and guarantees nanosecond precision and microsecond

accuracy. Individual timing measurements are reported in the unit appropriate to the task, typically milliseconds.

## 1.4 Outline

---

The chapters are outlined as follows. Chapter 2 introduces discrete shape writing – the simplest form of shape writing. Chapter 3 introduces the more advanced and promising form of shape writing that is called continuous shape writing. Chapter 4 shows how to recognize continuous shape writing. Chapter 5 shows how continuous shape writing can be used to control operating system and application functions. Chapter 6 surveys related work in the mobile text entry field and identifies the design dimensions of mobile text entry. It ends with a comparison of a selection of exemplary mobile text entry methods from the perspective of these design dimensions. Chapter 7 presents conclusions, reflections and future work.

This dissertation is intended to be read as a book – from start to finish. All chapters except Chapter 4 assume the reader is familiar with human-computer interaction. Chapter 4 assumes the reader is familiar with the basic pattern recognition literature. The contents in Chapter 4 are not necessary to understand the contents in the proceeding chapters. Chapter 6 can be understood without reading any of the preceding chapters, even though reading Chapter 2 and 3 is helpful.



# Chapter 2

## Discrete Shape Writing<sup>1</sup>

---

This chapter introduces the concept of discrete shape writing. The systems presented in this chapter are intended to complement software keyboards where the user inputs words by tapping on letter keys with a finger or a pen.

### 2.1 Introduction

---

A software keyboard is a touch-sensitive keyboard-display where the user selects letter keys by pressing them with for instance a stylus or a finger. In contrast to a physical desktop keyboard a software keyboard is typically operated with a single contact point, e.g. a finger or a pen. Another difference is the lack of tactile sensation of the key boundaries and the feedback of a key press.

One weakness of existing software keyboards is the verbatim process – the user has to tap letter by letter with complete accuracy. It is well known that natural languages have a great deal of regularity and redundancy [Shannon, 1948]. From an information theory point of view, tapping all letters with 100% accuracy is over-specifying the amount of information needed.

Goodman, Venolia, Steury and Parker [2002] proposes a method for software keyboard error correction that takes advantage of language regularities. Inspired by speech recognition technology, they calculate the probability of the intended key based on a character-level language model (letter sequence statistics) and a stylus tapping model derived from observations of users’ landing positions on the keys. Goodman et al. [2002] presents experimental results that indicate that their model reduced participants’ error rates compared to traditional uncorrected stylus typing [Goodman et al., 2002]. However it is not clear from their study to what degree the “tapping model” contributed to the error correction.

### 2.2 Relaxing Fitts’ Law

---

This chapter proposes an alternative solution. Fitts’ law [Fitts, 1954; MacKenzie, 1991] can be used to model the average time  $T$  required successfully hit a key of size  $W$  over distance  $D$  on a software keyboard (e.g. [Getschow, Rosen and Goodenough-Trepagnier, 1986]):

$$T = a + bID \tag{2.1}$$

---

<sup>1</sup> This chapter is a revised version of Kristensson and Zhai [2005].

$$ID = \log_2 \left( \frac{D+W}{W} \right) \quad (2.2)$$

where  $a$  and  $b$  are regression coefficients. For stylus typing on software keyboards these parameters have been estimated to  $a = 83$  and  $b = 127$  ms [Zhai, Sue and Accot, 2002].

This means that relative tapping accuracy imposes a certain speed ceiling. If the user attempts to go beyond the ceiling, the landing points of the stylus will tend to fall outside of a targeted key, resulting in a letter different from the intended one. In other words, the user will tend to break the  $W$  constraint. This adds to the user's frustration since it takes additional time and effort to correct these errors. Accuracy constraints are particularly problematic for users with certain motor control disabilities and for expert users who push their text entry speed limit. In the case of small mobile devices, the accuracy problem will be more acute.

The goal is therefore to relax the accuracy requirement of precisely tapping on each letter, effectively widening the constraints of  $W$ . This is possible based on two observations:

First, users' vocabulary constrains the possible letter key combinations and this redundancy in the interface is exploited by a language model. A simple but effective language model is a lexicon, i.e. a list of words.

Second, the landing point of the stylus on a software keyboard is a quantized high-resolution variable recorded by the tablet or the touch screen, in contrast to a physical desktop keyboard that can only record isolated key positions. A series of stylus landing points implicitly form a high resolution sequential point pattern on the software keyboard. The center positions of all letter keys needed for inputting a word also form a pattern on the software keyboard. The distance between these two patterns is captured mathematically by a similarity function. The system finds the user's intended word by computing the similarity function between the user's typing sequence and the ideal center hit points of all words in the lexicon. If no close word is found, the user's inputted letter keys are left unchanged.

## 2.3 Linear Discrete Shape Writing

---

The first correction system investigated uses a straight-forward matching method that is linear in relation to the number of stylus landing hit points. It compares the user's stylus hit points with the center points of the letter keys for all words in a lexicon.

### 2.3.1 Example

In Figure 2.1 the user has tapped on the keys  $r, j$  and  $w$  on a QWERTY layout. Without any error correction  $rjw$  would be returned as the typed word. However, when looking at the hit points and comparing them with the center points corresponding to the letter

keys of all words in the lexicon the word *the* is found as a close match. Therefore the incorrect word *rjw* can be automatically corrected into *the* despite the fact that the user missed all the targeted keys.

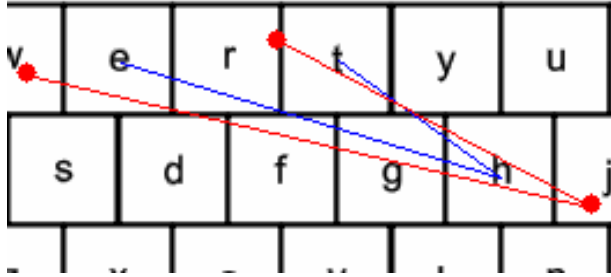


Figure 2.1. An example of linear discrete shape writing. The connected vertices indicate a user's stylus hit points over the letters *r*, *j* and *w* in sequence. The best geometrically mapped word *the* is has been found in a lexicon with 57,392 U.S. English words. The word is shown as connected lines anchored at the center locations of the letter keys *t*, *h* and *e*.

### 2.3.2 Similarity Function

A simple template-driven approach is used that matches the geometrical traces of the user's input and a word in the lexicon directly using an algorithm that has the following properties:

1. Scalable to a lexicon that practically includes all words needed by a user.
2. No training is required for the recognition algorithm.

Next, let  $X$  be an *unknown* pattern of  $n$  stylus hit points  $\{x_i\}$ , and let  $Y$  be a template pattern of  $n$  centers  $\{y_i\}$  of keys in the keyboard corresponding to letters comprising a word  $w$  in the lexicon,  $w \in \{w : w \text{ is a word with } n \text{ letters}\}$ . The average spatial similarity between the patterns can then be computed with the following algorithm:

$$D(X, Y) = \frac{1}{n} \sum_{i=1}^n \|x_i - y_i\|. \quad (2.3)$$

Equation 2.3 is perhaps most easily interpreted as a generalization of the Hamming distance [Hamming, 1950]. The Hamming distance measures how many characters are different in two strings of equal length. If two strings are identical, the Hamming distance is zero. Instead of measuring whether the characters at the  $i$ th position are different, the similarity distance function in Equation 2.3 measures the average spatial

(Euclidean) distance between two corresponding points at the  $i$ th position in the patterns.

To avoid matching unlikely words a threshold constant  $T$  is imposed on each point-to-point distance. If  $D(X, Y) > T$ , the distance to the word is set to  $\infty$ . Among all candidate words a subset is created that consists of the words with a  $D$  value below the threshold  $T$ . These words are returned to the system as a ranked list. The system output is the word with the smallest  $D$  value.

The above matching method is simple and conservative. Specifically, if the user taps on all the correct keys of a word, no other word can be closer. Also, the above scheme is easy to implement and since there are few point comparisons, exhaustive (linear) search through the lexicon is fast.

### 2.3.3 Delimiter

The system requires the user to delimit each word. Among other possible solutions such as a special-purpose physical button on the user's non-dominant hand, the linear correction system currently uses a set of delimiting characters. These characters are word delimiters in normal word processing, e.g. the tab-character, the space-character, semi-colon, etc.

## 2.4 Experiment 2.1: Accelerated Performance

---

Two experiments were conducted to assess the effectiveness of the linear correction system. The main concern was if the simple linear correction algorithm was enough to correct the vast majority of the errors users make in stylus typing. A second concern was to determine if the input speed was increased as a result of relaxing the Fitts' law  $W$  constraint.

### 2.4.1 Method

#### 2.4.1.1 Design

The participants tapped on a software keyboard with a stylus. Two conditions were investigated:

1. The software keyboard did not perform any automatic correction of the participants' writing. This condition served as the baseline.
2. The software keyboard automatically corrected participants' writing when the SPACEBAR key was tapped.

All participants typed using both conditions in this within-subject experiment. The order of the two conditions was balanced.

#### 2.4.1.2 *Participants*

14 paid participants were recruited from IBM Almaden Research Center. 12 were male and two were female.

#### 2.4.1.3 *Apparatus*

Participants tapped a stylus on a software keyboard shown on a touch-sensitive screen. The screen resolution was set to  $1024 \times 768$  pixels. The software keyboard dimensions were  $460 \times 220$  pixels. The software keyboard used the standard QWERTY keyboard layout. Despite QWERTY being suboptimal for stylus typing [Getschow, Rosen and Goodenough-Trepagnier, 1986], the QWERTY layout was chosen because 1) it is the de-facto software keyboard layout, and 2) participants are familiar with the layout on the QWERTY keyboard, thus the amount of practice required from the participants is minimized for the experiment.

#### 2.4.1.4 *Material*

The pangram “The quick brown fox jumps over the lazy dog” was used as the test sentence. This sentence is quite difficult to recognize correctly on the QWERTY layout since the word “dog” is close to “dig”, and “fox” is close to “fix”. The lexicon used by the recognizer consisted of 57,392 words.

#### 2.4.1.5 *Procedure*

Two tasks were used:

1. In the first task participants repeatedly wrote an individual word from the stimuli pangram. Participants were instructed to write as fast and accurately as possible. The interface did not allow participants to correct their mistakes.
2. In the second task participants repeatedly wrote the entire stimuli pangram. To proceed, the pangram had to be completely and correctly copied for the experiment. Participants had to type correctly, or correct their errors by re-positioning the text caret and use the BACKSPACE key on the software keyboard.

Task 1 was repeated ten times and task 2 was repeated 12 times.

## 2.4.2 **Results**

#### 2.4.2.1 *Entry Rate*

The average entry rates of the last three sentences were analyzed for any difference between the baseline and the discrete shape writing software keyboard (Keyboard Type). Repeated measures variance analysis showed that the only significant factor was Keyboard Type  $\times$  Order interaction ( $F_{1, 12} = 22.8, p < .001$ ). Neither Order nor Keyboard Type alone was a significant factor in speed. This means that there was an asymmetrical skill transfer between the two types. As Poulton [1966] argued, when there is an asymmetrical skill transfer, the only way to find the unbiased result is to

restrict analysis to the data from the first condition presented – effectively turning the experiment to a between subject design, although the power of the experiment is much weakened. With such an approach, the difference between the two keyboard types was still insignificant, although the average speed of the discrete shape writing keyboard condition (29.5 wpm) was 24% higher than the baseline condition (23.7).

#### 2.4.2.2 *Error*

Errors reported here are corrected errors. Recall that participants were not allowed to proceed until the entire sentence was written correctly.

On average the participants made 8.7 errors in the verbatim condition and 5.3 in the relaxed condition. The difference was not statistically significant.

## 2.5 **Experiment 2.2: Saturated Learning**

---

Experiment 2.1 revealed no clear advantage of using discrete shape writing. This could be due to a number of reasons. One reason could be that participants' learning of the text input method never saturated. To rule out this possibility a second experiment was set up that was explicitly designed to quickly saturate learning. Participants were asked to repeatedly write a single word five times in sequence, ten times. The task is artificial in the sense that the obtained results cannot be generalized to regular software keyboard typing on open text. However using the artificial experimental task it is possible to gain an understanding on if there is any speed performance difference easily achievable by users at all.

### 2.5.1 **Method**

#### 2.5.1.1 *Design*

Because an asymmetrical skill transfer effect was found in Experiment 2.1 this experiment was a between subjects experiment where participants were only exposed to a single condition.

#### 2.5.1.2 *Participants*

26 unpaid volunteers from the Linköping University campus and IBM Almaden Research Center were recruited. 18 were male and 8 were female. Some of the participants had participated in Experiment 2.1. These were balanced among the two conditions.

#### 2.5.1.3 *Apparatus*

Participants tapped a stylus on a software keyboard shown on either a touch-sensitive screen, or a CRT screen connected to an external Wacom tablet. The screen resolution was set to  $1024 \times 768$  pixels on the touch-screen and  $1600 \times 1400$  pixels on the CRT. The software keyboard used the standard QWERTY keyboard layout. The software keyboard dimensions were  $460 \times 220$  pixels.

#### 2.5.1.4 *Material and Procedure*

Each participant entered the word “computation” 50 times in ten groups. Participants had to correct errors remaining in each group before proceeding.

#### 2.5.2 **Pilot Performance**

A participant with a total of a few hours of experience (both conditions) participated in a pilot test. The participant typed the word “computation” 50 times (in ten groups) in both conditions as a pilot within-subject experiment. The user’s total number of errors was 18 with the baseline condition and 4 with the discrete shape writing condition, suggesting that the discrete shape writing condition had a positive effect. Counting the speed of the last 20 correctly typed words (the early words served as a buffer for the learning effect in this within subject test) the average was 50.1 wpm for the relaxed condition and 35.5 wpm for the baseline. The difference was statistically significant ( $F_{1,38} = 53$ ,  $p < .0001$ , within the subject). This suggests that advanced users could potentially gain a significant speed performance advantage using the discrete shape writing technique.

#### 2.5.3 **Results**

##### 2.5.3.1 *Entry Rate*

The participants’ grand mean entry rate was 23.2 wpm in the linear discrete shape writing condition and 22.6 wpm in the baseline condition. Analysis of variance showed that the difference was not statistically significant ( $F_{1, 25} = .038$ ,  $p = .846$ ). There were large individual differences in entry rate. It appears that some users could take advantage of the relaxation and exploit it more than others.

##### 2.5.3.2 *Error*

The total numbers of errors that had to be corrected by the users in all ten sentences were 38 in the baseline condition and 22 in the linear discrete shape writing condition. The latter number does not include errors automatically corrected into the correct word by the linear correction system. For the linear discrete shape writing condition, an analysis was performed of the corrections on the individual words for the last three sentences of each subject. A total of 46 errors were found (including errors automatically corrected into the correct word). The number and percent of errors in each error category is shown in Table 2.1.

Figure 2.1. The number and percent of errors in each error category.

<b>Error Category</b>	<b>Errors</b>	<b>%</b>
Errors automatically corrected	24	52%
Errors corrected by the user before matcher	17	37%
Deletions causing erroneous corrections	2	4.5%
Missed SPACEBAR key	2	4.5%
Other	1	2%

There were 24 instances where an error was auto-corrected to the correct word. Hence the error correction method captured most mistakes. The majority of the remaining errors were corrected by the user before auto-correction was applied. Among all the cases where automatic error correction took place the majority was successful: 24 instances were corrected correctly, which amounts to a success rate of 83%. In two cases deletions (the user omitted a character) caused erroneous corrections (the letter key sequences *computation* and *computation* caused incorrect replacements into the word *completion*). In two other cases the SPACEBAR key was missed, causing concatenation of two words into one so the user had to move the text caret between the words and insert a space character afterwards. These two categories of errors were very detrimental to users' perception of and performance of the linear discrete shape writing interface. It is plausible users perceived they could not really trust the system since it may give them implausible results when they (unconsciously) missed a key, particularly when the missed key was the SPACEBAR, which acted as the delimiter. It is likely this deficiency of the system caused users to take a more conservative approach and push less for speed.

## 2.6 Elastic Discrete Shape Writing

---

An 83% successful error correction rate is too low to be practical. Furthermore, two critical system deficiencies in the simple linear correction system were revealed after Experiment 2.1 and 2.2:

1. The choice of delimiter is crucial. The SPACEBAR key is a bad delimiter since it is the most likely key being hit. Erroneous delimitations must be avoided since they will result in incorrect replacements by the system or no correction at all if the user fails to delimit two long words.
2. The algorithm is too simplistic in that it can only handle patterns of the same length. Experiments 2.1 and 2.2 revealed that users made deletion errors (omission of an intended letter key) that the algorithm could not correct. Furthermore, users requested that the algorithm should handle insertion and transposition errors too. In general, the success rate of the algorithm must be very high, since erroneous corrections results in a higher cost for the user than if the algorithm did nothing at all.

Based on an analysis of the empirical data gathered in Experiment 2.1 and 2.2, improving delimitation and handling patterns of different lengths would directly lead to a 97% error correction success rate. Clearly a generalization of the linear similarity function in Equation 2.3 is required that is more flexible in matching users' stylus hit points against the word patterns.



As an example of the new algorithm's capabilities consider Figure 2.2. In Figure 2.2 the user has tapped on the keys  $r$ ,  $j$ ,  $n$  and  $w$  (hit points indicated as solid circles) on a zoomed-in part of the QWERTY layout. Without any error correction  $rjnw$  would be returned as the typed word. With the linear matcher no correction at all or an erroneous correction would be returned. The next system presented is capable of handle this situation and correct the input into the intended word  $the$ , despite the fact that the user missed all the targeted keys, and in addition had one extra spurious stylus tap.

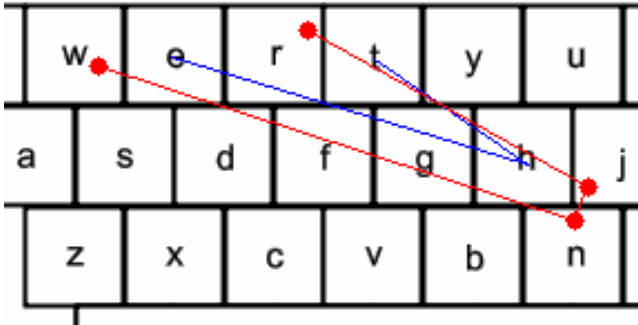


Figure 2.2. An example of error correction: the user tapped on the letter key sequence  $r-j-n-w$  but the intended letter key sequence  $t-h-e$  is returned. The best geometrically mapped word  $the$  has been found in a lexicon with 57,392 U.S. English words. The word is shown as connected lines anchored at the center locations of the letter keys  $t$ ,  $h$  and  $e$ .

### 2.6.1 Similarity Function

Let  $X$  denote an unknown pattern consisting of an ordered sequence of  $n$  stylus hit points  $\{x_i\}$  on a stylus keyboard, and let  $Y$  denote a template pattern consisting of  $m$  points  $\{y_j\}$  that are the centers of the corresponding keys for any word  $w \in \{w : w \text{ is a word in the lexicon}\}$ .

Define  $D(X, Y)$  as the minimum stretching cost needed to transform  $X$  into  $Y$ . Let  $D(X, Y) = K(n, m)$  be the minimum stretching cost of matching  $x_1 \dots x_n$  against  $y_1 \dots y_m$ .  $K(i, j)$  for the subsequences  $x_1 \dots x_i$  against  $y_1 \dots y_j$  can be computed using a recurrent equation of similar form as the traditional edit-distance between two strings over a finite alphabet [Wang and Pavlidis, 2004]. However this form is difficult to use for elastic discrete shape writing since the formulation in [Wang and Pavlidis, 2004] involves a constant penalty in inserting or ignoring a single point. Intuitively the cost of inserting or ignoring a point should depend on the distance between the points matched. Therefore the recurrence equation given by Wang and

Pavlidis [2004] is modified by multiplying the constant penalty  $\tau$  for inserting or ignoring a point with the actual distance between the points compared:

$$K(i, j) = \min \begin{cases} K(i-1, j-1) + \delta(x_i, y_j), \\ K(i-1, j) + \tau \times \delta(x_i, y_j), \\ K(i, j-1) + \tau \times \delta(x_i, y_j) \end{cases} \quad (2.4)$$

where  $K(0,0) = 0$ ,  $\delta(x_i, y_j)$  is the stretching cost  $x_i$  to  $y_j$ , and  $\tau$  is an empirically determined parameter weighting the cost of either ignoring or inserting a single point. The parameter value  $\tau = 2.0$  is currently used. It was discovered by experimentation with the implemented system.

The distance function  $\delta$  measures the spatial distance between the points. To avoid extreme stretching of a single point it is defined as:

$$\delta(x, y) = \begin{cases} \infty & \|x - y\| > r \\ \|x - y\| & \text{otherwise} \end{cases} \quad (2.5)$$

where  $r$  is the maximum distance a point can be stretched. Equation 2.5 is important because it acts as a decision function on whether to automatically correct the stylus tap pattern into an ideal word pattern in the lexicon or not. If the parameter  $r$  is too low automatic correction would never happen.

It is well known that the computation of  $K(n, m)$  for matching  $X$  against  $Y$  in Equation 2.4 can be solved efficiently using dynamic programming in  $O(nm)$  time [Levenshtein, 1965; Wagner and Fischer, 1974].

Equation 2.4 is not normalized with respect to the patterns' lengths and therefore precludes direct comparisons between patterns of varying lengths. Unfortunately normalization of Equation 2.4 is shown to be non-trivial [Marzal and Vidal, 1993]. Fortunately, a pseudo-normalized measure that is sometimes used in dynamic time warping in speech recognition [Rabiner and Juang, 1993] can be directly adapted:

$$D_N(X, Y) = \frac{D(X, Y)}{n + m} \quad (2.6)$$

The best matching word is the word whose pattern has the lowest normalized stretching cost  $D_N$  against the user's tapping pattern.

The algorithm presented can be seen as a generalization of a minimum edit-distance algorithm where the character equivalence function is replaced with the Euclidean distance between two points in a Cartesian coordinate system. In comparison, the

character equivalence function in string matching is defined to return 0 if the characters match and 1 otherwise.

Like the algorithm in Equation 2.3 the elastic discrete shape writing algorithm is template-based. Although conceivably a classic data-driven approach [Duda, Hart and Stork, 2001] could be used, the elastic discrete shape writing method has the following advantages:

1. Simple, elegant and straight-forward to implement and analyze. A generalization of the string edit-distance into a spatial edit-distance is conceptually intuitive for the task.
2. Scalable to a lexicon that practically includes all words needed by a user.
3. Can match sequences of different lengths and handle transposition errors.
4. No training of the classifier is necessary.

The last advantage (4) is crucial in practice. An algorithm based on training data, e.g. a linear machine [Duda and Hart, 1973], would require data from users' repeatedly stylus typing of all the words in the lexicon. Clearly, for a lexicon containing 10,000 – 100,000 words such an approach would be inflexible and cost prohibitive.

### 2.6.2 Indexing

Unlike the linear discrete shape writing algorithm, the algorithm presented in Equation 2.4 has quadratic complexity. To avoid a resource-intensive exhaustive search of a large (in the order of 50,000-100,000 words) lexicon the system uses a novel indexing technique that considerably narrows down the search space.

Straight-forward indexing methods are difficult to use since the modification of the constant penalty  $\tau$  into a non-constant function in Equation 2.4 has the side-effect that the similarity distance function  $D$  violates the triangle inequality and is no longer a metric [Wang and Pavlidis, 2004]. As a result, search methods for metric spaces [Chávez, Navarro, Baeza-Yates and Marroquín, 2001] cannot be used to narrow down the set of likely candidates.

Instead the indexing strategy is based on the observation that since Equation 2.5 constrains corresponding point-to-point distances between the unknown pattern and the template pattern to be shorter than  $r$ , the template patterns whose corresponding first and last point do not meet this constraint can be immediately eliminated without any loss in accuracy. If  $r$  is sufficiently conservative, e.g. 1.5 times the radius of a letter key on the keyboard, this observation directly leads to an effective indexing strategy.

Construct an ordered  $k$ -ary tree data structure of height 2 where  $k$  is the number of letter keys on the keyboard layout. Each node at index  $i$  at depth  $d$  represents a

circular cluster  $C_{id}$  where the  $i$ th key center is the cluster center, and  $r$  is the radius of the cluster. At index  $i$ ,  $1 \leq i \leq k$ ,  $C_{i1}$  represents a *start* position cluster and  $C_{i2}$  represents an *end* position cluster. A pattern  $Y$  of length  $m$  is indexed by a pointer in a cluster  $C_{j2}$  at depth 2 iff  $y_1 \in C_{i1}$ ,  $y_m \in C_{j2}$  and  $C_{j2}$  is a child node of  $C_{i1}$ . Set membership is used to denote that a point is contained in the circular cluster. When querying the index with an unknown pattern  $X$ , the system walks the tree in breadth-first order and collects the set of all patterns in the lexicon indexed at the same depth 2 clusters as  $X$ . This set is then searched exhaustively.

If  $r$  is too large or all words in the lexicon have patterns mapped to the same start and end point clusters, this procedure would still result in an exhaustive search. In practice the character frequencies are distributed unevenly but with enough spread for this indexing procedure to significantly reduce an exhaustive search. For example, the current lexicon in the system contains 57,392 U.S. English words and the largest possible set that needs to be searched exhaustively on a QWERTY layout is about 4,000 words when  $r = 1.5$  in letter key radius units.

### 2.6.3 Threshold

The classifier returns a subset consisting of the words in the lexicon with similarity distance  $D_N$  to the user's tapping pattern below a set threshold  $T$ . These word candidates are returned to the system as a ranked list. The system outputs the word with the shortest  $D_N$  distance.

The threshold  $T$  is fixed in the implemented system and set to  $T = 1.0r_k$ , where  $r_k$  is the radius of a key. In a future system the threshold could be changed from a constant to function dependant on for instance the distribution of point-to-point distances, or the movement dynamics of the user's sampled stylus typing pattern.

### 2.6.4 Lexicon

The lexicon used can be constructed with various methods. It can be a preloaded standard dictionary, or a list of words extracted from the user's previously written documents, including emails and articles, or words added by the user to the list, or a combination of all. Currently the system uses a lexicon containing about 57,392 words.

Indexing makes large lexicons computationally feasible for the recognizer. However, it is important that the lexicon is just large enough (but not larger than necessary) to include all words a particular user needs so the probability of unwanted corrections is minimized and the capacity of correct mapping for "sloppy" stylus typing is maximized.

### 2.6.5 Delimiter

As shown in Experiment 2.1 and 2.2, the SPACEBAR key proved problematic as a delimiter since the risk of accidentally mistyping the SPACEBAR key is just as likely as mistyping any other key. On the QWERTY keyboard layout the SPACEBAR key is enlarged, hence the SPACEBAR key is in fact the most likely key being mistyped. A solution that proved useful is to instead introduce a left-to-right pen-gesture action that is distinct from typing. This solution is similar in spirit to the optional spacebar pen-gesture found in many software keyboard implementations, e.g. [Hashimoto and Togasi, 1995].

## 2.7 Experiment 3: “Expert” Entry Rate – A Pilot Study

Proficiency in a text entry system such as discrete shape writing is a function of practice. The closed-loop action of typing on a regular software keyboard is limited by the human motor control system and can be reliably modeled using a character-level bigram model and Fitts’ law [Fitts, 1954]. Since discrete shape writing relaxes or “breaks” Fitts’ law we cannot, as of today, model discrete shape writing expert performance using any known human performance law. Also note that the recognition precision of discrete shape writing varies with the size and contents of the lexicon.

Instead of performing a theoretical modeling of the performance of discrete shape writing, expert performance was simulated by letting two proficient users repeatedly write selected sentences correctly (errors were not allowed). The test was carried out on a Tablet PC with a discrete shape writing system using the QWERTY keyboard layout and a lexicon containing 57,392 words. The results are shown in Table 2.2. Note that these numbers are “record” entry rates that do not reveal the true average and expert typing performance of discrete shape writing, and should only be considered a demonstration of the potential of the technique in “breaking” Fitts’ law in stylus typing. Also note that the system in the experiment used a very large lexicon as a stress test of the technique and the implemented system. A smaller “optimized” lexicon would increase the probability of desired automatic correction. As a reference, the theoretical average expert typing speed on a QWERTY software keyboard has been estimated to around 34.2 wpm [Zhai, Sue and Accot, 2002].

Table 2.2. Pilot entry rate estimates for the two participants that were “expert” users in elastic discrete shape writing.

Phrase	P1	P2
the quick brown fox jumps over the lazy dog	46.3	37.7
ask not what the country can do for you	45.4	40.1
intelligent user interfaces	51.3	51.8

## 2.8 User Interface

Apart from the distinct left-to-right sliding delimitation pen-gesture to signal end of word, the discrete shape writing interface is in principle indistinguishable from a standard software keyboard. This is an asset in one regard since users do not need to learn a completely new text entry method. Therefore they can directly transfer their stylus typing skills when using the new interface. It is also a liability because although discrete shape writing masquerades as a standard software keyboard, the recognizer component adds an extra layer of complexity to the user. If the recognizer misrecognizes the user's intended action, the incorrect automatic change of the user's input into an unintended word in the lexicon can appear perplexing. To mitigate such issues three new user interface components were considered for discrete shape writing.

### 2.8.1 Tapping Feedback

First, for each key tapped the corresponding character is outputted, just as in a normal software keyboard. To help users get an understanding of the geometrical pattern approach the system can display the hit points of the stylus taps (slowly fading away over time) as a way to "hint" to the user that the proximity information is taken into account. Whether or not this is a good idea remains to be empirically validated. As a general point, "clean" text entry user interfaces that reduce users' cognitive and visual information processing overhead are generally preferred; suggesting that stylus tap visualization should be activated sparsely or perhaps not at all during normal text writing. However, stylus tap visualization can be helpful for novice users as part of an initial practice or during a tutorial.

### 2.8.2 Automatic Correction Feedback

Second, since the system replaces the user's input with something else, it is important to inform novice users about the replacements. This is achieved by drawing the word the system inserted in place of an automatically corrected word with a distinct background color (Figure 2.3).

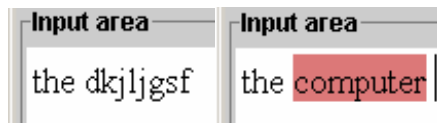


Figure 2.3. Example of correction indication.

### 2.8.3 Error Correction Interface

Third, since the matching algorithm outputs normalized scores it is possible to collect a ranked list of the best matches (list of alternative words). By pressing on a corrected word with the stylus the user brings up a selection widget allowing direct access to the list of alternative words. Among the possible solutions a pie menu was explored (Figure 2.4). The best match is shown at the 270° position in the pie and the best

alternative words follow counter clock-wise. An advantage when using a pie menu instead of a traditional linear pull-down menu is that the user can correct a word by simply selecting the incorrect word and flick the pen in the direction of the desired alternative word [Callahan, Hopkins, Weiser and Shneiderman, 1988].

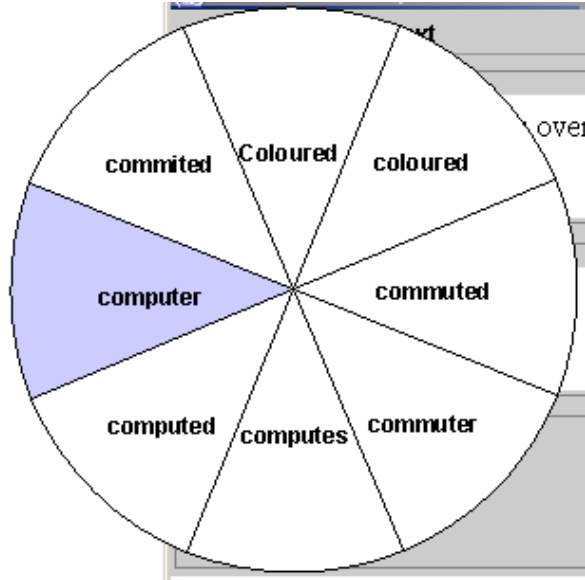


Figure 2.4. A pie menu reveals the word candidates for the automatically corrected high-lighted word in Figure 2.3.

## 2.9 Implementation

Both the linear and elastic discrete shape writing systems were written in Java version 1.4. The elastic discrete shape writing system has been tested on Sun Solaris, Linux and Windows XP Tablet PC Edition. On a 1 GHz Tablet PC the average latency is 40 ms with a lexicon consisting of 57,392 words. The matching process is fast enough that automatic correction is perceived as real-time to the user.

## 2.10 Discussion and Conclusions

Informal testing indicates that one can type faster with less effort with elastic discrete shape writing than on a regular software keyboard. This can be explained by the reduced need to correct frequent errors, and the more relaxed requirement for precision tapping. It is important to choose an appropriate correction threshold and a suitable lexicon, so that neither too many errors are left uncorrected nor too many input strings are changed into unintended words. In general in the experiments it was observed that users were more unforgiving when receiving unintended words than appreciating correct error corrections. It is therefore necessary to be conservative.

This chapter has explored using a geometric pattern recognition approach to relax the precision requirements in stylus typing. The first implementation of this approach, the linear correction system, was tested in two experiments. The data and experiences from these experiments guided the development of the elastic discrete shape writing system. Although the latter system has not been evaluated as formally as the first linear discrete shape writing system, informal testing indicates that the elastic discrete shape writing performs much better. The two key weaknesses observed with the linear discrete shape writing system were solved. First, the delimitation problem caused by missing the SPACEBAR key found in Experiment 21 is eliminated with the use of a pen-gesture as the word delimiter. Second, due to the introduction of elasticity in the matching algorithm, the discrete shape writing system now returns correct results even when a required key tap was missed or an extra tap accidentally added, as long as the overall shape of the input pattern matches the desired target word better than any other alternative. As a result of these improvements, a user can trust the system much more and be more comfortable in taking advantage of the precision relaxation.

The use of a pen-gesture also makes it possible for users to opt out from taking advantage of automatic correction. If the user presses the SPACEBAR key or any other delimitation character, the system's buffer is flushed but no correction takes place. Users are free to choose if and how much the automatic correction system should interfere with their typing.

The discrete shape writing system is essentially enabling users to take advantage of the regularities in the languages in a novel way – relaxing precision constraints by pattern matching, hence “breaking” Fitts' law. Prior techniques have either relied on users consciously “compressing” the input [Shieber and Baker, 2003; Shieber and Nelken, 2007] or using prediction (e.g. Masui [1998]) that demands cognitive and visual reaction time. In contrast, a discrete shape writing interface does not require a user to learn a compression technique or hope that the system predicts the desired input. An expert discrete shape writing user simply taps the word as quickly as possible and relies on the redundancy in the language to make sure that word patterns are sufficiently separated on the software keyboard.

Critical improvements can be made in primarily two domains. First, in Experiment 2.1 it was discovered that users corrected the input before the system in 37% of the cases. An extension of the system is to make it proactive and recognize the user's input before delimitation. Such a feature signals to the user that the system is automatically correcting the input right away. On the other hand, this feedback may be confusing because of the lower quality of data for the algorithm (guessing based on substrings rather than whole words). Second, an improvement is to use higher-level language information (e.g. word  $n$ -gram models) about the text the user is writing to constrain the lexicon dynamically. By dynamically constraining the lexicon the probability increases that the system can perform an accurate automatic correction.



The discrete shape writing interface works with any keyboard layout, either QWERTY or an optimized layout. The discrete shape writing technique can also be transplanted to other text entry interface such as eye-typing. The discrete shape writing interface is a practical and easy-to-implement solution to improve the verbatim and error-prone input method of today's stylus keyboards; requiring little, if any, training from the end-user's part.

### **2.10.1 Data-Driven vs. Template-Driven Automatic Corrections**

In comparison to Goodman et al. [2002] one limitation with discrete shape writing in its current form is that users do not get any feedback on corrections until after they have finished writing the words. In the system presented by Goodman et al. [2002] automatic corrections are made after each single character is pressed.

However, the advantages of discrete shape writing in comparison to a data-driven statistical letter sequence approach such as Goodman et al. [2002] are numerous. The discrete shape writing system's matching effect works on the word level, and the words explicitly belong to an individual user's lexicon. New words can be added and removed in a customized dictionary; different languages such as Swedish, German, Chinese pinyin, etc. can be mixed without affecting the performance or behavior of the system. Depending on the size of the lexicon, the error tolerance of the discrete shape writing interface can be adjusted, either by the user or automatically by the system. Note that if the user aims at the correct letters in a word, the resulting shape will tend to approximate the ideal word pattern and be correctly matched. A user's input pattern can still be successfully matched to the intended word even if some of the hit points are far away from the correct keys, as long as the word patterns in the lexicon are sufficiently separated. Since the system uses the geometrical tapping trace, some amazing corrections can be achieved. It is for example possible to correct the user's input even if the user missed all the intended letter keys (see Figure 2.2 and Figure 2.3), something that is virtually impossible to replicate with another technological approach without taking the spatial hit point information of entire words into account. Furthermore, the intuitive spatial interpretation of the matching method can enable expert users to develop a strategy on how to take advantage of the error correction scheme.

### **2.10.2 Summary**

This chapter presented the design evolution of discrete shape writing. Experiment 2.1 and 2.2 show that there is no clear advantage of linear discrete shape writing in relation to a traditional software keyboard. The much improved elastic discrete shape writing system has as of yet not been empirically validated in a larger controlled study, although based on analysis of the empirical data collected in Experiment 2.1 the improved system would automatically correct 97% of the participants' errors. "Expert" estimates indicate that it is possible that users can reach over 40 wpm in entry rate with the new system.

Qualitatively the primary limiting factor of software keyboard typing is motor inefficiency of repeatedly pressing and releasing a pen in order to articulate a single word. While discrete shape writing might be an improvement, it is not a big leap forward. The next chapter explores an alternative approach where users do not have to lift up and press down the pen for every letter written.

# Chapter 3

## Continuous Shape Writing<sup>1</sup>

---

The previous chapter introduced discrete shape writing, a scheme in which users' geometric discrete tap patterns are matched against a set of words' ideal geometric mapping onto the software keyboard layout. This chapter presents an alternative continuous scheme in which the users perform continuous pen movements on the touch-screen surface.

### 3.1 Introduction

---

To write a word in continuous shape writing, rather than tapping the word serially as in discrete shape writing presented in Chapter 2, the user slides the pen from letter key to letter key in the word (Figure 3.1).

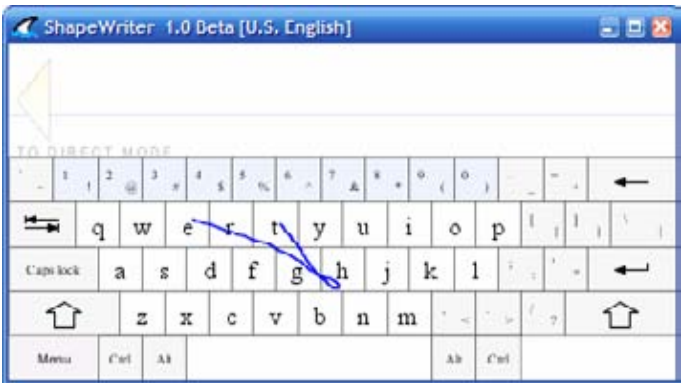


Figure 3.1. An example of continuous shape writing. The user is writing the word *the*. The pen trace begins at the letter key *t* and continues through *h* towards *e*. When the user lifts up the pen the word *the* is recognized as the intended word.

---

<sup>1</sup> This chapter is partly based on work previously published in Kristensson [2002], Zhai and Kristensson [2003], Kristensson and Zhai [2004], Kristensson [2004], Kristensson [2005], Kristensson and Zhai [2005], Zhai and Kristensson [2007], Kristensson and Zhai [2007a] and Kristensson and Zhai [2007b]. However some of my thinking has changed since these prior publications. All text, figures and tables used in this chapter are original work. The discussions and results in the following sections have not been previously published: 3.3 Shape Writing on Mobile Phones; 3.5 Localization; 3.7 Experiment 3.2: Immediate Efficacy; and 3.8 Experiment 3.3: Accelerated Novice Learning.

### 3.1.1 Recognition

The intended word can be recognized by a pattern recognizer, even though many irrelevant letters are crossed. There are two primary reasons why this approach works. First, language redundancy eliminates many improbable letter key combinations. Second, words on a software keyboard form shape representations. Words can be viewed as a sequence of letter key coordinates in a keyboard topology.

#### 3.1.1.1 Language Redundancy

Not all letter key combinations form valid words in a language. There is an infinite number of letter key combinations, but only a finite set of words in any human language. Further, users only write a small subset of the set of total words in the user's language. The subset of words the user is writing is called the active vocabulary. It is typically much smaller than the subset of words the user understands, the passive vocabulary. The valid words can be captured in a lexicon. Typically a subset of words consisting of most words in a typical users' active and passive vocabularies are used. In the continuous shape writing software systems described in this dissertation, lexicons containing over 55,000 words are used.

#### 3.1.1.2 Words Represented as Spatial Traces

Words in a lexicon can be mapped to spatial traces along the letter keys of the word on a software keyboard. In Figure 3.2 the words *the* and *while* are mapped onto a software keyboard layout.



Figure 3.2. The words *the* and *while* mapped onto a QWERTY software keyboard layout.

Internally a pattern recognizer stores a set of word shape representations. When the user is pen-gesturing a word shape the pattern recognizer compares the pen trace against all the internal word shape representations and finds the set of word shapes that most closely matches the user's pen trace. The best matching word in this set is displayed to the user as the matching word. Two effective procedures for performing this recognition are presented in Chapter 4.

### 3.1.2 Movement Efficiency and Chunking

A problem with discrete shape writing and traditional software keyboard typing is movement inefficiency. Users are required to repeatedly move, lift up and press down the pen or finger input device on the software keyboard surface. Anecdotal evidence

from centuries of stenography research has pointed out the impeding effect repeated pen lifts have on performance [Melin, 1927; Melin, 1929].

From a motor control point-of-view, users' discrete pointing actions partition the process of writing a word into several closed-loop planning and execution sub-units that can be modeled by Fitts' law [Fitts, 1954] or first-order lag [Jagacinski and Flach, 2003]. This fact was recognized by Montgomery [1982] when he proposed the touch-sensitive wipe-activated keyboard. In the wipe-activated keyboard some of the common words in U.S. English (for example *the*) are adjacent and thus enables the user to slide the finger over certain letter key combinations rather than serially tapping them.

Even though discrete shape writing relaxes the target width constraint, the Fitts' law model of pointing still imposes a hard limit on performance. In contrast, the continuous pen trajectory movement for continuous shape writing can be "chunked" and learned and executed from motor memory as a single open-loop action. An expert user knows the spatial topology of the keys in the software keyboard and can initiate a motion in the proximity of the keys.

Buxton [1986] argues that gestural interfaces in general should chunk atomic user interface actions into meaningful actions. Novice and expert users alike have a greater, more meaningful and expressive user interface language if primitive actions are abstracted into compound actions with semantic intent. Transforming a pen-tap sequence into a single pen-gesture is a step towards that vision. For example, the letter key taps *t*, *h* and *e* do not have any intrinsic meaning individually. The word *the* on the other hand is a meaningful unit.

In relation to traditional pen-based input techniques such as handwriting, hand printing and specialized pen-gesture alphabets (e.g. Unistrokes [Goldberg and Richardson, 1993]) continuous shape writing is more efficient because pen-gestures result in entire words.

### 3.1.3 Continuous Learning

A central idea behind continuous shape writing is that the initial novice user's visually-guided action gradually transforms into fast open-loop muscle-memory recall.

In the novice phase the user traces words from letter key to letter key. Over time motor memory builds up in the user's brain for the frequently articulated words. Once a word is practiced enough, the user can recall the articulation from memory automatically (see Figure 3.3). An individual user is thus always in a continuum between complete novice and complete expert.

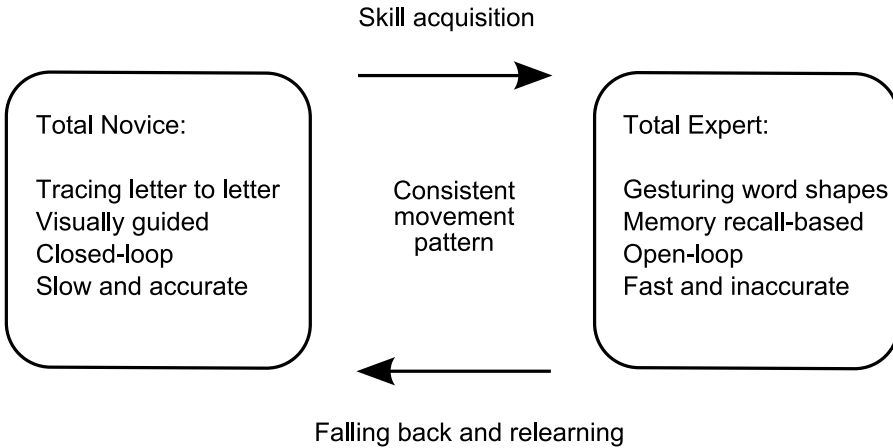


Figure 3.3. The learning loop for an individual word using continuous shape writing.

Fitts [1964] introduces three different stages of skilled performance. The first is a cognitive stage where users consciously learn the skill. At this stage the user devotes a lot of attention towards learning. The second stage is a step towards automation of the skill. Fitts [1964] calls it the associative stage. In this stage the learner tries different strategies to find those that contribute to success or failure. The last stage is the autonomous stage where the skill is performed with little or no conscious thought. Similar theories have been proposed later in different contexts and with different terminologies, for example the knowledge-rule-skill based performance theory [Rasmussen, 1983].

The learning process for continuous shape writing can be viewed through the glasses of the stage theory. In the cognitive stage the user needs to devote considerable attention towards spelling out the intended word and locating the relevant keys on the keyboard. In the associate stage the user tries out the optimal pen-trajectory strategy for the word. Some words are more easily articulated and / or recognizable for the system than others. This process is heavily affected by feedback from the system in form of misrecognitions. At this stage the user eventually knows the limitations of the system for a specific word and can reproduce a pen-gesture articulation that is reliably recognized. Over time this articulation fixates in the user's motor memory. At this point, the autonomous stage is reached.

In general motor skills are learned by repetition [Rosenbaum, 1990; Willingham, 1998]. A common example of muscle memory is keypad codes. Many people do not remember the keypad code by a number sequence. Instead a spatial pattern is recalled when the code needs to be entered. Many keypad codes can be stored without conflict because their meaning is different. One keypad code is used for bank withdrawals, another is used to access the office, etc.

An open question is how much effort a user needs to invest to learn the pen-gestures. There is evidence that repetitive training of spatial actions by adults unconsciously consolidates from a short-lived representation in the brain into a long-term established form [Shadmehr and Holcomb, 1997]. This process does not stop when practice ends, rather it unconsciously continues in the human brain many hours after practice. Sleep is also assumed to be an important factor for efficient consolidation of motor skill [Shadmehr and Wise, 2006]. Strong long-term motor memory consolidation appears to at least in part be an unconscious process. Therefore it may not be necessary for a user to write the same words many times over in sequence once the user knows how to reliably reproduce the word. Experiment 3.1 later in this chapter shows that users typically learn to recall (without a software keyboard as a visual reference) 15 pen-gesture word shapes per 45 minute training session.

#### 3.1.3.1 *Relation to Marking Menus*

An interface with a similar skill transition path is marking menus [Kurtenbach, Sellen and Buxton, 1993]. Marking menus are graphical pie menus where users make selections by moving the pen in angular directions. Pie menus are hierarchical menu structures. The selection of a pie menu “slice” can trigger the display of an additional nested pie menu, and so on.

When the user makes a selection in the pie menu the user is implicitly articulating a pen-gesture pattern. Initially the user does not know the contents within the pie menu and has to visually explore the alternatives, similar to menu selections in traditional hierarchical linear pull-down menus. When a particular menu selection is repeated, the movement pattern of the pen-gesture required to reach the selection becomes reinforced in the user’s muscle memory. Given enough practice, the user can quickly articulate the pen-gesture pattern directly from muscle memory rather than visually navigating the pie menu structure.

To create an incentive for users to articulate pen-gestures to reach the pie menu items rather than resorting to visual navigation Kurtenbach, Sellen and Buxton [1993] introduced a delay for the graphical pie menu to pop-up. When users press down the pen the pie menu is not revealed until after a time delay. If the user knows the pen-gesture the user can directly articulate the pen-gesture for the menu selection. The pen-gesture is recognized by an angular pattern recognizer. Otherwise, the user waits until the graphical pie menu is revealed and navigates the pie menu to the desired menu item.

In marking menus there is a distinction between navigation and specification. In the first case the user is performing a navigation task when traversing a pie menu structure. In the second case the user is performing a pen-gesture specification. In the latter case the underlying structure is flat. For example, if the pie menu contains 40 menu items, the pattern recognizer must be able to recognize 40 different pen-

gestures. From the pattern recognizer's point-of-view there is no hierarchy among the pen-gestures.

In comparison, continuous shape writing does not feature a delay and the transition from novice to expert is not binary (menu vs. no menu) but in a continuum. There is a contrast to the word entry versus menu selection task that is important to emphasize. When the user initially performs menu selection the user must navigate a structure. The desired menu item can be in many places and on many levels. In fact the desired menu item may be completely absent in the menu structure. In contrast, when writing a word the user generally knows the word exists in the system. Therefore there is no need for navigation via any popup menu structure or similar mechanism. Instead, in continuous shape writing the software keyboard serves as a visual map for the specification task. The user articulates a pen-gesture pattern by moving from letter key to letter key in the intended word. Continuous shape writing aids users in the specification task by presenting a visual map of the keyboard to the user. In contrast, marking menus do not guide users in the specification task: when the marking menu is not revealed the display is blank. Marking menus only aid the user in the navigation task.

#### **3.1.4 Disambiguating Pen-Gestures and Pen Taps**

Note that continuous shape writing does not preclude the use of a traditional software keyboard to type in Internet bank codes and other sensitive information for example. However, some care must be taken to effectively and reliably disambiguate users' pen-gestures from pen taps. The problem is not trivial because some words such as *an* on the QWERTY layout are comprised of adjacent letter keys. Further, touch-screens and tablets tend to return unreliable measures for pen taps, often exaggerating the distance traveled by the pen from pen-down to pen-up. Thus a simple pen-trace length-based threshold does not work. Fortunately, an effective method to distinguish pen taps from pen-gestures is to find the number of keys the user's pen trace intersects. If this number is larger than one, the input can be regarded as a pen-gesture. Empirical measurements of user's tapping patterns have shown that users tend to tap closely to the center of a key [Goodman, Venolia, Steury and Parker, 2002; Zhai, Sue and Accot, 2002]. In conjunction with the principle that all letters crossed can potentially be a continuous shape writing pen-gesture, the above procedure maximizes the probability that an intended tap or pen-gesture is interpreted correctly.

## **3.2 User Interface**

---

Shape writing is built on a foundation of pattern recognition of users' input. As such, there is an added complexity layer between the user and the system. Since pattern recognition can result in misrecognitions the added complexity layer is another "failure point" of the interactive system that does not exist in a traditional software keyboard. To alleviate the issue effective user interfaces are required that



simultaneously prevent errors and minimize user effort to correct them when they do happen.

### 3.2.1 Keyboard Design

The most obvious, and in fact to some extent critical, user interface for shape writing is the design of the software keyboard.

The original QWERTY typewriter layout has been adopted in desktop computer keyboards since the first personal computers. As a result a majority of computer users are proficient in, or at least familiar with, the QWERTY layout. Therefore it is natural that the de-facto layout used for commercial software keyboards is also QWERTY.

However, given the advantage of optimized software keyboard layouts in stylus typing (e.g. [Getschow, Rosen, Goodenough-Trepagnier, 1986] and others) the initial keyboard layout design in the first shape writing prototype used the optimized ATOMIK [Zhai, Smith and Hunter, 2002] keyboard layout (Figure 3.4). Figure 3.5 and Figure 3.6 show the first practical version of shape writing that can use both the ATOMIK and QWERTY software keyboard layouts.

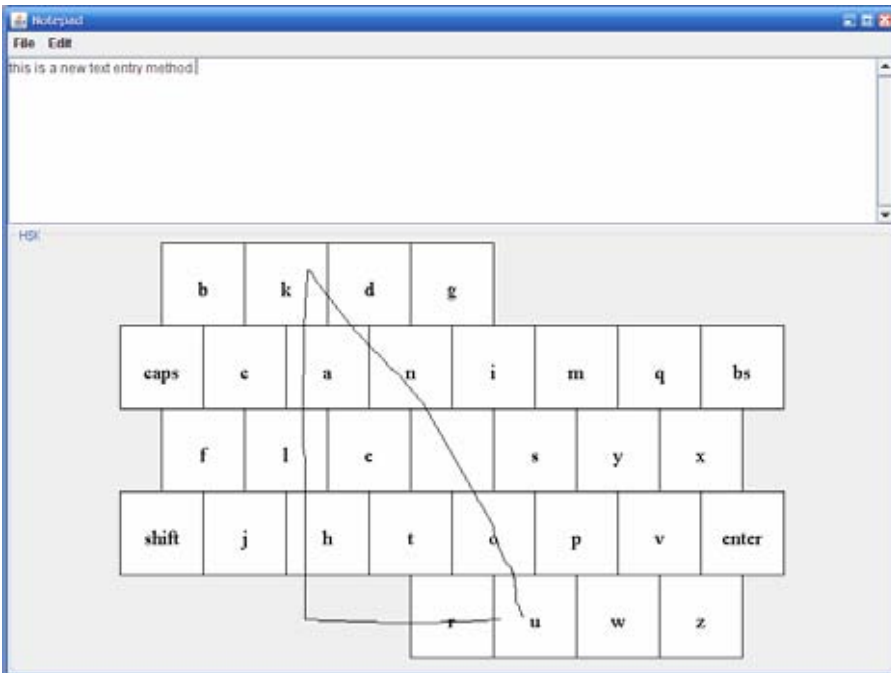


Figure 3.4. The first shape writing implementation. The user's input in the figure will be recognized as *that*. (The abbreviation HSK (Hybrid Shorthand Keyboard) was an early project name abandoned in April 2002.)

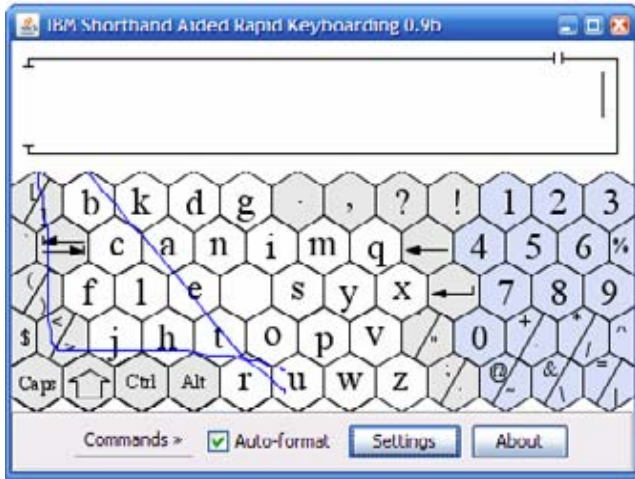


Figure 3.5. The first practical implementation of shape writing. It was released to the public on the IBM alphaWorks *emerging technologies* website in 2004. The user’s input in the figure will be recognized as the word *that*. The keyboard layout is the hexagonal ATOMIK layout.



Figure 3.6. Shape writing using the QWERTY keyboard layout. The user’s input in the figure will be recognized as the word *that*.

Figure 3.7 shows an improved ATOMIK keyboard layout design. Two new sets of keyboard key designs are introduced. The first new key is the quad-key that contains four different keys. The second new key is the split-key that contains two different keys split along the diagonal. The new keys are introduced to avoid the user having to initially press SHIFT to reach common keys. The simple motivation is that one Fitts’ law task with a smaller target width is faster than two Fitts’ law tasks with larger target widths (within limits) [Fitts, 1954]. Since Fitts’ law predicts slower response times on smaller target widths less frequently used letter keys are grouped into split-

keys. Even less frequently used keys are grouped into quad-keys. To avoid users accidentally triggering system keys such as ALT and CONTROL, these keys have a slightly decreased target width and are separated by a no-action border area. In consideration of Benford's law that states (roughly) that the digit one (1) occurs 30% of the time in most writing, the 1-key, as well as the 0-key, is assigned a full-size key.

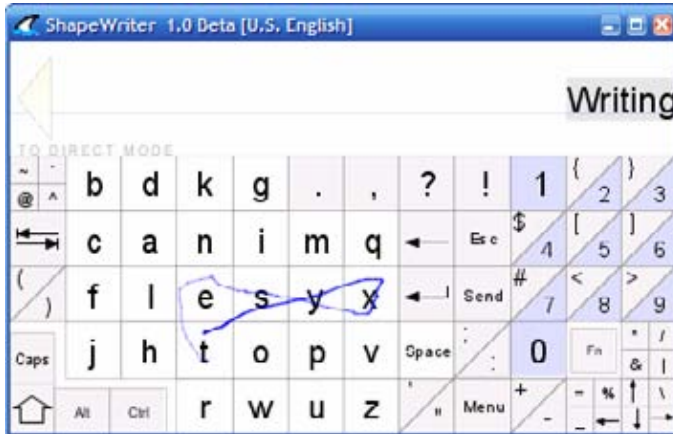


Figure 3.7. Shape writing using the re-designed ATOMIK layout. The user's input will be recognized as the word *text*. Ink fades out gradually over time (described in the Feedback subsection later in this section).

### 3.2.2 Error Correction

All text entry methods inevitably lead to errors. Therefore it is important to enable fast and flexible error correction mechanisms. For instance, with a standard desktop computer setup, the user can use the BACKSPACE, DELETE, HOME, END and arrow keys to move the caret around in the document, and delete text in both the forward and backward direction using the DELETE and BACKSPACE keys). The de-facto direction-manipulation text editing interface used by the graphical user interfaces (GUI) allows the user to move the caret at any desired position in the text. In fact, the search for comparatively more effective tools for text editing led to the publication of a now famous computer input device study paper by Card, English and Burr [1978].

Efficient error correction can be improved when words rather than characters should be corrected. For this purpose the continuous shape writing system uses a user interface component called an edit buffer (see Figures 3.8 and 3.9). Words appear in the edit buffer to the right and pushes existing words to the left. When words cross the left edge they are synthesized into keyboard key strokes and injected into the operating system's key stroke dispatch queue.

### 3.2.2.1 Correcting a Confusion Error

With continuous shape writing, words are written on a word-by-word basis. This means an error results in an entire word being incorrect. The system alleviates this problem by providing several pen-gesture functions for intuitive editing. Figure 3.8 shows how a user deletes the two words *and editing* by simply crossing them. Any individual word or sequence of words can be deleted by a crossing action. The intuitiveness of crossing out words as a delete action has been confirmed in the literature [Wolf and Morrel-Samuels, 1987].

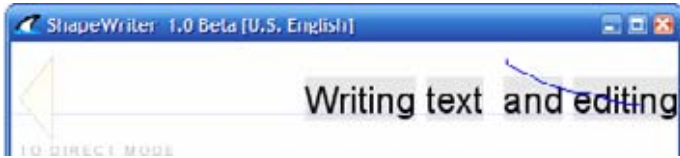


Figure 3.8. The user deletes a series of words by crossing them.

Every word that is outputted from the continuous shape writing recognizer is marked with gray background to signal to the user they are selectable. The user can delete such words completely in one action by pressing the BACKSPACE key once. This function allows users to quickly try again if misrecognition occurred. It is also possible to change the position of the text caret by crossing words top-down. The text caret repositions to either the beginning or the end of the crossed word, depending on where the user crossed it. For example, in Figure 3.9 the user has repositioned the text caret behind the word *Writing*. The part of the crossed word that is closest to the text caret becomes highlighted for a brief period of time to show the user where the text caret moved and reinforce the idea that the part of the word that is crossed matter to the text caret repositioning function. The highlight gradually fades out during a brief period (800 ms).



Figure 3.9. The user has repositioned the text caret by crossing the rightmost part of the word *Writing* from above.

When the user selects a word output from the recognizer a pull-down list of alternate words is displayed. The alternate words are ranked by a confidence score output from the recognizer (essentially a measure of how close the words are for the recognizer in relation to the user's input). The first entry in the list is always the top ranked word that was output by the recognizer. The user can select an alternate word that replaces the word in the edit buffer.

To enable efficiency while simultaneously reducing novice users' frustration, the selection process of an alternate word supports two different selection methods. With the first method the user first taps on the word with the pen to reveal the list of alternate words. Then the user selects the desired word from the list with another tap with the pen. In the second method, the user taps and holds the pen down on the word. Then the user slides the pen to the desired word and lifts up the pen. The second method is more fluid but informal user testing revealed that some users are used to pull-down menu widgets appearing after a tap only. These users were confused when the menu immediately disappeared when they lifted the pen.

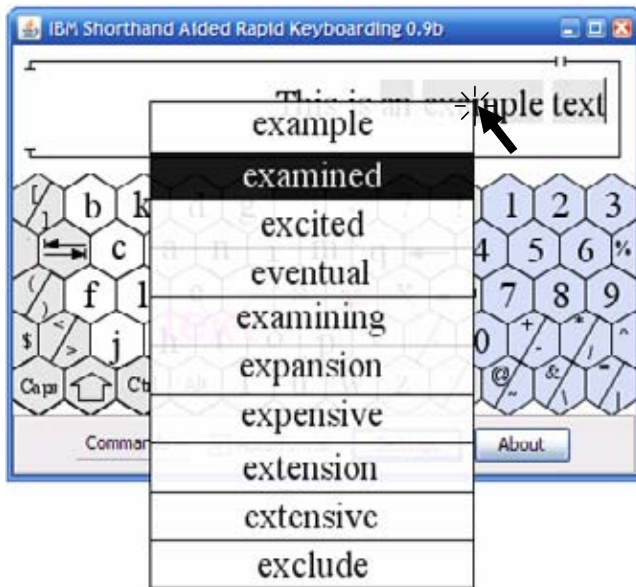


Figure 3.10. The user reveals a semi-transparent list of alternative word candidates by selecting a word.

### 3.2.2.2 Correcting an Out-of-Vocabulary Error

Another possible error is the out-of-vocabulary (OOV) error [Furnas, Landauer, Gomez and Dumais, 1987]. If a user wants to write a word that does not exist in the lexicon the user has to add that word explicitly. Since continuous shape writing recognition does not rely on training data (see Chapter 4 for details) words can be immediately added to the system's lexicon. If a user taps a word using the software keyboard the system automatically performs a check to see if the word exists in the system lexicon. If it does not, the user's tapped word is drawn with a surrounding dashed rectangle to create an affordance for the user to click on it (see Figure 3.11). When the user clicks on the word, the user can select the ADD TO LEXICON function from the pull-down menu (see Figure 3.12). The new word can be immediately shape written by the user afterwards.

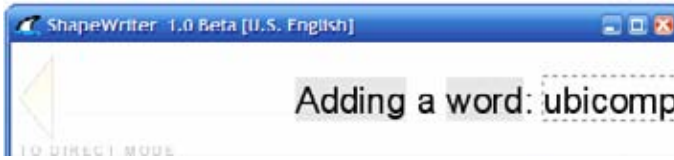


Figure 3.11. The user has written the word *ubicomp* that does not exist in the lexicon.

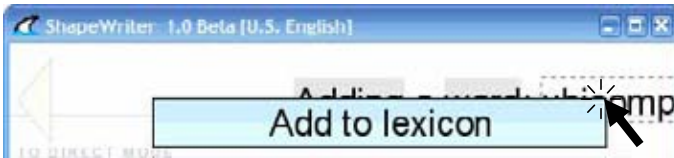


Figure 3.12. The user adds the word *ubicomp* to the lexicon.

### 3.2.3 Feedback

#### 3.2.3.1 Display of Recognized Word and Reinforcement of the Ideal Shape

As soon as the user lifts up the pen or finger, the recognized word is displayed. The recognized word is displayed at the point of pen-up since that is the most likely position where the user's focus of visual attention will be.

In addition the ideal shape of the recognized word is displayed for a brief period of time (600 ms). The display of the ideal shape reinforces the shape of the word to the user. Figure 3.13 shows the ideal shape for the word *system* displayed over the software keyboard. The dot indicates the starting position of the ideal shape.

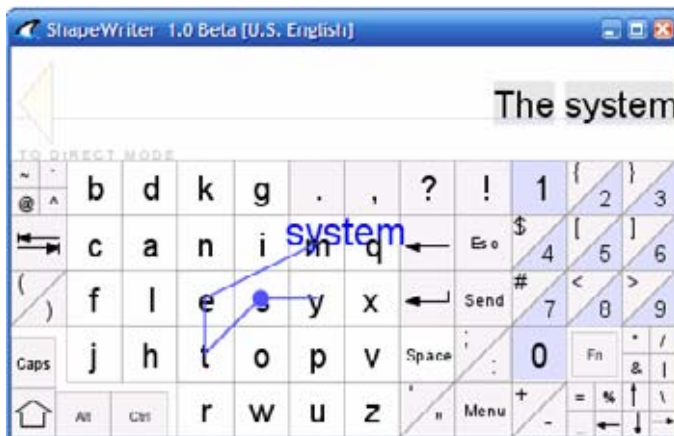


Figure 3.13. The user's pen-gesture has been recognized as the word *system*. The recognized word is displayed at the point where the pen was lifted.

### 3.2.3.2 Minimizing Pen-Trace Clutter

Displaying the ink of the pen as the user articulates a pen-gesture is advantageous. First, it gives the user information that the pen motion is still recorded. Second, it provides a sense of orientation to the user on where the pen has traveled on the keyboard. However, some words are longer than others and when the pen has moved back and forth on the keyboard the visual clutter from the pen trace becomes distracting. A solution is to progressively fade the tail part of the pen-gesture when the trace increases in length (see Figure 3.14).

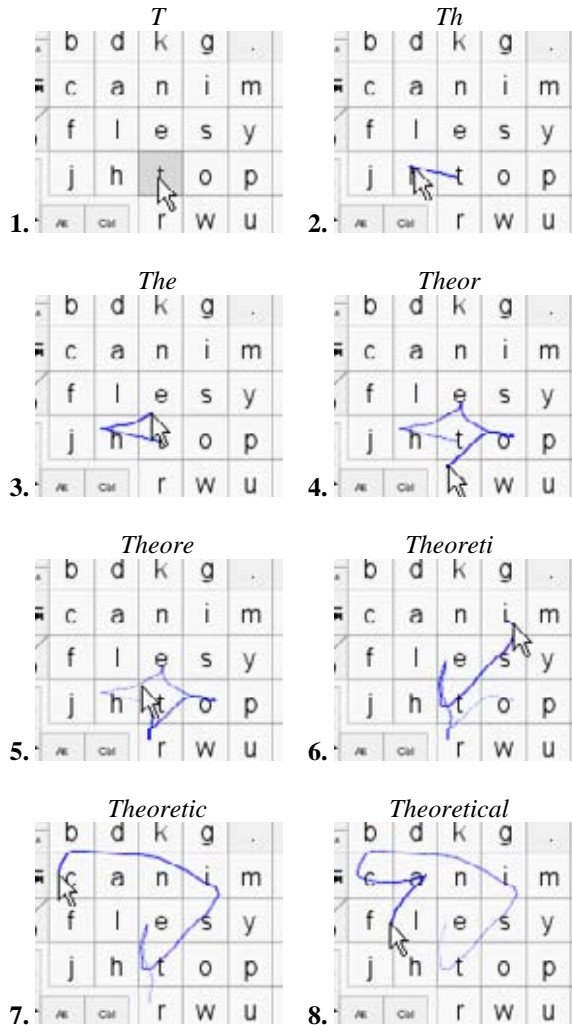


Figure 3.14. The user is writing the word *theoretical*. The tail part of the user's pen trace gradually fades out as the pen trace progresses to minimize visual clutter.



Figure 3.15 shows the effect of pen-trace clutter reduction when the user has written the word *theoretical* on the ATOMIK layout using continuous shape writing. As can be seen in Figure 3.15, beyond a certain threshold there is no clear advantage of displaying the entire pen trace because the pen trace is simply too cluttered to be understandable.

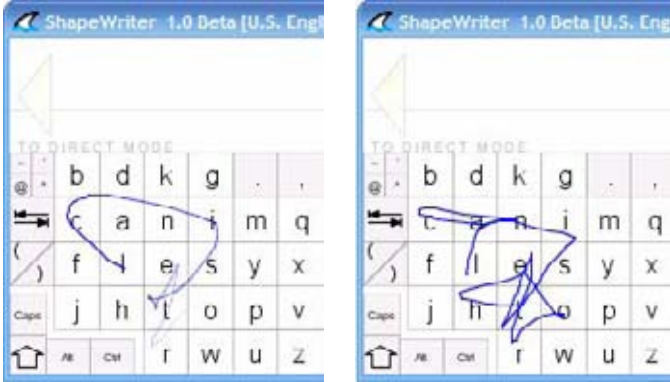


Figure 3.15. The difference between reduced (left) and non-reduced (right) pen-trace clutter when writing the word *theoretical*.

### 3.2.3.3 Morphing Visualization

A morphing algorithm is implemented to help novice users understand what part of their pen traces contributes to the match of the recognized word. After the user has lifted up the pen the complete pen trace of the user is drawn with blue ink and the ideal shape of the word is drawn in red ink (see Figure 3.16). Thereafter both traces are resampled into an equal large number of equidistant sample points. Next, the sample points that are indexed at the same position in the user's pen trace and the ideal word shape are connected by imaginary lines. A pair of two imaginary lines is then formed into an area by connecting the four vertices in the line-pair in sequence. This area is painted by a low-translucent blue color. The visual area explicitly communicates the spatial distance between the pen trace and the ideal word shape to the user. To create a stronger visual effect the user's pen trace is gradually transformed into the ideal word shape over time (Figure 3.16). The intermediate forms of the user's pen trace are found by linear interpolation.



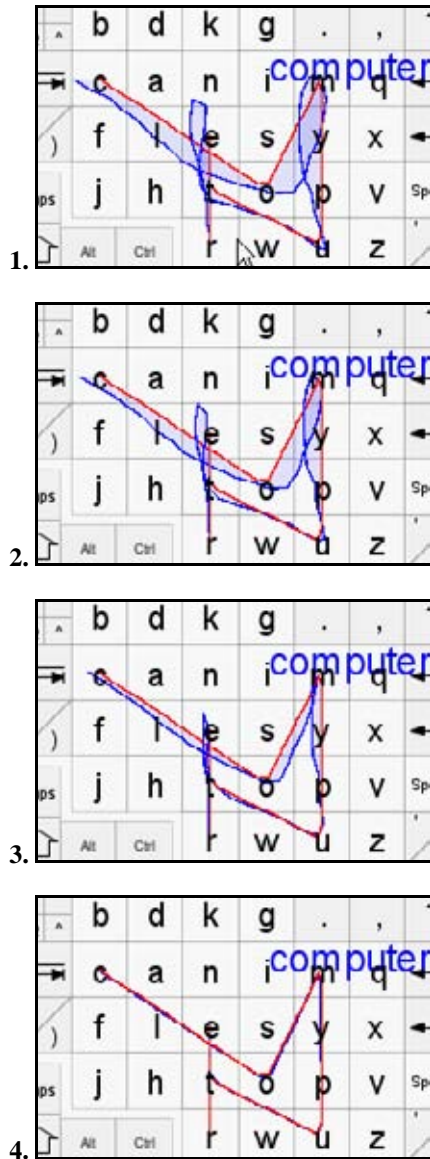


Figure 3.16. The user has written the word *computer*. The user's pen trace is gradually morphing towards the ideal word shape.

The effect of the morphing visualization has not been formally tested. Informal user testing shows that some users think it is “cool” and “interesting”. It appears morphing visualization has a positive qualitative effect of keeping some novice users interested in the text entry method. The downside of using morphing visualization is that it demands visual attention and adds clutter to the interface. Academically it remains an open question whether there are any measurable quantifiable effects of this form of

visual feedback on for example novice users' error rates, or the closeness of users' pen trace in relation to the ideal word shape. In practice all visual feedback features are selectable options in the software. Morphing visualization is probably also effective as a support-tool in a pen-gesture design toolkit, e.g. [Long, Landay and Rowe, 1999].

### 3.2.4 Avoiding the Hand Obscuring the Keyboard Layout

A practical problem with continuous shape writing is that the user's hand tends to obscure the keyboard. This problem has also been reported by participants in experiments (see Experiment 3.2, later in this chapter).

A solution is to display a "virtual" projection of the software keyboard above the real software keyboard (Figure 3.18). A red cursor on the projection indicates the current pen location.

A problem with the phantom keyboard approach is that it does not scale down to small mobile phone displays. In such cases, a method such as offsetting the cursor (similar to take-off [Potter, Weldon and Shneiderman, 1988]) can be used.



Figure 3.17. Example of writing with a phantom keyboard. The user is in the process of writing the word *keyboard*.

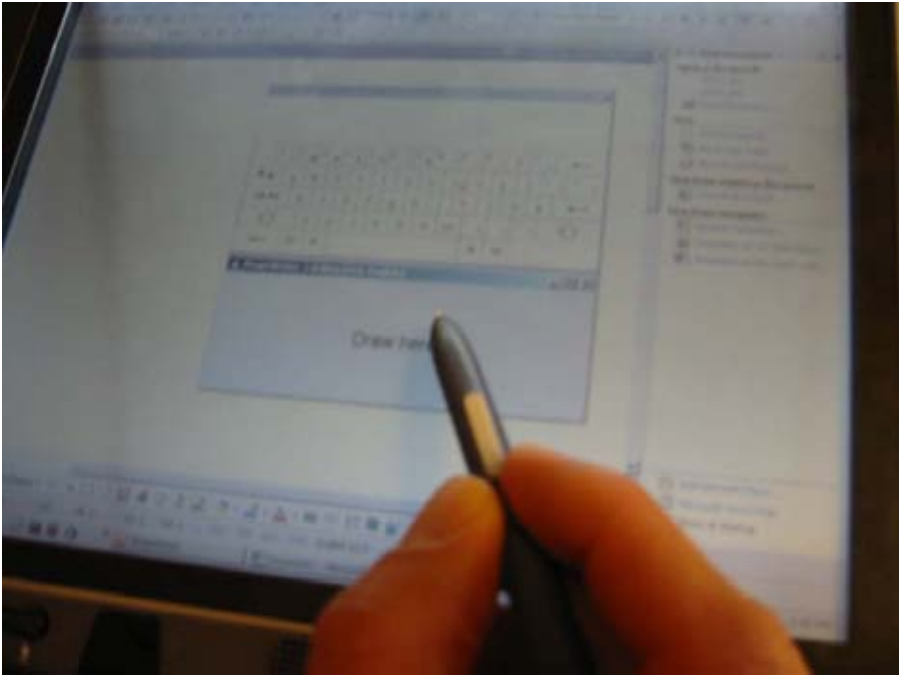


Figure 3.18. Photograph of the phantom keyboard activated on a computer tablet.

### **3.3 Shape Writing on Mobile Phone**

---

Continuous shape writing has been implemented for mobile phones (Figure 3.19). This section outlines the user interface changes that were necessary to streamline the user experience for the limited mobile phone form factor.

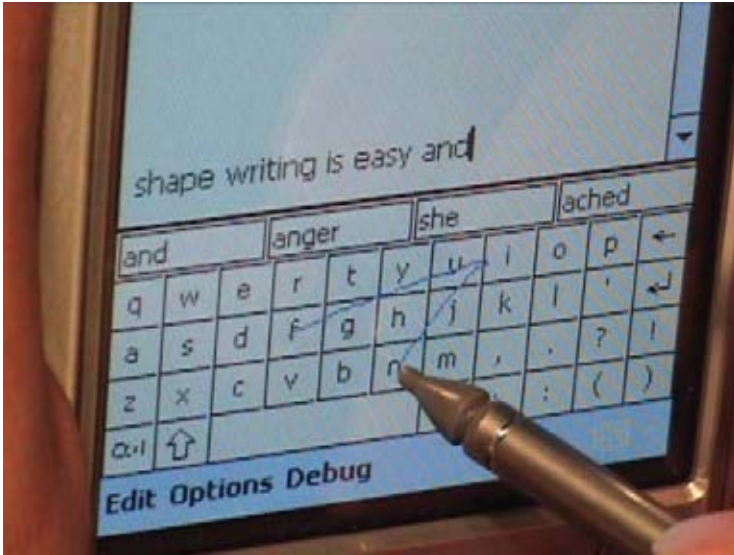


Figure 3.19. The user interface for continuous shape writing on mobile phones. The user is writing the word *fun*.

The keyboard QWERTY and ATOMIK keyboard layouts have been streamlined to fit the small mobile phone screen (Figure 3.20). Only essential letter and system keys are displayed on the first level.

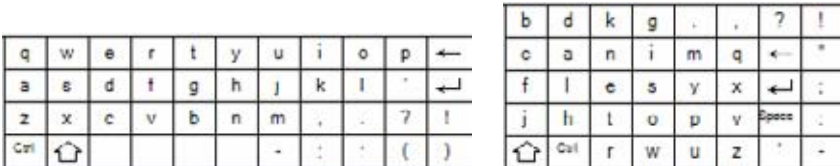


Figure 3.20. The QWERTY and ATOMIK software keyboard layouts for continuous shape writing on mobile phones.

Because of the small screen estate mobile phones cannot afford a secondary user component such as the edit buffer. Instead words are synthesized into keystrokes and injected into the keystroke queue of the mobile phone operating system as soon as the words are recognized. To delete unintended or misrecognized words the user can use any of the built-in editing functions made available from the mobile phone operating system.

In addition, alternate word candidates are displayed as soon as the user lifts up the pen. In Figure 3.21 the user has written the word *the* (marked in blue color). The alternate words *thyme* and *thrive* that also partially match the user’s pen trace are displayed next to *the*. The user can change the word at the system text caret by selecting any of the alternate words. If none of the alternate words is the user’s

intended word, the user can select the recognized word (*the* in Figure 3.21) and immediately delete the entire word at the caret.

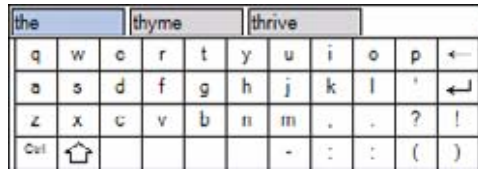


Figure 3.21. The alternate word candidates for the user's pen-gesture are displayed immediately in close vicinity to the software keyboard.

The mobile phone implementation remembers the alternate words for a fixed set of outputted words. In the current implementation this buffer can hold 200 alternate words. When the user changes position of the system text caret the continuous shape writing system automatically retrieves the alternate words for the word at the text caret. Therefore the user does not need to constantly scan the list of alternatives for every word entry. Rather, text correction can happen when for example the user is ready to send a written email and proof-read the text.

The mobile phone continuous shape writing pattern recognizer uses a tweaked version of the recognition algorithms described in Chapter 4. As an example of such a tweak the lexicon is encoded into a compact format that makes it possible to simultaneously derive both the original string (the word) and the pattern representation. Assuming average word length is five characters, using this lexicon compression scheme 50,000 words and their pattern representations can be stored in only 450K memory. Using indexing the average latency is less than 20 ms with a lexicon consisting of 50,000 patterns on a mobile phone equipped with a 32-bit 168 MHz Texas Instruments OMAP1510 CPU. Continuous shape writing is not only a concept, it is already practical for mobile phones.

### 3.4 Practice Game

A challenge for new text entry products is the need for an easy and fun tutorial, or game, on how to effectively write text using the new method. There is a multitude of typing tutors that have been developed for the desktop keyboard. One of the highly polished typing tutors is *Typing of the Dead* by Sega Entertainment where the user kills zombies by typing text shown in front of them. A variant of that game has recently been researched in the context of learning how to write Japanese characters [Stubbs, 2003].

A balloon practice game was created to teach shape writing (see Figure 3.22). The basic game concept is a variant of the traditional typing tutor game in which the user must respond to letters, or words, on the screen as quickly as possible. In this version balloons are floating upwards on the screen. In each balloon a word is displayed. To

pop the balloon the user has to shape write the word displayed on the balloon. To get a better score and accuracy the user must pop as many balloons as possible while avoiding writing incorrect words.



Figure 3.22. The shape writing practice game. The score next to the balloon icon indicates how many words that were successfully inputted (in the figure 78). The percentage next to the crosshair icon shows the accuracy (in the figure 75%). By selecting the `AUTO` option the system automatically pen-gestures the currently shown words.

The design goals of the game were threefold:

1. Efficiency. It should teach users to shape write the most common words in a short amount of practice time
2. Fun. The game should be engaging, or at least not repetitive and boring.
3. Challenge. The game should challenge users to write faster.

### 3.4.1 Efficiency

Not all words occur at equal frequency. In fact, word distribution is heavily skewed and can be modeled with Zipf's law [Zipf, 1935]:

$$P(r) \sim \frac{1}{r^\alpha} \quad (3.1)$$

where  $P(r)$  is the probability of occurrence of the word,  $r$  is the rank of the word, and  $\alpha$  is close to unity.

An example of Zipf's law is the distribution of English words. In the British National Corpus (BNC) 46% of the corpus is composed of the 100 most frequently used English words. Therefore even practicing the top 100 words will greatly benefit the user. This is also the reasoning behind the practice game displaying words in the order of their frequency of occurrence (in a large corpus). For example, the first words displayed in the practice game for U.S. English are: *the, of, and, etc.*

Not all words are equally difficult, and previous research indicates that users think it is frustrating to be forced to re-write words they are already good at [Zhai, Sue and Accot, 2002]. On the other hand, until a word shape is completely subsumed into a user's motor memory, repetition is necessary in order to push the user to expert mode. For this reason, words are scheduled to appear according to an expanding rehearsal interval (ERI) algorithm [Landauer and Bjork, 1978]. See Experiment 3.1 for a detailed explanation of ERI.

By tapping on the *Auto* button the game demonstrates to the user how to write the ideal word shape for the currently displayed word. Psychomotor research has shown the benefit of "observational practice" [Kohl and Shea, 1992]. The demonstration mode can also serve as an instructive interface and prevent habits stemming from users getting confused on how recognition works. For instance, in Experiment 5.2 in Chapter 5 it was observed that some users thought that arcing around the keys would make recognition easier as opposed to directly cross the letter keys. However, since shape writing recognition is agnostic regarding the individual keys crossed this habit only creates more opportunities for confusion in the recognizer.

### 3.4.2 Fun

To let the game feel less predictable the balloons do not float up to the sky using a direct vertical route. Instead a balloon follows a movement path defined by a spline whose two control points vary in the  $x$  and  $y$  directions with pre-set intervals. The acceleration of a balloon along the path also has a random component that enables a balloon to temporarily "catch wind" and float a little faster along its movement path.

### 3.4.3 Challenge

To motivate users to write fast, the balloons do not "stick" at the top of the screen. Instead balloons disappear, and are counted as a "miss", when they float above the view port. Therefore the user must quickly write the prompted words on the balloons before they disappear. The number of balloons displayed simultaneously on the screen, and the speed in which the balloons rise in the air increase slightly as the game progresses to keep the game interesting when the user improves. Another variation under consideration is to allow novice users to "freeze" the currently displayed balloons and repeatedly practice those words. Freezing the screen allows users who do not currently understand how shape writing works to practice words in a playful, yet relaxed setting. To avoid user frustration in having to start the game from the

beginning (e.g. [Zhai, Sue and Accot, 2002]), and to keep players interested in going back to the practice game, the current player state is automatically saved and restored every time the user exits or starts the game.

### 3.5 Localization

Continuous shape writing can be directly used in all languages that define words as a sequence of letters drawn from a limited-size alphabet. Currently, continuous shape writing has been implemented for U.S. English, German (Figure 3.23) and Korean (Figure 3.24). This section describes how the continuous shape writing system was modified to handle German and Korean.

#### 3.5.1 German

The German alphabet is similar to English. The only essential difference is that the German alphabet also contains the diacritic (umlaut) letters *ü*, *ä* and *ö*; and the ligature *ß* (Eszett).

The standard German software keyboard is based on the QWERTY layout. To improve typing performance when working with German text the *y* and *z* letter keys have switched place. In addition, the extra letters in the German alphabet are inserted on the right hand side of the Latin letters. Since the top-left row-letter sequence has changed in relation to QWERTY, the German layout is called QWERTZ (Figure 3.23).



Figure 3.23. Writing German words with continuous shape writing on the QWERTZ software keyboard layout.

#### 3.5.2 Korean

The Korean language uses an alphabet that consists of 24 letters called jamo. Of these 14 are consonants and 10 are vowels. Unlike the Latin alphabet text composition is not merely a linear stream of jamo. Rather, jamo are composed into syllabic units called Hangul. The composition of jamo into Hangul follows algorithmic rules.



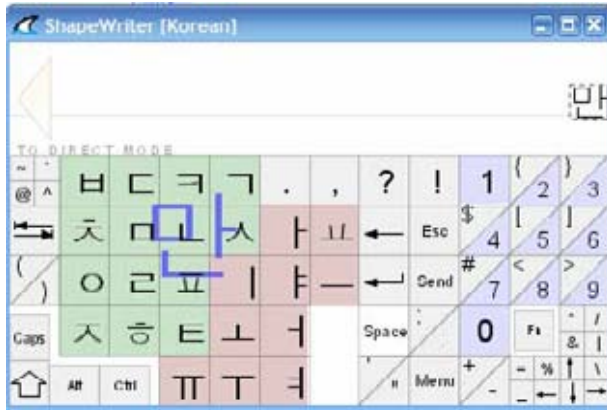


Figure 3.24. Continuous shape writing of Korean text. The keyboard layout is rather arbitrary and designed by me for testing purposes only. The green keys are consonants and the red keys are vowels.

The Unicode standard [The Unicode Consortium, 2003] defines an algorithm that deterministically translates jamo into Hangeul. To achieve this, the Unicode standard defines more jamo characters than there are jamo letters in the jamo alphabet. In the Unicode standard 19 lead, 21 vowel and 27 tail jamo characters are defined. Separate representation of jamo with identical glyphs is required for a canonical jamo string representation that can be unambiguously translated into Hangeul (see *Hangeul Syllable Composition* algorithm (page 87) in [The Unicode Consortium, 2003]).

In the following discussion the Unicode character encoding in hexadecimal radix will follow in parenthesis after the jamo.

By inspection of the Unicode charts, it is evident the excessive lead, vowel and tail jamo characters are merely combinations of the jamo alphabetic primitives (letters). For instance, the extra jamo lead ㅏ (0x11A9) is merely a repetition of the jamo letter ㅏ (0x1100). The lead jamo ㅏ (0x1100) has a corresponding tail jamo ㅏ (0x11A8) with an identical glyph. Note that the specific Unicode character encodings are critical. The jamo to Hangeul translation will not work correctly unless the tail jamo is at the end of the jamo string and vice versa for the lead jamo.

For continuous shape writing the details of jamo to Hangeul translation is not critical. On the contrary, from a user's point-of-view it makes more sense to only represent the core 24 jamo letters that have distinct glyphs as letter keys on the software keyboard.

This can be achieved by designing a surjective function that maps canonical Unicode jamo strings into the 24 jamo letters that are represented by letter keys. Tables 3.1, 3.2 and 3.3 lists lookup tables for three surjective functions that achieve this mapping. Note that the *Hangeul Syllable Decomposition* algorithm [The Unicode Consortium, 2003] is well defined for all canonical jamo strings. Since all Hangeul syllables can be

decomposed into canonical jamo strings this means (by transitivity) that all Hangul syllables can be encoded into the 24 jamo letter keys on the software keyboard (Figure 3.25).

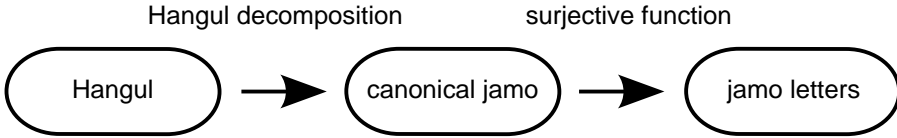


Figure 3.25. Converting Hangul to jamo letters.

The surjective mapping makes the reverse process of converting continuous shape writing jamo letters into canonical jamo strings ambiguous. To avoid this ambiguity the system has a table that maps continuous shape writing jamo letter streams back to the original canonical jamo strings. This table is consulted when the canonical jamo string is required for conversion to Hangul for display.

Table 3.1. Surjective function from lead jamo consonants to jamo letters.

In	Unicode [In]	Out	Unicode [Out]
ㄱ	0x1100	ㄱ	0x1100
ㄴ	0x1101	ㄱ	0x1100
ㄷ	0x1102	ㄷ	0x1102
ㄸ	0x1103	ㄷ	0x1102
ㄹ	0x1104	ㄷ	0x1102
ㄺ	0x1105	ㄹ	0x1105
ㄻ	0x1106	ㄹ	0x1105
ㄼ	0x1107	ㄹ	0x1105
ㄽ	0x1108	ㄹ	0x1105
ㄾ	0x1109	ㄹ	0x1105
ㄿ	0x110A	ㄹ	0x1105
ㅀ	0x110B	ㅀ	0x110B
ㅁ	0x110C	ㅁ	0x110C
ㅂ	0x110D	ㅁ	0x110C
ㅃ	0x110E	ㅁ	0x110C
ㅄ	0x110F	ㅁ	0x110C
ㅅ	0x1110	ㅅ	0x1110
ㅆ	0x1111	ㅅ	0x1110
ㅇ	0x1112	ㅇ	0x1112

Table 3.2. Surjective function from tail jamo consonants to jamo letters.

In	Unicode [In]	Out	Unicode [Out]
ㄱ	0x11A8	ㄱ	0x1100
ㄲ	0x11A9	ㄱ	0x1100
ㄳ	0x11AA	ㄱ, ㄴ	0x1100, 0x1109
ㄴ	0x11AB	ㄴ	0x1102
ㄵ	0x11AC	ㄴ, ㄷ	0x1102, 0x110C
ㄶ	0x11AD	ㄴ, ㄹ	0x1102, 0x1112
ㄷ	0x11AE	ㄷ	0x1103
ㄸ	0x11AF	ㄷ	0x1105
ㄹ	0x11B0	ㄷ, ㄱ	0x1105, 0x1100
ㄺ	0x11B1	ㄷ, ㄴ	0x1105, 0x1106
ㄻ	0x11B2	ㄷ, ㅁ	0x1105, 0x1107
ㄼ	0x11B3	ㄷ, ㄴ	0x1105, 0x1109
ㄽ	0x11B4	ㄷ, ㅂ	0x1105, 0x1110
ㄾ	0x11B5	ㄷ, ㅅ	0x1105, 0x1111
ㄿ	0x11B6	ㄷ, ㅈ	0x1105, 0x1112
ㅁ	0x11B7	ㅁ	0x1106
ㅂ	0x11B8	ㅂ	0x1107
ㅃ	0x11B9	ㅂ, ㄴ	0x1107, 0x1109
ㅅ	0x11BA	ㅅ	0x1109
ㅆ	0x11BB	ㅅ	0x1109
ㅇ	0x11BC	ㅇ	0x110B
ㅈ	0x11BD	ㅈ	0x110C
ㅊ	0x11BE	ㅊ	0x110E
ㅋ	0x11BF	ㅋ	0x110F
ㅌ	0x11C0	ㅌ	0x1110
ㅍ	0x11C1	ㅍ	0x 1111
ㅎ	0x11C2	ㅎ	0x1112

Table 3.3. Surjective function from jamo vowels to jamo letters.

In	Unicode [In]	Out	Unicode [Out]
ㅏ	0x1161	ㅏ	0x1161
ㅑ	0x1162	ㅏ, ㅣ	0x1161, 0x1175
ㅓ	0x1163	ㅓ	0x1163
ㅕ	0x1164	ㅓ, ㅣ	0x1163, 0x1175
ㅗ	0x1165	ㅗ	0x1165
ㅛ	0x1166	ㅗ, ㅣ	0x1165, 0x 1175
ㅜ	0x1167	ㅜ	0x1167
ㅠ	0x1168	ㅜ, ㅣ	0x1167, 0x1175
ㅡ	0x1169	ㅡ	0x1169

ㅏ	0x116A	ㅏ, ㅑ	0x1169, 0x1161
ㅓ	0x116B	ㅏ, ㅑ, ㅣ	0x1169, 0x1163, 0x1175
ㅕ	0x116C	ㅏ, ㅣ	0x1169, 0x1175
ㅗ	0x116D	ㅗ	0x116D
ㅛ	0x116E	ㅛ	0x116E
ㅜ	0x116F	ㅛ, ㅑ	0x116E, 0x1165
ㅠ	0x1170	ㅛ, ㅑ, ㅣ	0x116E, 0x1165, 0x1175
ㅡ	0x1171	ㅛ, ㅣ	0x116E, 0x1175
ㅜ	0x1172	ㅜ	0x1172
ㅡ	0x1173	ㅡ	0x1173
ㅝ	0x1174	ㅡ, ㅣ	0x1173, 0x1175
ㅣ	0x1175	ㅣ	0x1175

Table 3.4. The test lexicon used for Korean.

Jamo	Hangul	Sound Encoding
ㅏ, ㅑ	ㅏ	ma (syllable)
ㅓ, ㅑ	ㅓ	ha (syllable)
ㄴ, ㅑ	나	na (syllable)
ㄴ, ㅑ, ㅗ, ㅛ	나무	na-mu (word: tree)
ㅏ, ㅑ, ㄱ	목	mok (syllable)
ㅏ, ㅑ, ㄴ	만	man (syllable)

The Hangul column entries in Table 3.4 were inputted by me with the Korean continuous shape writing system directly. Table 3.4 demonstrates three base cases of translation from jamo to Hangul. The first three entries merge pairs consisting of an initial jamo character followed by a vowel into Hangul syllables. The fourth entry merges four jamo characters into two Hangul syllables (a word). The last two entries handle the case where three jamo alphabet primates are composed into a single Hangul syllable.

Figure 3.26 shows the Korean word for *tree* (na-mu) being gestured by connecting the jamo letters ㄴ ㅑ ㅗ ㅛ in sequence. The final Hangul it encoded as the two characters 나무. Figure 3.27 shows the software keyboard where the jamo letter keys are “flipped” to reveal their corresponding sounds.

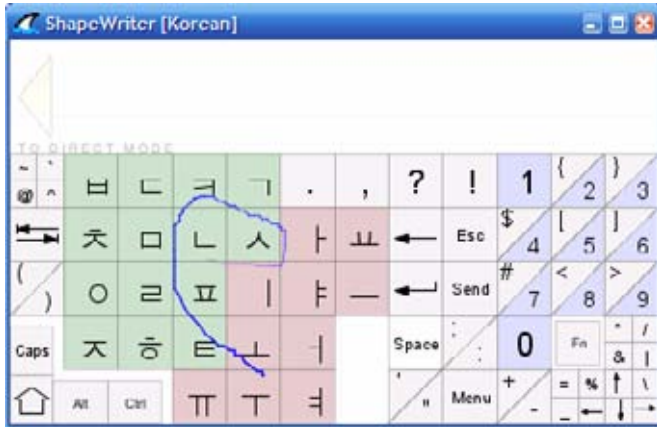


Figure 3.26. Entering the Korean word for tree (나무) using continuous shape writing with a Korean keyboard layout and a Korean lexicon.



Figure 3.27. The Korean software keyboard can optionally display the sounds of the jamo letter keys. In the edit buffer the Korean word for *tree* is shown in Hangeul (나무).

From the foundation outlined in this subsection it is trivial to mine Hangeul corpora and create a practical lexicon for Korean continuous shape writing. Finally, note that there are 20 mappings that consist of two jamo characters, and only two mappings that consist of three jamo characters. To increase efficiency it may be worthwhile to consider introducing three-jamo-character-mappings as separate keys on the software keyboard.

### 3.6 Experiment 3.1: Learning

The above sections have laid out the technical foundation and conceptual motivations for continuous shape writing as an effective and enjoyable text entry method.

However, in the end the usefulness and efficacy of a text entry method can only be captured and validated through empirical studies. Text entry is heavily skill-based and factors that affect this skill are complex and not completely understood. For example, five hundreds years of research in stenography never resulted in a conclusive understanding on either how to design an efficient stenographic system, or which existing stenography system that in some sense would be superior (see Melin [1927; 1929] for a comprehensive survey and analysis). Another example is the typewriter invention and the much debated QWERTY layout. To this day, there is still no clear consensus on the amount of quantifiable advantage of changing the QWERTY layout to DSK (Dvorak Simplified Keyboard) (for U.S. English) [Yamada, 1980; Norman and Fisher, 1982]. In fact, the QWERTY-Dvorak debate created a topic in economics of “path dependency” called *qwertynomics* [David, 1985] (see also Liebowitz and Margolis [1990] for an opposite view).

As a skill-based technique text entry is heavily affected by users’ cognitive abilities and motor control performance. Factors such as age, prior experience [Rosenbaum, 1991] and native language [Isokoski and Linden, 2004], are known to impact motor skill acquisition and performance in text input. In regard to continuous shape writing, some users generally hand write neater and draw better than others. To some extent the spatial ability of drawing appears to be related to users’ amount of practice and general interest in the task. For example, Kozbelt [2001] shows that artists copy line drawings significantly better than non-artists.

It is hard to estimate an average text entry rate because users’ skill, typing style and the text genre affect performance. The true average text entry rate is most likely highly individual and near-impossible to capture in a controlled experiment. Logging long-term users’ actual typing data for several years is probably more informative. In the end, what is important is that the text entry rate is not perceived as a hindrance but rather as an enabler. For example, an inefficient or tedious keypad-based text entry method that causes users to postpone writing urgent emails and thereby negatively affecting users’ work style is clearly unacceptable.

For these reasons and given the scope of the dissertation I leave a comprehensive empirical evaluation of continuous shape writing as future work. Instead, I report on three small-scale initial experiments that each tests different and specific aspects of shape writing. The results obtained from these experiments set out the basic expectations of shape writing performance.

The first experiment focuses on learning. The experiment is designed to answer the two perhaps most basic research questions on continuous shape writing: First, can users learn the pen-gesture shapes for words? Second, if so, what is the typical learning rate and is there a low limit to the number of shapes one can memorize in shape writing?

### 3.6.1 Method

#### 3.6.1.1 Design

The experiment consisted of five sessions. The first session was a practice session only, and the last session was a testing session only. The second, third and fourth sessions consisted of first a testing session and afterwards a practice session.

#### 3.6.1.2 Participants

Six paid volunteers were recruited from the Linköping University campus. Two were male and four were female. Their ages ranged from 20-30 (mean = 24.7, sd = 2.6). None of the participants had any prior experience with either continuous shape writing or the ATOMIK software keyboard layout. All participants were native Swedish speakers, fluent in English.

#### 3.6.1.3 Apparatus

















A Wacom tablet model ET-0405-U was used as the pen input device. The tablet was connected to a Windows 2000 desktop computer. A 21" CRT screen with a desktop resolution of  $1280 \times 1024$  pixels was used as the display.




































































A software keyboard with the ATOMIK keyboard layout was used. The pattern-recognizer for continuous shape writing was based on the elastic matching algorithm [Tappert, 1982]. For a description of how the algorithm was implemented see [Kristensson, 2002]. 100 words were inserted into the lexicon. The pen-gestures were recognized independent of scale and translation. That is, it did not matter to the recognizer where the user articulated the pen-gesture, nor did it matter how large the pen-gesture was in relation to the software keyboard.

#### 3.6.1.4 Material

100 words taken from the British National Corpus (BNC) were used in the experiment. The words consisted of the top ranked words in the BNC. However, since the words would be recognized invariant of scale and translation transformations for the purpose of testing users' ability to learn and remember shapes in this experiment, words that were ambiguous were filtered out and replaced with words ranked 101-300 in the BNC. All words used in the experiment are listed in Table 3.5.

Table 3.5. The words used in Experiment 3.1 and their corresponding shape as defined by the classic ATOMIK layout (Figure 3.29). The dot indicates the starting position of the pen-gesture.

the		that		knowing		while	
and		this		about		problem	
in		these		could		against	
inside		those		think		service	

have		did		people		never	
has		does		after		house	
had		done		right		down	
having		doing		because		school	
he		are		between		report	
him		our		before		start	
his		from		through		country	
it		which		place		really	
its		will		become		provide	
they		were		such		local	
them		said		change		member	
was		can		point		within	
their		whose		system		always	
not		went		group		follow	
for		gone		number		without	
you		other		however		during	
your		another		again		bring	
she		being		world		although	
her		seeing		course		example	
with		knew		company		question	
on							

---

3.6.1.5 Procedure

To improve learning the ERI (expanding rehearsal interval) method was used [Landauer and Bjork, 1978]. ERI has earlier been successfully used in stylus typing learning [Zhai, Sue and Accot, 2002]. With the ERI paradigm words are not repeated uniformly. Instead, the interval between the word repetitions is gradually increased. As Zhai, Sue and Accot [2002] notes this is similar to how people learn foreign words. The practice sessions used the ERI algorithm to schedule the appearance of words. A single word appeared within a set interval initially configured to 30 seconds for the word. If a user successfully reproduced the pen-gesture shape for the word the interval was doubled. If a user failed to reproduce the pen-gesture shape for the word



the interval was left unchanged. After the user had responded to a word the next word scheduled to appear within its rehearsal interval was displayed. If no such word existed, a new unpracticed word was picked from word list, configured with an initial rehearsal interval of 30 seconds, and displayed to the user.

Practice sessions lasted 40 minutes with a 5-minute break in the middle. Participants were encouraged to take breaks anytime they wanted.

The experiment software prompted a single word to the participant (Figure 3.28). If the participant could not remember the word shape the participant was instructed to reveal the software keyboard by pressing the SHOW KEYBOARD button (Figure 3.29). This operation was also necessary when the user was confronted with a previously unseen word.

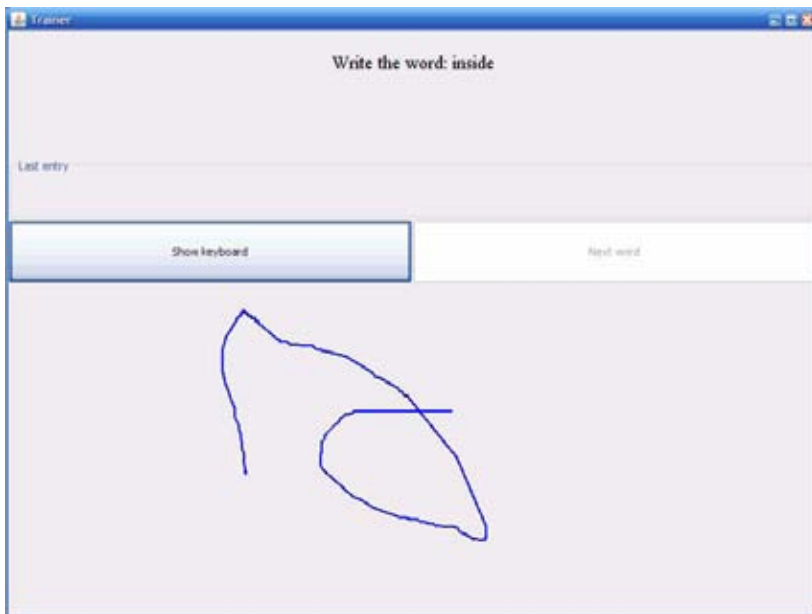


Figure 3.28. The user interface in the practice sessions in Experiment 3.1. The user is writing the word *inside* without the software keyboard as a visual reference.

When the user pressed the SHOW KEYBOARD button the current entry was considered a failed entry. The ideal word shape was presented superimposed on the software keyboard. If the user's last pen trace was available, the pen trace was scaled and translated in relation to the ideal word shape and also shown to the user (Figure 3.29). When the user pressed the NEXT WORD button the software keyboard was automatically hidden and a new word presented.

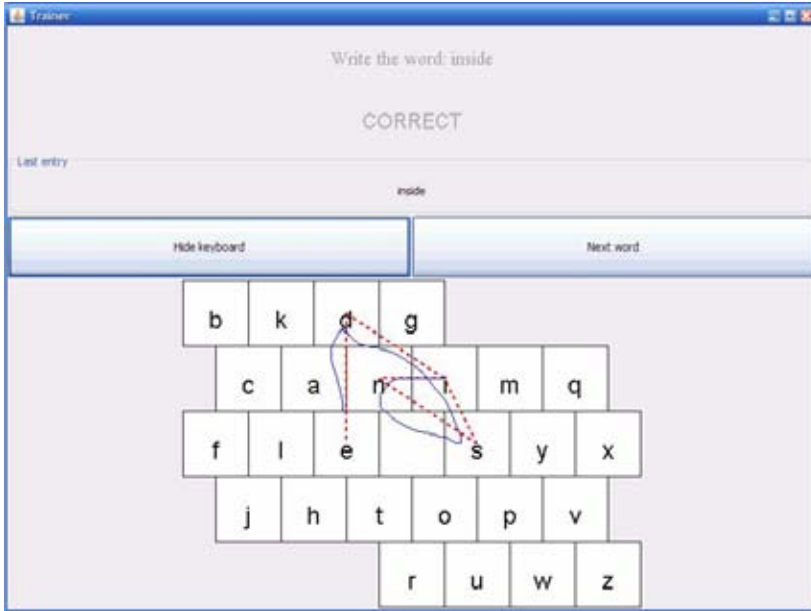


Figure 3.29. The user interface in the practice sessions in Experiment 3.1. The user has pressed the SHOW KEYBOARD button and the user's pen trace (shown as a solid stroke) is displayed along with the ideal trace (shown as a dashed stroke) overlaid on the software keyboard.

In the testing sessions words the participants had already practiced were presented in random order (Figure 3.30). Testing sessions lasted 6 to 20 minutes. The user had two attempts for each word tested. If the user failed two attempts the next word was tested. After all words that had previously been presented to the user during practice had been tested, the testing session ended.



Figure 3.30. The user interface in the testing sessions in Experiment 3.1. The user is in the process of writing the word *inside* without the software keyboard as a visual reference.

## 3.6.2 Results

### 3.6.2.1 Recall Rate

The results showed that given practice all participants were able to write the shapes for all words they practiced. Typically 7-15 repetitions were required. Figure 3.31 plots the number of words participants were able to recall as a function of test session number. Since the pattern-recognizer was in an early prototype stage sometimes participants could enter shapes that were mistakenly misrecognized. Therefore, participants were given a second chance to try to enter the shape of a word again, if the first attempt failed.

Figures 3.31 and 3.32 show that the number of words learned per session (first or second attempt) is a linear function of session number. There were no major individual differences found (see also Figure 3.32). Participants learned 14.7 words on average per session ( $sd = 4.9$ ).

In the last session participants successfully recalled 52.2 word shapes ( $sd = 43$ ,  $min = 39$ ,  $max = 67$ ) on the first attempt and 60.3 ( $sd = 10.8$ ,  $min = 49$ ,  $max = 77$ ) on the second attempt. The error rate was 23.7% ( $sd = 7.7$ ) for the first attempt, and 11.2% ( $sd = 6.3$ ) for the second attempt.

There were no signs of a slower learning rate in later sessions. In fact, in the last session the learning rate improved slightly (see Figure 3.32). If there is a limit to the

number of word shapes a grown-up user can learn, and how high that limit might be is unknown.

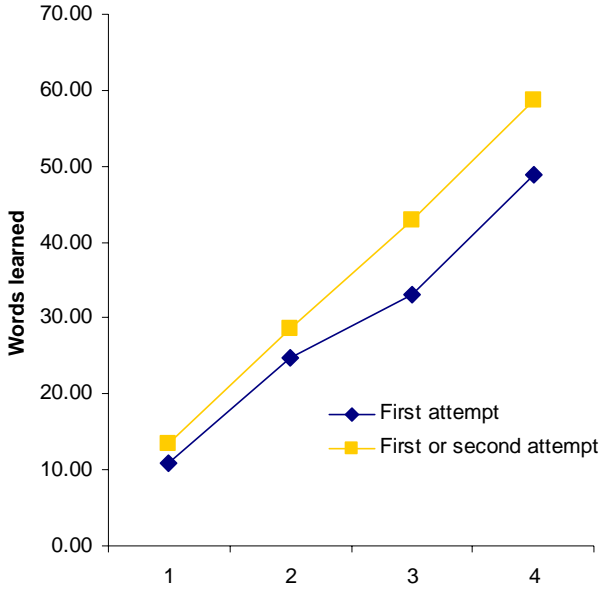


Figure 3.31. Average number of words correctly written as a function of test session number.

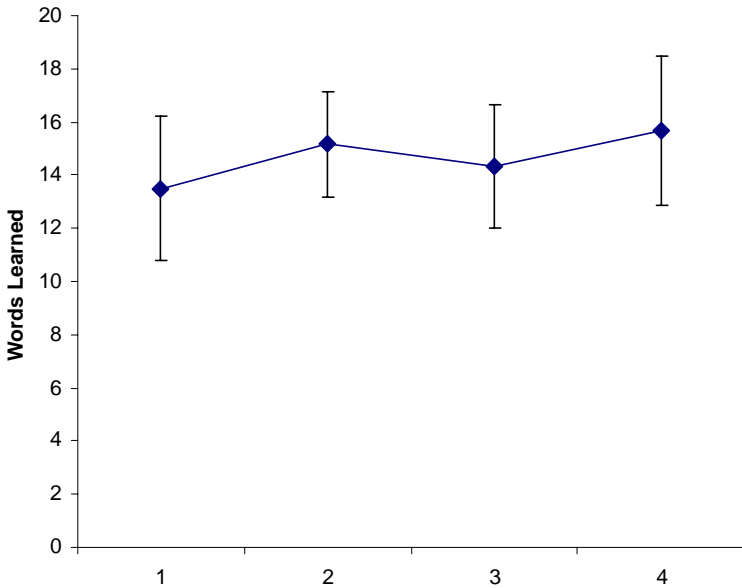


Figure 3.32. Mean and standard deviation of number of words users learned per session as a function of test session number.

### 3.6.2.2 *Response Time*

The average response time (recorded from pen-down to pen-up) was 2430 ms (sd = 2600). Note that response times were expected to be high because participants' focus was set on remembering the shapes for the words. Participants were also not instructed to perform quickly. The word *house* had the fastest response time (mean = 392 ms), and the word *service* the slowest (mean = 19090 ms).

These response times should be interpreted with care. Many participants began a stroke and then stopped in-stroke when trying to recall where to go next. This was especially true for the longer and more complicated pen-gesture shapes of words.

### 3.6.2.3 *Subjective Rating and Open Comments*

After the experiment participants answered a set of questions on a 7-grade Likert scale.

Participants thought the experimental task was medium frustrating (mean = 2.8, sd = 1.4), stimulating to use (mean = 5.3, sd = 0.8), slightly easy to use (mean = 4.2, sd = 1.9), and slightly easy to learn (mean = 4.2, sd = 1.5).

On the question "If such a method is made available for practical use, would you learn it?" with a scale ranging from -3 *Definitely no* to +3 *Definitely yes* participants answered 1.6 on average (sd = 1.2). When asked if they would use it in a list of situations, none of the participant choose *Not at all*,  $\frac{1}{3}$  of the participants answered *Yes, when a physical keyboard is not available* and  $\frac{2}{3}$  of the participants answered *Yes, to replace keyboard typing sometimes*, and none *Yes, to replace keyboard typing all the time*.

Some participants felt that the rehearsal interval multiplier was set too aggressive and as a result words that were correctly entered were rehearsed less frequent than desirable. Other participants liked the rehearsal interval multiplier as it was. This suggests the rehearsal interval multiplier perhaps should be tweaked on an individual basis in a future version.

After each session participants were encouraged to write a brief note with open comments that they might have. Most participants wrote that the entry method was somewhat exciting: "It's fun!", "Kind of fun...", "...more fun than boring".

## 3.6.3 **Discussion**

The experiment showed that users learn on average around 15 word shapes per session. The learning rate showed no sign of slowing down as the sessions progressed. In the end all participants had learned to recall at least 39 word shapes without any visual reference.

Further, participants felt that continuous shape writing was fun.  $\frac{2}{3}$  of the participants even stated at the end of the experiment that if continuous shape writing was made

available for practical use they would want to replace keyboard typing sometimes with it.

In summary, participants could quickly learn the shapes for the continuous shape writing pen-gestures. Since both the shape writing method in itself and the keyboard layout ATOMIK was completely novel to the participants, the results indicate that users indeed have a strong capability to rapidly learn novel pen-gestures with practice.

### **3.7 Experiment 3.2: Immediate Efficacy**

---

The following experiment investigates the initial learning curve for novice users exposed to continuous shape writing.

Text entry proficiency is a function of practice. A user that completely masters continuous shape writing has learned the shapes of thousands of words in motor memory. Zipf's law [Zipf, 1935] predicts that the frequency of a word is approximately inversely proportional to its rank. Zipf's law has the effect that common words are repeated disproportionately often. Therefore it is reasonable to expect users to relatively quickly learn the shapes for the highest ranked words in the language. On the other hand, less ranked words are repeated disproportionately less. From this analysis the expected learning curve for a highly skilled expert continuous shape writing user is most likely very long. On the hand, less common and longer words are usually made of more common fragments whose shapes are likely remembered in a relatively short amount of practice with shape writing. Overall, users will always be somewhere in between a complete novice and a complete expert when writing open text.

This experiment examines users' efficacy with continuous shape writing during the first 40 minutes of use. The goal of the experiment is to verify that continuous shape writing is practical from the start.

Zhai, Sue and Accot [2002] shows that for well-practiced users the ATOMIK layout is much faster for software keyboard typing than QWERTY. In addition, Experiment 3.1 previous in this chapter showed that users could relatively quickly learn the shapes of shape writing pen-gestures on ATOMIK. On the other hand, participants are completely unfamiliar with ATOMIK but most likely accustomed to typing with QWERTY. To gain any insight into whether the layout used is a significant factor to initial performance, both QWERTY and ATOMIK were tested.

To obtain a reference point the thumb keyboard was introduced as a baseline condition. Several other options were considered such as handwriting recognition or Graffiti which are the status-quo text entry methods on pen-computers. However, both handwriting recognition and Graffiti has been shown to be relatively slow (< 15 wpm [Card, Moran and Newell, 1983; Sears and Arora, 2002]). In contrast, although not in the same pen-stroke-based category as shape writing, thumb keyboard was recently

demonstrated to be a highly competitive mobile text entry method where users eventually after several hours of practice reached almost 60 wpm on average [Clarkson, Clawson, Lyons and Starner, 2005].

### 3.7.1 Method

#### 3.7.1.1 Design

The experiment was a within-subjects learning curve experiment with 2 sessions. Each session was split into two sub-sessions with a different condition in each session. In one sub-session participants used continuous shape writing. In the other participants used a thumb keyboard. The starting condition alternated between the sessions. The initial starting condition was balanced across the participants. In the shape writing condition half of the participants were assigned QWERTY and half assigned ATOMIK.

#### 3.7.1.2 Participants

Ten paid volunteers were recruited from the Linköping University campus. Eight were male and two were female. Their ages ranged from 19-35 (mean = 23.1, sd = 4.7). Six participants had previously used a handheld computer before. Eight participants had used a pen-based computer or mobile phone before. No participants had used shape writing before. Four participants had used a thumb keyboard before. The participants were screened for dyslexia and repeated stress injury (RSI). All participants were native Swedish speakers, fluent in English.

#### 3.7.1.3 Apparatus

The shape writing condition used a Fujitsu-Siemens tablet PC. The screen was a 14" TFT LCD touch screen with 24-bit color depth and 1024 × 768 pixel resolution. The screen was set to landscape orientation. The tablet PC was placed on top of a desk during the experiment.

The thumb keyboard condition used a Hewlett-Packard iPAQ h6315 Pocket PC mobile phone. The screen was a 3.5" TFT LCD touch screen with 16-bit color depth and 240 × 320 pixel resolution. The thumb keyboard was connected to the bottom of the device using the serial connector. The thumb keyboard model Hewlett-Packard HSTNH-D01K was designed specifically for this particular mobile phone model and was firmly attached to the device once connected. The device dimensions (excluding thumb keyboard attachment) were 137.64 (height) × 74.6 (width) × 20.8 (depth) mm. The device (excluding thumb keyboard attachment) weighted 190g. Participants were instructed to hold the smart phone with both hands and type using the thumbs.

Both the thumb keyboard and the software keyboard in the continuous shape writing condition used letter keys with a diameter of 8 mm.

See Figure 3.33 for a photograph of the tablet and mobile phone.

The continuous shape writing condition used the pattern recognizer described in Chapter 4. The lexicon contained 15,000 words. The keyboard layout for the shape writing condition was either QWERTY or ATOMIK. It was explained to the participants that the ATOMIK layout was optimized for better performance but might be harder to learn. Participants were then given the choice of which layout they wanted to use. The exception to this rule was the two last participants who were involuntarily assigned QWERTY layout in order to balance the number of participants using either QWERTY and ATOMIK (five participants per layout). The two participants that did not get to choose layout were not informed until after the experiment that an alternative layout for shape writing existed.



Figure 3.33. The tablet computer and mobile phone used in Experiment 3.2 and 3.3. The thumb keyboard is connected to the mobile phone.

#### 3.7.1.4 Material

Since the goal of this experiment was to study the initial performance in the first 40 minutes of “natural” use, the text used in the study was meant to simulate naturally occurring common phrases. The phrase set released by MacKenzie and Soukoreff [2003] was used as material in the study. Some words in the phrase set were misspelled and these were corrected. The phrase set consists of 500 phrases in U.S. English. The phrase set has been criticized for containing many “trigger words” that confuse persons suffering of dyslexia [Kano, 2005]. To eliminate this confound



participants were screened for dyslexia, as described earlier. The order of the phrases was scrambled for each participant and condition to avoid a skill transfer confound across the conditions.

### 3.7.1.5 Procedure

Each sub-session lasted 20 minutes. The experiment software presented a single phrase on top of the screen. The participant was asked to quickly and accurately write the phrase displayed. The participants were also instructed to correct errors as they discovered them. The experiment software interfaces for the shape writing and thumb keyboard conditions are shown in Figures 3.34 and 3.35 respectively.

In both the shape writing condition and the thumb keyboard condition a phrase was considered entered when the participant pressed the ENTER key. The initial time stamp was measured when the participant entered a key in the thumb keyboard condition, or when the participant pressed down the pen in the shape writing condition.



Figure 3.34. The experiment software interface in the shape writing (QWERTY) condition (running on a computer tablet). Participants were encouraged to move the software keyboard to a comfortable position.



Figure 3.35. The experimental software running in the thumb keyboard condition on a mobile phone.

To avoid fatigue a 30 second break was automatically enforced by the software every 5 minutes. Break scheduling was designed to not interfere with the writing task and therefore only occurred after the current phrase was completed. During a break the software displayed a countdown informing the user when the session would be resumed.

## 3.7.2 Results

### 3.7.2.1 Error Rate

There are two primary sources when measuring error rate. The first is uncorrected errors which are the errors participants never discovered or bothered to correct in their text. The second is the number of corrected errors where the user discovered and corrected an error. Because participants' correction of errors reduces entry rate, the number of uncorrected errors is the source that matters when measuring error rate in relation to entry rate. The corrected errors are used when the editing process is analyzed. Below the unqualified term error rate refers to the number of uncorrected errors only.

Overall, error rate was low and did not depend on either condition or keyboard layout. The average error rate for both sessions was 0.9% (sd = 0.6) for shape writing and 1.1% (sd = 0.6) for thumb keyboard. Repeated measures analysis of variance showed that the difference between the two conditions was not significant ( $F_{1, 9} = .494, p = .5$ ).

Within the shape writing condition, the average error rate for both sessions was 1.1% (sd = 0.2) for QWERTY and 0.8% (sd = 0.9) for ATOMIK. Variance analysis (between subjects) showed that the difference was not significant ( $F_{1,9} = .474$   $p = .511$ ).

The fact that the error rate for shape writing is very low is highly encouraging.

It is also interesting to examine the amount corrected errors made by participants. Note however that this measurement of error rate is not related to entry rate because the time taken for error correction is included in the entry rate figure and cannot be separated out. In practice, error correction is a task integrated in all text entry methods and cannot be completely avoided.

In the shape writing condition the participants corrected 5.1% (sd = 8.6) of the words on average.

A more detailed analysis of the corrected errors in the shape writing condition reveals that the uncorrected average error rate was 5.6% (sd = 8.8) during the first 20 minutes, and 4.6% (sd = 8.4) during the last 20 minutes. One participant in particular had an error rate that was more than 2.5 standard deviations away from the mean and contributed disproportionately to the gross error rate. From inspection of the log file it appears this participant was “trying out” the system while in-test in the beginning of each session thus inflating error rate. Excluding this participant, the uncorrected average error rate was 3.2% during the first 20 minutes, and 2.2% during the last 20 minutes. During the first 20 minutes three participants had zero recognition errors, and during the last 20 minutes six participants had zero recognition errors.

With thumb keyboard participants corrected 4.7% (sd = 3.1) of the characters on average. Because shape writing and thumb keyboard operates on different levels – characters vs. words and have vastly different correction mechanisms it is inappropriate to compare the two methods’ corrected errors against each other directly. However, an analysis of the number of words that were corrected (with the BACKSPACE key) in the thumb keyboard condition reveals that participants corrected on average 24.8% of all words (actual words) in the thumb keyboard condition.

From this analysis it appears shape writing results in much less word errors than thumb keyboard (5.1% [including all participants] vs. 24.8%). On the other hand, errors in shape writing are more severe because they result in a different word (although often most characters in the incorrect word are still correct in relation to the user’s intended word), while thumb keyboard errors most of time results in a misspelled word where only a single character is wrong. Because of the difficulty in performing a direct comparison in corrected errors on either the character or word level, statistical significance tests were not performed on these errors.

### 3.7.2.2 *Entry Rate*

The grand average entry rate of all trials in both sessions was 16.6 wpm ( $sd = 6.5$ ) for shape writing and 27.7 wpm ( $sd = 3.9$ ) for thumb keyboard. Repeated measures analysis of variance showed that the difference is significant ( $F_{1,9} = 21.788, p < .01$ ). Note, however, that the shape writing condition lumps together both the QWERTY and ATOMIK sub-conditions.

Not surprisingly, entry rate performance varied considerably between QWERTY and ATOMIK shape writing. The average entry rate averaged over both sessions was 20.9 wpm ( $sd = 5.6$ ) for QWERTY and 12.3 wpm ( $sd = 4.1$ ) for ATOMIK. Variance analysis (between subjects) showed that the difference was significant ( $F_{1,9} = 7.563, p < .05$ ). Users were significantly faster when using QWERTY in the initial 40 minutes of use.

Repeated measures analysis of variance showed that even with QWERTY, continuous shape writing was significantly slower than thumb keyboard (20.9 ( $sd = 5.6$ ) vs. 25.8 ( $sd = 3.4$ ) wpm;  $F_{1,4} = 7.829, p < .05$ ). This result was expected given 1) users' brief exposure to continuous shape writing, and 2) users' skill transfer from desktop to thumb keyboard typing.

Figure 3.36 plots the initial curve for the two sessions (recall that an individual session lasted 20 minutes). As shown in Figure 3.36 the entry rate (averaged over all participants) was higher for thumb keyboard across all 5-minute intervals. It is also evident that continuous shape writing was faster with QWERTY than ATOMIK. From Figure 3.36 is also evident that shape writing with QWERTY is immediately practical. After 5 minutes of practice the average entry rate is around 15 wpm. After 20 minutes of practice the average text entry rate is beyond 20 wpm. As a reference point, Wobbrock, Chau and Myers [2007] found that participants writing with the common T9 predictive text entry method never surpassed 16 wpm, not even after 15 20-minute sessions.

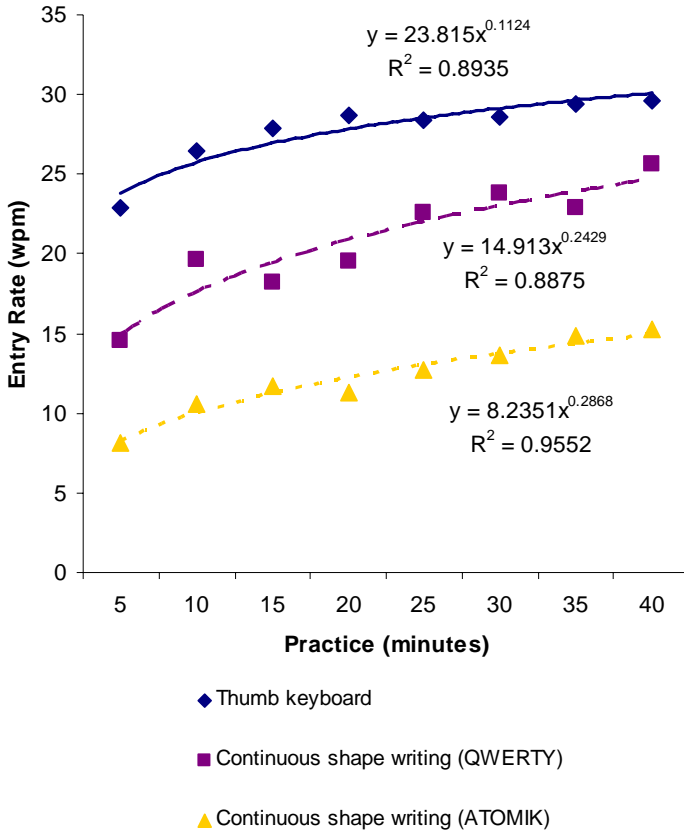


Figure 3.36. Average entry rate (wpm) as a function of practice (minutes). Power law regression curves are superimposed for reference [Crossman, 1959].

There were considerable individual variations. Figure 3.37 shows the entry rate (wpm) distribution among participants for both conditions during the last 10 minutes in the experiment. As is evident in Figure 3.37, the spread is much wider in the continuous shape writing condition than in the thumb keyboard condition. In terms of entry rate, the continuous shape writing condition had simultaneously both the worst (entry rate < 9 wpm) and best (entry rate > 40 wpm) performers. The thumb keyboard condition resulted in a narrower distribution centering at an entry rate around 29 wpm (see also Figure 3.37).

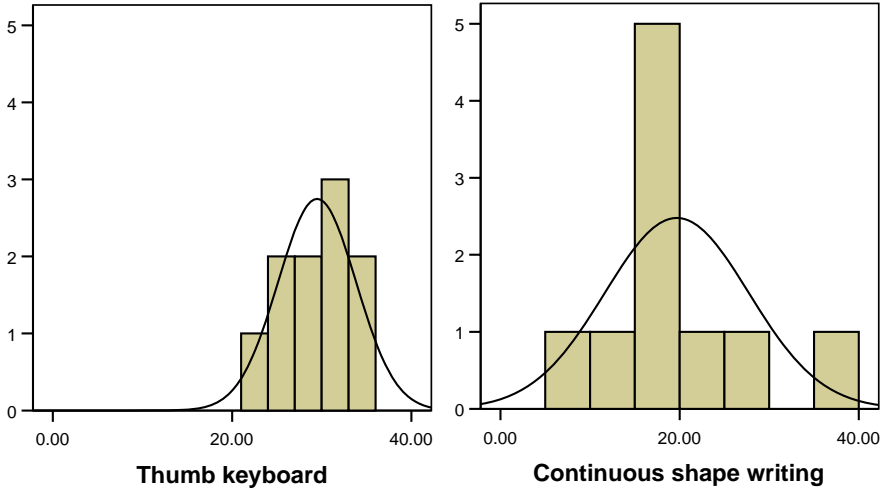


Figure 3.37. Entry rate (wpm) histograms of the participants in both conditions for the last 10 minute interval in the experiment. Note that participants that used QWERTY and ATOMIK layouts are lumped together in the left plot.

Figure 3.38 plots the learning curve for the top performers in both the continuous shape writing and thumb keyboard conditions. Note that these were different individuals. The fastest participant with thumb keyboard appears to have quickly saturated at around 35 wpm. In contrast, the fastest participant with continuous shape writing (QWERTY) continuously improved (except at the 30-35 minute interval when performance was near constant), and exceeded the fastest thumb keyboard typist at the last 5-minute interval.

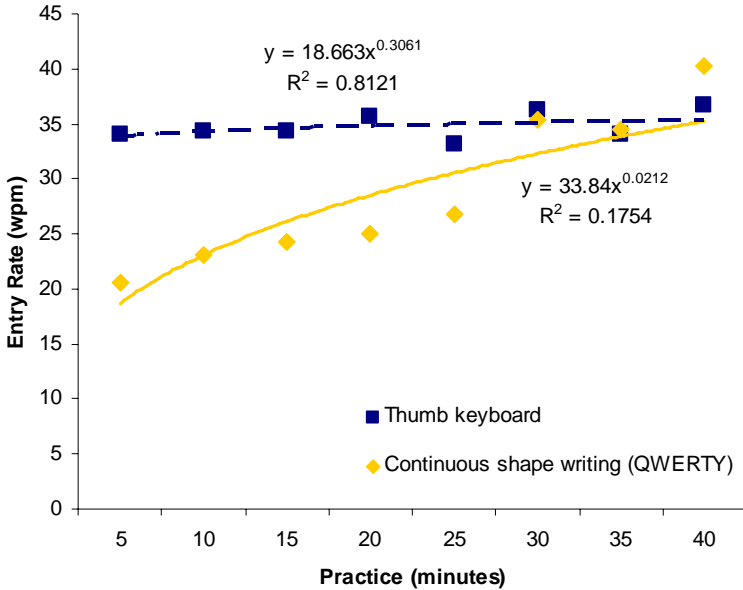


Figure 3.38. Entry rate (wpm) as a function of practice (minutes) for the participants with the highest average entry rate in the shape writing (QWERTY) and thumb keyboard conditions respectively. As a reference point, power learning curves [Crossman, 1959] are superimposed for both conditions.

### 3.7.2.3 Subjective Ratings and Open Comments

After the experiment participants answered a set of questions on a 7-grade Likert scale. Friedman's repeated measures non-parametric test was used to determine significances.

An overview of the subjective ratings is shown in Figure 3.39. Participants liked writing with shape writing (mean = 5.2, sd = 1.0) more than the thumb keyboard (mean = 4.3, sd = 1.0), but the result was not significant ( $\chi^2 = 2.7$ , df = 1,  $p = .102$ ). Participants felt thumb keyboard was more physically demanding (mean = 4.5, sd = 1.6) than shape writing (mean = 3.6, sd = 1.2), but the result was not significant ( $\chi^2 = .4$ , df = 1,  $p = .527$ ). Participants significantly ( $\chi^2 = 9.0$ , df = 1,  $p < .01$ ) looked more on the software keyboard (mean = 5.6, sd = 0.8) than the thumb keyboard (mean = 4.1, sd = 1.5). Last, participants significantly ( $\chi^2 = 5.45$ , df = 1,  $p < .05$ ) felt writing with shape writing was more fun (mean = 5.4, sd = 0.8) than thumb keyboard (mean = 3.7, sd = 1.4).

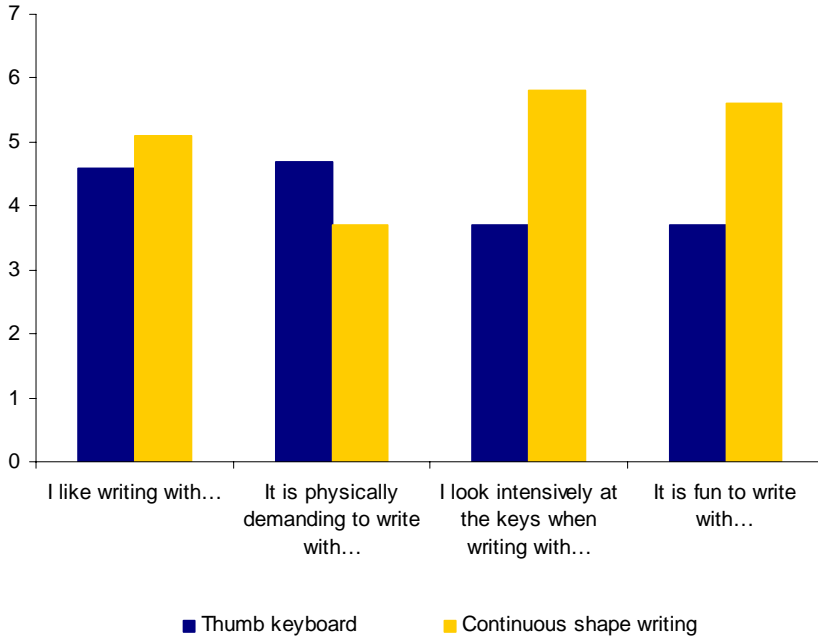


Figure 3.39. Participants' subjective ratings in Experiment 3.2.

On the statement “The geometric trace of the words come to my mind when writing with [condition]”, with 5 possible responses: { 1 = *never*, 2 = *rarely*, 3 = *sometimes*, 4 = *often*, 5 = *always* }, participants answered that it occurred to them more often with continuous shape writing (mean = 3.3, sd = 0.9) than for thumb keyboard (mean = 2.7, sd = 1.3), but the result was not significant ( $\chi^2 = 2.0$ , df = 1,  $p = .157$ ).

### 3.7.3 Discussion

The results show that after 35 minutes of practice users can reach over 25 wpm with continuous shape writing (QWERTY). Layout does matter. QWERTY is faster to learn than ATOMIK. No participant that used ATOMIK was able to reach an entry rate faster than 20 wpm. On average, thumb keyboard was faster than continuous shape writing in this initial stage of use. An interesting exception is that the only participant that reached an entry rate > 40 wpm used continuous shape writing (QWERTY). This suggests that the ATOMIK layout may be only attractive to advanced users who are willing to invest effort and sacrifice short-term performance to learn a new layout that eventually might pay off later. On the other hand, even though the participants dealt with a double learning task (shape writing as a method of entry and a completely new keyboard layout), their average performance after five minutes of learning was still comparable to character-based hand writing methods such as Jot where participants typically achieves < 10 wpm after a few minutes of practice [Sears and Arora, 2001; Költringer and Grechenig, 2004]. A surprising anecdotal result is that among the first eight participants, five chose ATOMIK over QWERTY.



In comparison to thumb keyboard, participants felt shape writing was more fun to write with than thumb keyboard. The majority of the participants (90%) stated that physical discomfort with thumb keyboard and the small keys was a problem in the open comments section of the questionnaire handed out after the experiment. Some participants (20%) reported discomfort when using continuous shape writing because they had to lean over a table to write. Actual use of shape writing on a mobile phone would alleviate or remove this problem.

Participants self-reported that they found themselves looking at the software keyboard more than the thumb keyboard. 40% of the participants said their hand tended to obscure the keys when writing with shape writing. There was no apparent relation between keyboard layout used and this remark.

Clarkson, Clawson, Lyons and Starner [2005] reports on a thumb keyboard experiment where participants reached on average 31.7 wpm during the first 20 minutes. In comparison, in this experiment participants reached 26.wpm during the first 20 minutes. Clarkson et al. [2005] reports an average 6% “total error rate” [Soukoreff and MacKenzie, 2003b] during the first 20 minutes of use. The total error rate metric lumps together both corrected and uncorrected errors and it is therefore not possible to conclude whether error rates obtained in this experiment were similar to those in Clarkson et al. [2005].

Since there is an inherent speed-accuracy tradeoff present in text entry, it is possible that participants in the study reported by Clarkson et al. [2005] did not attempt to correct as many errors. Because only “total error rate” is reported, it is also possible participants did correct many errors and were simply faster. Other possible explanations for the entry rate difference include random variation, and the facts that participants in this experiment were older and not native English speakers.

### **3.8 Experiment 3.3: Accelerated Novice Performance**

---

Experiment 3.2 showed that users can reach over 25 wpm after 35 minutes of practice when shape writing with the QWERTY layout. Further, an individual user can reach over 40 wpm after 35 minutes of practice. It is well known in the literature that text entry performance is skill-based and high performance depends on extended practice.

It is plausible that shape writing as a novel skill, in particular in combination with a new optimized layout, requires large amount of practice for the user to truly excel. In comparison, a physical keyboard-based method benefits from skill transfer from the desktop keyboard, which even though it takes years of practice to be proficient with, is a skill typically already learned by computer users.

To gain a glimpse of what level of performance is possible had the user gained substantial amount of practice, an “accelerated learning” experiment was designed. This experiment accelerates novice performance for a single phrase by letting

participants repeatedly practice writing individual words. Thereafter entry rates achieved by participants repeatedly writing the phrase are measured. The goal of the experiment is to quickly find the asymptotic novice user entry rate performance.

### 3.8.1 Method

#### 3.8.1.1 Design

The experiment was within-subjects and single-session. The session was split into two sub-sessions with a different condition in each session: one using shape writing and the other using thumb keyboard. The starting condition was balanced across the participants. In the shape writing condition half of the participants were assigned QWERTY and half assigned ATOMIK.

#### 3.8.1.2 Participants

Ten paid volunteers were recruited from the Linköping University campus. Nine were male and one was female. Their ages ranged from 21-35 (mean = 24.1, sd = 4.1). Seven participants had participated in Experiment 3.2. Seven participants had previously used a handheld computer before. Eight participants had used a pen-based computer or mobile phone before. Eight participants had used a thumb keyboard before. The participants were screened for dyslexia and repeated stress injury (RSI). All participants were native Swedish speakers, fluent in English.

#### 3.8.1.3 Apparatus

The apparatus used was identical to Experiment 3.2.

#### 3.8.1.4 Material

Five phrases were randomly selected from the Enron email corpus [Klimt and Yang, 2001]. The phrases were selected based on the condition that at least half of the words in the phrases were among the top-100 ranked words in the American National Corpus (ANC). This procedure was designed to minimally ensure the phrases model typical U.S. English. The phrases are shown in Table 3.6.

Table 3.6. The phrases used in Experiment 3.3 and their average word rank in U.S. English in a large corpus.

Phrase	Average Rank
Thanks for taking care of this	246.6
You forgot to attach the request	2351.2
Does anyone else have any comments	269.3
Look forward to seeing you soon	269
My hands are full	196.5

### 3.8.1.5 Procedure

Each sub-session lasted a maximum of 25 minutes. Each sub-session had two phases.

The first sub-session was a practice phase where an isolated word from the test phrase was displayed and practiced (see Figures 3.40 and 3.41). Participants were instructed to write the word as quickly as possible and refrain from correct their input. In the continuous shape writing condition a word entry was considered inputted when the user lifted up the pen. In the thumb keyboard condition users had to press the ENTER key to signal end-of-word. No measurements were recorded in the practice session.



Figure 3.40. The experimental practice session interface in the shape writing condition.

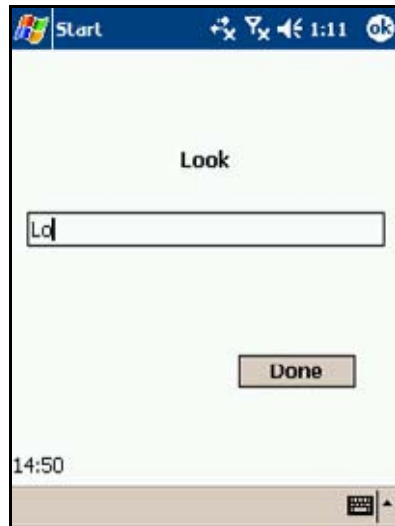


Figure 3.41. The experimental practice session interface in the thumb keyboard condition.

The second sub-session was a test phase where the entire phrase was displayed and entered (see Figures 3.42 and 3.43). A display showed the entry and error rate for the last entry to the left, and the best entry and error rate the participant had achieved so far to the right. The best entry rate was only updated if the error rate was  $< 10\%$ . Otherwise, the display showed the last error rate in red as an indication to the participant that the error rate was too high to be accepted.

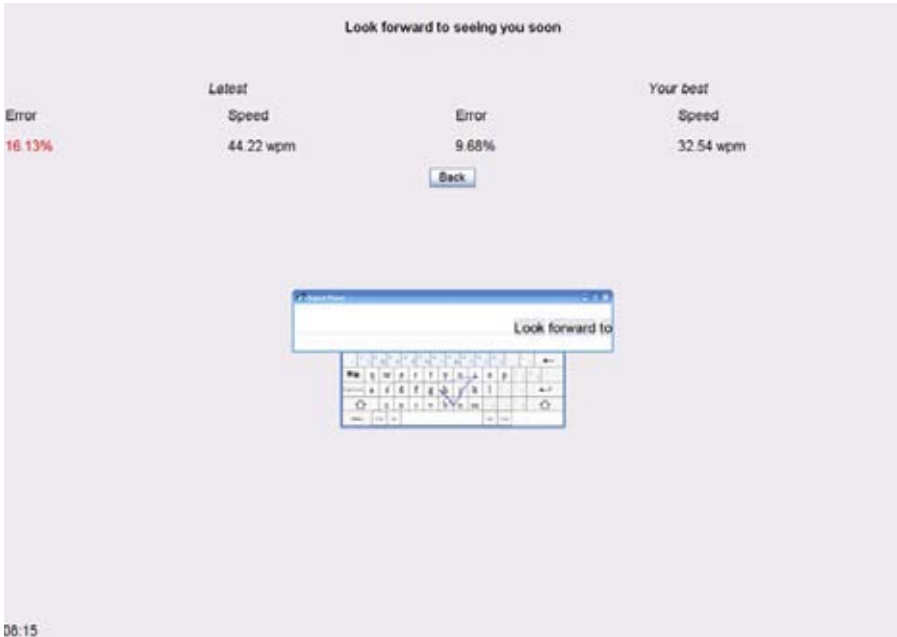


Figure 3.42. The experimental testing session interface in the shape writing condition.



Figure 3.43. The experimental testing session interface in the thumb keyboard condition.

In both the shape writing condition and the thumb keyboard condition a phrase was considered entered when the participant pressed the ENTER key. The initial time stamp

was measured when the participant entered a key in the thumb keyboard condition, or when the participant pressed down the pen in the shape writing condition.

Participants were instructed to rest whenever they wanted to after a phrase had been entered.

### 3.8.2 Results

#### 3.8.2.1 *Entry Rate*

The maximum entry rate is in the following discussion defined as the maximum entry rate achieved by an individual participant.

The average entry rate for all entries with no errors was 45.8 wpm for continuous shape writing and 46.4 wpm for thumb keyboard. Repeated measures analysis of variance showed that the difference was not significant ( $F_{1,9} = .035$ ,  $p = .855$ ). The performance distributions are shown in Figure 3.44. As is evident in Figure 3.44 individual performance varied considerably. Particularly, in the shape writing condition it is apparent that some participants learned and could write text using shape writing more effectively than others. The average maximum entry rate with no errors was 57.5 wpm ( $sd = 19.5$ ) for shape writing and 59.1 wpm ( $sd = 13.3$ ) for thumb keyboard. Repeated measures analysis of variance showed that the difference was not significant ( $F_{1,9} = .152$ ,  $p = .705$ ).

The particular layout used in the shape writing condition had no effect. The average entry rate with no errors was 46.5 wpm ( $sd = 8.1$ ) for QWERTY and 45.1 wpm ( $sd = 20$ ) for ATOMIK. Analysis of variance (between subjects) showed that the difference was not significant ( $F_{1,9} = .024$ ,  $p = .881$ ). The average maximum entry rate with no errors was 56.4 wpm ( $sd = 8.6$ ) for QWERTY and 58.6 wpm ( $sd = 28$ ) for ATOMIK. Analysis of variance (between subjects) showed that the difference was not significant ( $F_{1,9} = .028$ ,  $p = .871$ ).

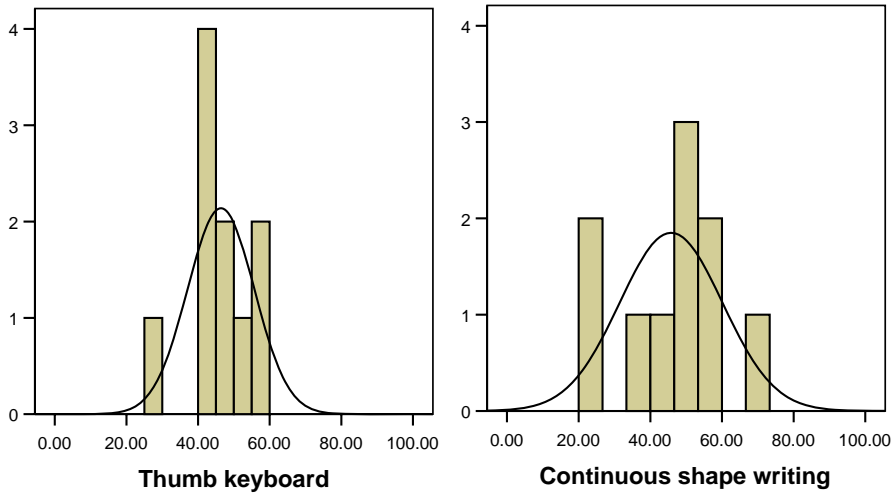


Figure 3.44. Entry rate frequency distribution among participants.  
Error rate = 0%.

Large individual differences were expected due to the short sessions exposed to participants. Recall that only a maximum of 15 minutes of practice was allowed and the test only lasted for 10 minutes. Figure 3.45 shows the average entry rate obtained by each participant for both conditions as a function of participant number, where participants are ranked by the sum of their average entry rate in both the shape writing and thumb keyboard conditions. Figure 3.46 shows the maximum entry rate obtained by each participant for both conditions as a function of participant number, where participants are ranked by the sum of their maximum entry rate in both the shape writing and thumb keyboard conditions.

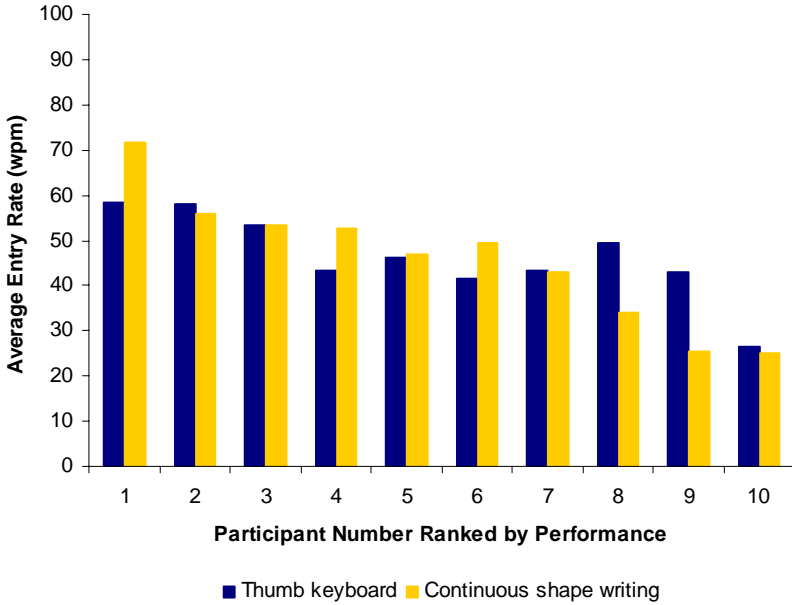


Figure 3.45. Average entry rate (wpm) for both conditions as a function of participant number, ranked by performance. Error rate = 0%.

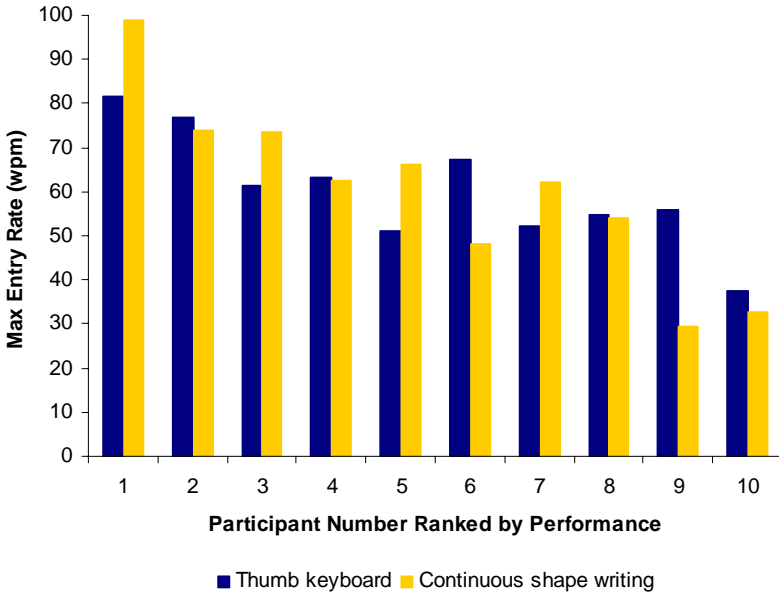


Figure 3.46. Maximum entry rate (wpm) for both conditions as a function of participant number, ranked by performance. Error rate = 0%.



### 3.8.2.2 Subjective Ratings and Open Comments

After the experiment participants answered a set of questions on a 7-grade Likert scale. Friedman's repeated measures non-parametric test was used to determine significances.

An overview of the subjective ratings is shown in Figure 3.47. Participants significantly ( $\chi^2 = 4.5$ ,  $df = 1$ ,  $p < .05$ ) liked writing with continuous shape writing (mean = 5,  $sd = 1.6$ ) more than the thumb keyboard (mean = 3.8,  $sd = 1.8$ ). They also felt thumb keyboard was physically more demanding (mean = 4.8,  $sd = 1.8$ ) than continuous shape writing (mean = 3.4,  $sd = 1.2$ ), but the result was not significant ( $\chi^2 = 2.778$ ,  $df = 1$ ,  $p = .096$ ). Participants stated they looked slightly less on the thumb keyboard (mean = 5.2,  $sd = 1.6$ ) than the software keyboard (mean = 5.9,  $sd = 1.0$ ), but the result was not significant ( $\chi^2 = .667$ ,  $df = 1$ ,  $p = .414$ ). Last, participants stated that they significantly ( $\chi^2 = 6.4$ ,  $df = 1$ ,  $p < .05$ ) felt continuous shape writing was more fun to write with (mean = 5.3,  $sd = 1.2$ ) than thumb keyboard (mean = 3.7,  $sd = 1.6$ ).

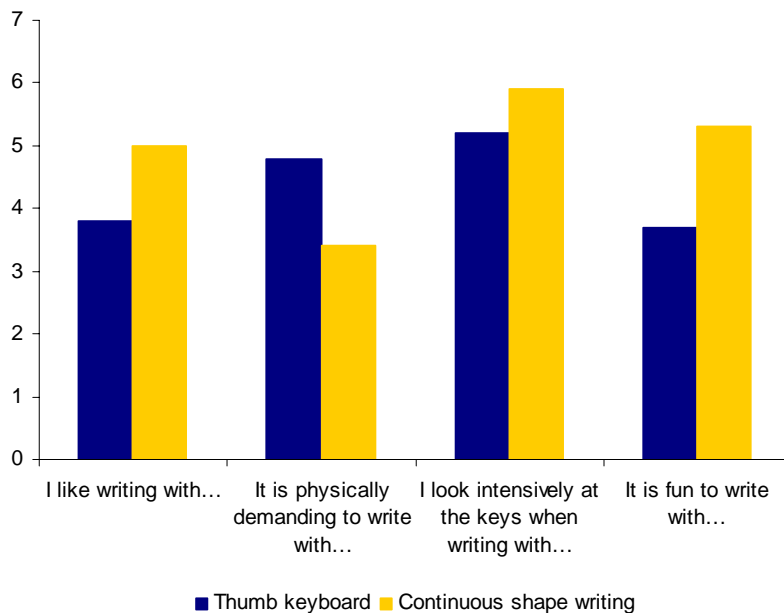


Figure 3.47. Participants' subjective ratings in Experiment 3.1.

On the statement "The geometric trace of the words come to my mind when writing with [condition]", with 5 possible responses: { 1 = *never*, 2 = *rarely*, 3 = *sometimes*, 4 = *often*, 5 = *always* }, participants significantly ( $\chi^2 = 5.45$ ,  $df = 1$ ,  $p < .05$ ) answered that it occurred to them more often with continuous shape writing (mean = 4.1,  $sd = 0.7$ ) than for thumb keyboard (mean = 2.8,  $sd = 1.3$ ).

In the open comments section, 2 participants wrote that it was annoying that the hand obstructed the keys when shape writing. A single participant stated that the thumb keyboard was very physically demanding. Another participant wrote that the thumb keyboard had too small keys to be comfortable.

### 3.8.3 Discussion

On average participants had entry rates around 46 wpm and error rates around 1% using both shape writing and thumb keyboard. As in Experiment 3.2 the spread in entry rates was larger in the continuous shape writing condition than in the thumb keyboard condition.

Recent research has shown that the motor memories of newly practiced skills do not stabilize until after 5 hours of practice [Shadmehr and Holcomb, 1997]. Considering that the test session followed immediately after the practice session an alternative experimental setup where the test session was delayed may have lead to different results. Thumb keyboard is less likely to have suffered from this effect because of the strong skill transfer from desktop keyboard typing.

In relation to previous reported empirical studies of the baseline thumb keyboard condition, Clarkson et al. [2005] reports on a thumb keyboard experiment where participants reached on average 60.3 wpm after 20 20-minute sessions. In comparison, in this experiment participants only reached 46.4 wpm with the thumb keyboard. The different results are remarkable because in Clarkson et al. [2005] participants wrote open text, while in this experiment participants repeated a well-practiced phrase. Therefore it is reasonable to believe that entry rates in this experiment would have been higher or at least at a comparable level as in Clarkson et al. [2005]. One explanation could be the high (> 8%) average “total error rate” reported in Clarkson et al. [2005]. Perhaps participants in the experiment [Clarkson et al., 2005] never were able to type 60 wpm with no errors on any phrases at all. Another explanation is that participants in this experiment never saturated their thumb keyboard performance and could do better given more practice. Other possible explanations for the entry rate difference include random variation, and the facts that participants in this experiment were older and not native English speakers.

An open question is whether participants saturated their performance. Because the software keyboard was always present as a visual reference it cannot be excluded that participants’ articulations were primarily visual-guided rather than primarily recall driven. If this is the case, the participants never completely saturated learning and transitioned into fast open-loop recall from motor memory. This explanation appears likely given the evidence that the fastest participant with ATOMIK was two times faster than the two slowest participants using the same layout. Because ATOMIK is unfamiliar to the participants, this layout is particularly sensitive to participants who never saturate their learning. Participants who never saturate rely on visual search of

the keys which will be particularly slow when the layout is previously unknown to them.

A future experimental setup should involve several sessions of practice and testing that focuses on learning and open-loop recall similar to the experimental method in Experiment 3.1. These sessions should be separated by at least a day to allow proper motor memory consolidation. After such sessions one or two test sessions should be initiated where the software keyboard is hidden to force recall from motor memory. While arguably artificial, such an experimental setup is more likely to reveal the true expert entry rate limit for shape writing.

### 3.9 Summary

---

This chapter has presented the concept and design rationales of continuous shape writing as a novel mobile text entry method. In addition critical user interface components and localization issues have been discussed.

Experiment 3.1 showed that users aided by an expanding rehearsal algorithm learn the shape writing word shapes relative fast. On average users learned 15 word shapes per 40 minute training session. The results indicated that it is possible to completely memorize the shape writing gestures for all words tested by all participants, with a relatively low number of repetitions (typically 7-15). The results to some extent justified one of the basic principles of shape writing – the use of continuous geometric patterns as representations of words.

Experiment 3.2 showed the initial 40-minute learning curve for continuous shape writing in comparison to thumb keyboard as the baseline condition. Probably due to skill transfer from desktop typing, participants could achieve on average 29.6 wpm with thumb keyboard after 35 minutes of practice. For shape writing users initial performance depended on the keyboard layout used. Participants were significantly faster with QWERTY than ATOMIK. Using QWERTY, participants achieved an average entry rate at 25.6 wpm after 35 minutes of practice. Using ATOMIK, participants achieved an average entry rate at 15.3 wpm after 35 minutes of practice. Average error rates were very low in both conditions, < 1% for continuous shape writing, and 1.1% for thumb keyboard. The experiment showed that although novel to the users, continuous shape writing using QWERTY was a practical text entry method from the start (average entry rate was 19.7 wpm after five minutes of practice) without much initial learning requirement. With a new and optimized layout such as ATOMIK, the results suggest that certain amount of practice is necessary before a practical speed (e.g. > 15 wpm) can be obtained.

Experiment 3.3 showed that on average participants reached entry rates higher than 45 wpm for both continuous shape writing and thumb keyboard. The average and maximum entry rate distribution among participants had a higher spread for

continuous shape writing than for thumb keyboard. Some participants using continuous shape writing were much faster than others. Three participants using continuous shape writing (ATOMIK) were able to write the assigned test phrase in over 70 wpm with no errors at least once. One participant using continuous shape writing (ATOMIK) reached a maximum text entry rate of 99 wpm with no errors. This was the highest entry rate recorded in the experiment.

In summary, for pen-based interfaces, continuous shape writing is a competitive technology. Novice users have low error rates  $< 1\%$  and reaches an average entry rate of 25 wpm after 35 minutes of practice. An advanced user can reach over 40 wpm after 35 minutes of practice. Expert users can potentially reach over 70 wpm for well-practiced words and phrases. There is empirical evidence that when using continuous shape writing on ATOMIK a completely saturated phrase can be inputted at an entry rate over 98 wpm with no errors.

In addition users liked writing with continuous shape writing more than writing with the thumb keyboard, and also felt continuous shape writing was a more fun text entry method to use.

# Chapter 4

## Recognizing Continuous Shape Writing<sup>1</sup>

---

This chapter provides the technical details necessary to implement an effective continuous shape writing recognizer.

### 4.1 Introduction

---

Pen-gesture and handwriting recognition belongs to the general field of pattern recognition. Duda, Hart and Stork [2001] divides the pattern recognition process into five major components: sensing, segmentation, feature extraction, classification and post-processing (Figure 4.1).

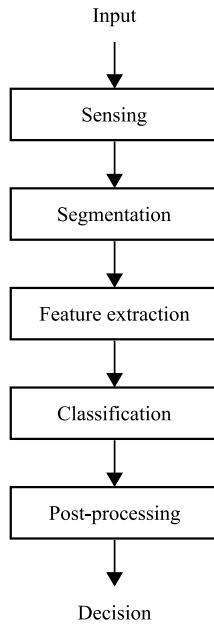


Figure 4.1. The general process workflow of a pattern recognition system [Duda et al., 2001].

The sensors convert outside stimuli into a machine readable format, the segmentation component divides the input into recognizable chunks, the feature extractor extracts the relevant features from the input, the classifier finds the best matching class for the

---

<sup>1</sup> Parts of section 4.2: Pen-Gesture Recognition are from Kristensson [2004]. Subsection 4.3.1: Multi-Channel Recognition consists of revised parts from Kristensson and Zhai [2004].

input, and the post-processing unit performs additional processing such as for instance error risk analysis or integration of the results from multiple classifiers [Duda, Hart and Stork, 2001].

In pen-gesture recognition the input to the recognizer is usually an ordered series of time stamped two-dimensional quantized points on a pixel grid (e.g. touch screen, monitor display) typically sensed from pen, finger or mouse movement. Pen-gestures are naturally segmented because the system receives a signal when pen-up, finger lift or mouse button release occurs. Feature extraction, classification and post-processing depend on the pen-gesture recognizer method used. Traditional approaches in pen-gesture recognition have relied on region encoding [Diamond, 1957; Teitelman, 1964; Newman and Sproull, 1979], special recognition “programs” [Lipscomb, 1987], feature matching [Goldberg, 1997], trainable linear machines [Duda and Hart, 1973; Rubine, 1991] or elastic matching [Burr, 1981; Tappert, 1982].

## 4.2 Pen-Gesture Recognition

---

This section lists the most closely related work to pen-gesture recognition in the literature. The literature is in fact surprisingly sparse on pen-gesture recognition and mainly focused on on-line and off-line handwriting recognition. On the other hand, handwriting recognition has benefited from tremendous research efforts in the last two decades [Tappert, Suen and Wakahara, 1990; Plamondon and Srihari, 2000; Nafiz and Yarman-Vural, 2001; Koerich, Sabourin and Suen, 2003].

### 4.2.1 Region Encoding

Originally region encoding was originally developed to recognize isolated numerals [Diamond, 1957] and characters [Teitelman, 1964]. Diamond [1957] presents a hardware implementation for recognizing isolated numerals. The user slides a pen connected to a source of potential across a series of line crossings. The line crossings are thin conductors that become energized by the pen and thereby activate latches. Numerals are then defined as one or more pre-defined unique series of crossed lines. The system described by Diamond [1957] is template-based. Each template is encoded as a series of regions that are visited in order by the pen. This is the central idea behind region encoding.

In the first trainable pen-gesture recognizer [Teitelman, 1964] each pen-gesture in the lexicon is overlaid on a  $n \times m$  grid. The grid is typically  $3 \times 3$  but other grid resolutions are possible. In the implementation by Teitelman [1964] the regions need not be rectangular, and can in fact be arbitrarily defined by the user if new regions are required to discriminate among new pen-gestures that are added later to the template set. The idea in Teitelman [1964] is that the user trains the system by entering all desired pen-gestures (for instance alphanumeric characters) into the system and creates new regions as they are required. The pen-gesture templates are stored as a sequence of regions visited in a binary tree. When the system is trained, an arbitrary

pen-gesture articulation is encoded into a series of regions visited. The binary tree is then searched for the pen-gesture with the closest matching region encoding.

In a later implementation the regions are fixed and defined in such a matter that each template pen-gesture only requires 16-bit storage [Newman and Sproull, 1979].

Region encoding can be effective if only a small set of pen-gestures are used and the definition of the pen-gestures is arbitrary. Region encoding is invariant to scale and translation differences between the input and template pattern. Region encoding is also computationally efficient and easy to implement. For these reasons it is a popular method for various “mouse gesture” software packages. For example, the mouse gesture extension to the Mozilla Firefox web browser uses a variant of region encoding<sup>2</sup>.

There are primarily two downsides to region encoding. First, the coarse division of an input pattern into regions constrains the expressiveness and size of the vocabulary of template patterns considerably. Second, region encoding is intolerant to input patterns that break the region sequence. The decision step that assigns a region to part of an input pattern is binary and a small deviation in the input pattern causes a region to become misinterpreted and thereby results in recognition failure.

#### 4.2.2 Feature Matching

A less limiting pen-gesture recognition methodology is extraction of geometric features. As an example consider the Unistrokes [Goldberg and Richardson, 1993] recognition algorithm [Goldberg, 1997]. In [Goldberg, 1997] six geometric features are extracted from the user’s pen trace. Examples of two such features are the spatial distances between the horizontal and vertical components of the pen down and pen up locations [Goldberg, 1997]. Once the six features have been extracted, the features are used to look up the best matching Unistroke in a table.

The algorithm in [Goldberg, 1997] is template-based and does not rely on training data. Rather, the table is built by hand to discriminate among the possible pen-gestures in the Unistrokes alphabet. While this approach may be sufficient for a specialized application with a small set of pen-gestures, it is unlikely the approach would scale as the number of pen-gestures in the template set increases.

#### 4.2.1 Linear Machines

An alternative to template matching is to train a classifier to recognize certain pen-gestures written by the user. The classifier is trained using features that are extracted from prototypical pen traces representing a pen-gesture. A common statistical classifier is the linear machine [Duda and Hart, 1973]. An example of a linear

---

<sup>2</sup> mozgestOverlay.js revision 1.198.4.3 code lines 471-561. Accessible at: <http://www.mozdev.org/source/browse/optimoz/mozgest/mozgest/content/mozgestOverlay.js?rev=1.198.4.3&content-type=text/x-cvsweb-markup> [Accessed April 18, 2007].

machine pen-gesture classifier is the Rubine recognizer [Rubine, 1991]. It uses 13 features, such as initial slope angle, length of the trace, etc. The Rubine recognizer have been used in many research projects such as for example an automated website design tool [Newman, Lin, Hong and Landay, 2003] and a pen-gesture design toolkit [Long, Landay and Rowe, 1999].

Mathematically a linear machine is a set of linear discriminant functions  $\{g_i(\mathbf{x})\}_{i=1}^c$  operating on a set of  $c$  classes  $\{\omega_i\}_{i=1}^c$ . The classifier assigns  $\omega_i$  to an input feature vector  $\mathbf{x}$  if [Duda and Hart, 1973]:

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \text{ for all } j \neq i \quad (4.1)$$

In the case of a linear machine, the covariance matrix is  $\sigma^2\mathbf{I}$ , where  $\sigma^2$  is the variance, and  $\mathbf{I}$  is the identity matrix. The linear discriminant function is:

$$g_i(\mathbf{x}) = \mathbf{w}_i' \mathbf{x} + \omega_{i0} \quad (4.2)$$

where

$$\mathbf{w}_i = \frac{1}{\sigma^2} \boldsymbol{\mu}_i \quad (4.3)$$

and

$$\omega_{i0} = -\frac{1}{2\sigma^2} \boldsymbol{\mu}_i' \boldsymbol{\mu}_i + \log P(\omega_i) \quad (4.4)$$

where  $P(\omega_i)$  is the prior class probability. Equations 4.2-4.4 are derived from the assumption that the feature vectors follow a multivariate Gaussian distribution [Duda and Hart, 1973].

There are some limitations of this approach for pen-gesture recognition. First, a linear machine requires training data. That means every pen-gesture must be entered by a user (and each inputted pen-gesture is assumed to represent a prototypical input version of the pen-gesture) several times for the classifier to be able to handle deviations in the signal. Second, a linear machine is a simplification of statistical pattern classification. First, the features are assumed to be statistically independent, and some features such as the length and duration of the pen-gesture do not fulfill this criterion. Second, it is assumed that each feature has the same variance. Analytically, another problem with a linear machine is the possibility of two geometrically dissimilar shapes to be seen as completely identical to the recognizer. For an example of this deficiency with the recognizer proposed by Rubine [1991], see Figure 4.2.



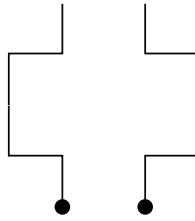


Figure 4.2. The left and right pen-gestures (starting points marked by dots) are completely ambiguous for the classifier presented in Rubine [1991].

There are also advantages to a pen-gesture recognizer based on a linear machine. It is easy to implement, and should have reasonable performance for a small set of pen-gestures.

#### 4.2.2 Elastic Matching

Elastic matching is a template matching procedure that is popular in image processing. It is inspired by a dynamic programming method used to align time-shifted wave signals in speech recognition called dynamic time warping [Rabiner and Juang, 1993]. The earliest use of elastic matching for hand-drawn line matching appears to be a paper by Burr [1981]. Tappert [1982] presents a method of using elastic matching in handwriting recognition. Niblack and Yin [1995] proposes using elastic matching for shape matching in the IBM QBIC (query by image content) image search system.

The central idea in elastic matching is that a distance between two sets of points (or connected lines) can be computed by finding the minimum amount of deformation necessary to transform one set of points into the other. Elastic matching is scale, translation and rotation invariant if one pattern is preprocessed to a “best fit” to the other pattern (e.g. via least-squares or another method).

The advantage of elastic matching is that it can be readily used as a template matcher without any training data. Further, it is relatively noise resistant method in comparison to more primitive region encoding and feature matching methods. A downside is the computational demands of the method, especially in the light that for most applications it is not a metric [Fagin and Stockmeyer, 1998] and therefore difficult to index and search using metric-based methods [Chávez, Navarro, Baeza-Yates and Marroquín, 2001]. In online handwriting recognition, where the concept of an ideal template is not as attractive as in pen-gesture recognition, elastic matching has mostly been replaced by statistical data-driven methods [Tappert, Suen and Wakahara, 1990; Plamondon and Srihari, 2000].

For general pen-gesture recognition, elastic matching can be transformed into a linear matching function (no stretching necessary) by resampling both patterns to contain

the same amount of equidistant-spaced points. The advantage by first resampling both patterns is that the comparison between the patterns is linear in relation to the number of sample points instead of quadratic. This form of “zero look-ahead elastic matching” is the foundation for the shape channel and shape distance function defined later in this chapter.

### 4.3 Continuous Shape Writing Recognition

This section describes two technical solutions to recognize continuous shape writing. The writing in this chapter is focused on principles rather than implementation.

First, it is important to keep in mind that many different methods can be used to tackle the fundamental problem of continuous shape writing recognition. As an example, consider the perhaps most obvious method: record all letter keys crossed by the user’s pen trace (in sequence). Now use a lexicon to look up all words that can be formed inside this letter key sequence (first and last letter keys must be present in the word) and rank them according to their frequency in a large corpus. Return the highest ranked word as the best match. Such an approach works if the lexicon is small. However, consider the fragileness of the method. First, the user must articulate the pen-gesture over the letter keys that comprise the intended word. This essentially changes shape writing into traditional software keyboard typing. The only difference is that the user slides the pen from letter key to letter key instead of tapping with the pen. Second, consider what happens if the user unintentionally slides across an unintended letter key. In such case, the user might get a completely different word because the unintended letter key will contribute to the search for the best matching word. For example, in Figure 4.3 the pen trace is geometrically more similar to *white* but the simplified recognizer could just as well recognize the pen trace as *wire*. Clearly, a more sophisticated approach is required for continuous shape writing to be viable.

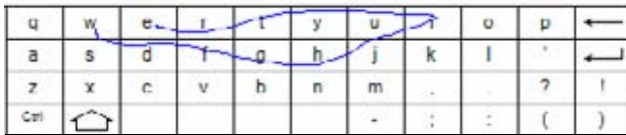


Figure 4.3. The pen trace intersects the valid letter key combinations *white* and *wire*.

From a strictly theoretical point-of-view all possible letter key combinations that form words or parts of valid words can be captured in a probabilistic model. Given evidence such as for example the user’s pen trace and previously written words the recognizer infers the user’s intended word from this model and a model of the user’s language.

Theoretically, many well-known approaches such as linear discriminant functions and neural networks, etc. can be applied to the problem. See Duda et al., 2001 for an extensive review of statistical classification methods.

In practice a probabilistic model is impractical to build. First, there are thousands of possible words. Second, continuous shape writing is a novel text entry method and unlike handwriting there does not exist a “right” way to write with continuous shape writing. Therefore there is a bootstrap problem – to be able to record pen trace data from users a shape write a system must be built. However, without any training data, how can a reliable recognition system be constructed?

#### 4.3.1 Multi-Channel Recognition

One novel solution is to use a multi-channel architecture where each channel does not necessarily have the discriminative power on its own, but the collective power of two or more channels can separate the ideal shapes of words.

The multi-channel approach is based on the following observations. First, shape alone cannot discriminate among thousands of words in the lexicon. This is further supported later in this chapter. Second, location information, i.e. where the user’s pen gesture is located on the software keyboard, can be used to discriminate among words that are close in shape alone.

Shape and translation comparisons are different and need to be combined into a single confidence score. Figure 4.4 illustrates the overall workflow with the channel architecture. This process is known as “combining classifiers” or “sensor fusion” in the pattern recognition community [Xu, Krzyák and Suen, 1992].

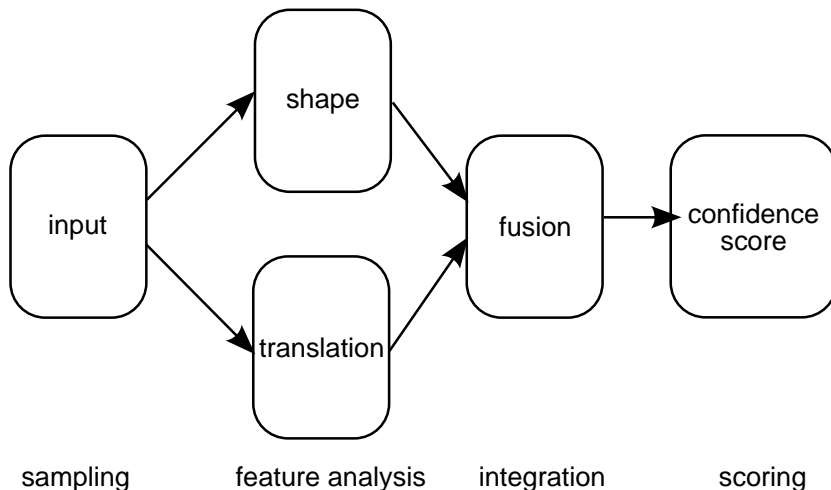


Figure 4.4. The recognition process for continuous shape writing.

#### 4.3.1.1 *Shape Channel*

The shape channel analyzes a user's unknown pattern (the user's pen trace) against a template pattern (the ideal shape of a word on a software keyboard layout). At an early stage the shape channel used elastic matching. However, it was discovered that sufficient shape information is obtainable by a simpler distance measure dubbed proportional matching.

The proportional matching cost between an unknown pattern  $u$  and a template pattern  $t$  is defined as:

$$x_s = \sum_{i=1}^n \|u_i - t_i\| \quad (4.5)$$

where  $n$  is the total number of sampling points in an individual pattern. It is easy to see that elastic matching with zero look-ahead reduces to Equation 4.5. Before applying Equation 4.5 the patterns are normalized in scale and translation.

Normalization in scale is achieved by scaling the largest side of the bounding box of a pattern to a pre-determined length  $l$ :

$$s = \frac{l}{\max(w, h)} \quad (4.6)$$

where  $w$  and  $h$  are the original width and height of the bounding box. Normalization in translation (location) is achieved by translating the patterns' geometric centroids to the origin in the coordinate system.

The final result of the shape channel is an approximate scale and translation invariant distance measure of the similarity between the patterns, based on the average sum of the corresponding equidistant sample points' spatial distance.

#### 4.3.1.2 *Location Channel*

Shape alone cannot discriminate sufficiently among tens of thousands of word shapes. Therefore a second channel is introduced that examines where on the software keyboard the unknown pattern is positioned. The rationale for having such a channel is twofold. First, location information provides increased recognition accuracy. Second, location is part of the user's memory of the shape for a word and therefore will be reproduced during pen-gesture articulation.

Table 4.1 lists how many words in a lexicon consisting of 20,000 words that are completely identical in shape (scale-translation independent spatial similarity). With the QWERTY keyboard layout 7.3% of the words' shape representations are identical if only shape is considered. With the ATOMIK keyboard layout 5.6% are identical in shape.

Table 4.1. Completely confusable word pairs on the QWERTY and ATOMIK software keyboard layouts when varying shape and location features are considered. The lexicon contains 20,000 words

Features	QWERTY	ATOMIK
Shape	1461	1117
Shape and start	609	519
Shape and end	589	522
Shape and both ends	537	493

Therefore the location algorithm computes the distance of the unknown input trace  $u$  to the template (ideal) trace  $t$  of word  $w$  on the software keyboard (now both  $u$  and  $t$  are absolute).  $t$  is defined by the lines connecting the centers of the letter keys that constitute  $w$ . Both  $u$  and  $t$  are re-sampled to a fixed number  $n$  of equidistant points. The location channel distance is defined as:

$$x_i = \sum_{i=1}^n \alpha(i) \delta(i) \quad (4.7)$$

where  $n$  is the number of sample points in the patterns.  $\delta(i)$  is defined as:

$$\delta(i) = \begin{cases} 0, & D(u,t) = 0, D(t,u) = 0 \\ \|u_i - t_i\|, & \text{otherwise} \end{cases} \quad (4.8)$$

where  $u_i$  and  $t_i$  are the  $i$ th points of  $u$  and  $t$  respectively.  $D$  is in turn defined as:

$$D(p, q) = \sum_{i=1}^N \max(d(p_i, q) - r, 0) \quad (4.9)$$

where  $d$  is:

$$d(p_i, q) = \min(\|p_i - q_1\|_2, \|p_i - q_2\|_2, \dots, \|p_i - q_n\|_2) \quad (4.10)$$

where  $n$  is the number of sample points in the patterns and  $r$  is the radius of an alphabetical key. This means that an invisible “tunnel” is formed of one key width that contains all letter keys in  $w$ . A perfect distance score of zero is given when the entire gesture input trace  $u$  is within this tunnel of  $t$ . Otherwise, the sum of the spatial point-to-point distances is used. In other words, Equations 4.7-4.10 give special weight to traces that are contained within the tunnel of radius  $r$  whose path is formed by serially connecting all the individual keys used in a word. This was based on the observation from actual use that when all letters in a word are traced (“tunneling”), one would expect the word to be recognized no matter what the shape of the trace is.

$\alpha(i), i \in (1, n)$  are weights for different point-to-point distances  $\left(\sum_{i=1}^n \alpha(i) = 1\right)$ .

The shape of  $\alpha(i)$  can be set in various ways. For example it could be dynamically trained through a large amount of data when available. It can also be prescriptively set. Currently the system uses a function that gives the lowest weight to the middle point, and the rest of the points' weights increase linearly towards the two ends. This is because when producing a pen-gesture it is easier for the user to pay visual attention to the beginning and ending points than the rest of the locations.

#### 4.3.1.3 Channel Integration

The shape and location channels output distance scores between an unknown pen-gesture and templates drawn from the lexicon. These distances in the two channels are not on a common scale and cannot be directly compared. The issue of multiple classifier integration of distances or scores (sensor fusion) is not new in pattern classification but is either too general (e.g. majority voting and related methods in Duda et al. [2001]) or too specialized to a certain domain (e.g. Xu et al. [1992]). Therefore the system uses a devised method specifically for continuous shape writing recognition to convert channel distances to a common integration scale.

As is common in engineering, a reasonable assumption is that the distance from an input pen-gesture to the template of the intended word (in either channel) follows a Gaussian distribution. In other words, if an input pen-gesture has distance  $x$  to a template  $y$ , the probability of  $y$  being the targeted word can be calculated using the Gaussian probability density function:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right] \quad (4.11)$$

where  $\mu = 0$  and  $\sigma$  can usually be obtained through training from large amount of data.  $\sigma$  reflects how sensitive a channel is. For example if  $\sigma$  equals to one key radius, those templates whose distance to the input pen-gesture are greater than one key width ( $2\sigma$ ) have practically zero probability of being the user's intended word. In the system  $\sigma$  is used prescriptively as a parameter to adjust the weight of the contribution of each channel. The greater  $\sigma$  is, the more flat the  $p(x)$  distribution will be, and hence the less discriminatory the channel is when it is integrated with the other channels (hence less weight). From the word candidates  $w \in W$ , the marginalized probability of a word  $w$  with distance  $x$  being the user intended target word is

$$p'(w) = \frac{p(x)}{\sum_{i \in W} p(i)} \quad (4.12)$$

Finally probabilities from both the shape ( $p'_s$ ) and location ( $p'_l$ ) channels are integrated into a confidence score according to Bayes' rule:

$$c(w) = \frac{p'_s(w)p'_l(w)}{\sum_{i \in W_s \cup W_l} p'_s(i)p'_l(i)} \quad (4.13)$$

where  $p'_s(w)$  and  $p'_l(w)$  are the probability scores from the shape and location channel respectively; and  $W_s$  and  $W_l$  are the sets of word candidates for each channel.

### 4.3.2 Shape-Translation Distance Function

This subsection and the next introduce an alternative variant of continuous shape writing recognition. It was devised based on the following three observations of the channel architecture outlined earlier. First, analytically, it is hard to understand what parts of the shape and location channels in the channel architecture that exactly contribute to a recognition result. Second, the individual  $\sigma$  parameters for each channel are difficult to tweak because of their non-linear effect and resulting interaction between the two channels. Third, that distance scores follow a Gaussian distribution is an assumption, not a fact.

The proposed distance function in this subsection is methodologically more intuitive than the previously presented channel architecture because measurement and integration of shape and translation scores are in the same measurement space.

The basic assumption is that the similarity distance functions for shape and translation in the recognizer are linearly related. Then the fusion of these two functions can be defined as a linear combination.

Consider two patterns  $X$  and  $Y$  as ordered sequences of equidistant points  $\{\mathbf{x}_i\}$  and  $\{\mathbf{y}_i\}$  respectively. In the following discussion  $X$  will be considered the unknown pattern, and  $Y$  will be considered the template pattern.

Now define the linear similarity distance between two patterns  $X$  and  $Y$ ,  $|X| = |Y|$ , as:

$$d(X, Y) = \sum_{j=1}^m \left( \frac{1}{n} \sum_{i=1}^n \delta_j(\mathbf{x}_i, \mathbf{y}_i) w_j(i) \right) \quad (4.14)$$

where  $m$  is the number of distance functions,  $\delta(\mathbf{x}, \mathbf{y})$  is a distance function between two points, and  $w(i)$  is a weighting function.

Equation 4.14 is a weighted sum of an ensemble of distance functions. The measurement space of the distance functions must be the same otherwise the total sum

is meaningless as a distance. Note that if all comparisons are zero the distance between the patterns is zero. In this case the patterns are considered identical.

As previously discussed, there are at least two important distance functions needed to separate patterns for effective continuous shape writing recognition. The shape distance function  $\delta_s$  measures overall shape similarity between two corresponding points in two patterns. The translation distance function  $\delta_t$  measures the degree of separation in location between two corresponding points in two patterns.

#### 4.3.2.1 Shape Distance Function

The shape distance function returns a measure that reflects shape similarity between two patterns  $X$  and  $Y$ , by measuring the Euclidean distance between two points  $\mathbf{x}_i$  and  $\mathbf{y}_i$ ,  $0 < i \leq n$ , where  $n$  is the (equal) number of sample points in the patterns. To ensure that  $\delta_s$  and  $\delta_t$  share a common measurement space the points must be normalized into the coordinate system that measures translation distance. In the following discussion this space will be called a template normalized space.

Define an affine transformation matrix  $\mathbf{T}$  in homogeneous coordinates that transforms point  $\mathbf{x}$  into the local coordinate system of  $\mathbf{y}$  in such a way that a good fit is obtained between  $X$  and  $Y$ . A straight-forward but effective example of such a transformation matrix is:

$$\mathbf{T} = \begin{bmatrix} s & 0 & dx \\ 0 & s & dy \\ 0 & 0 & 1 \end{bmatrix} \quad (4.15)$$

where

$$s = \frac{\max(w_X, h_X)}{\max(w_Y, h_Y)} \quad (4.16)$$

where  $w_X, h_X, w_Y, h_Y$  refer to the width and height of the bounding boxes of patterns  $X$  and  $Y$  respectively; and  $dx$  and  $dy$  is the distance between the geometric centroids (centers of mass) of  $X$  and  $Y$ .

Other rubber-band transformations are also possible, e.g. shear and rotation.

Next, define a point  $\mathbf{x}$  as a column vector in homogeneous coordinates:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (4.17)$$



where  $x$  and  $y$  represents the horizontal and vertical components of the point. Then the matrix product  $\mathbf{T}\mathbf{x}$  is an alibi transformation of the point  $\mathbf{x}$  into the template normalized space of the template pattern  $Y$ .

Hence define  $\delta_s$  as:

$$\delta_s(\mathbf{x}, \mathbf{y}) = \|\mathbf{T}\mathbf{x} - \mathbf{y}\|. \quad (4.18)$$

This means that the shape distance similarity function measures the degree of similarity between two patterns' points, where one pattern has been normalized into the measurement space of the other pattern.

#### 4.3.2.2 Translation Distance Function

Translation distance is defined as:

$$\delta_t(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| \quad (4.19)$$

In other words, translation distance is the Euclidean distance required to move one pattern (on a point-to-point basis) into the other pattern's position.

#### 4.3.2.3 Shape-Translation Distance

Combining in shape-translation distance  $st$  is then defined as (cf. Equation 4.14):

$$\begin{aligned} st(X, Y) &= \frac{1}{n} \sum_{i=1}^n (\delta_s(\mathbf{x}_i, \mathbf{y}_i)w_s(i) + \delta_t(\mathbf{x}_i, \mathbf{y}_i)w_t(i)) \\ &= \frac{1}{n} \sum_{i=1}^n (\|\mathbf{T}\mathbf{x}_i - \mathbf{y}_i\|w_s(i) + \|\mathbf{x}_i - \mathbf{y}_i\|w_t(i)) \end{aligned} \quad (4.20)$$

The contribution of each per-point comparison is controlled by adjusting the corresponding weighting function  $w(i)$  where  $i=1, \dots, n$  indexes total  $n = |X| = |Y|$  comparisons.

The weighting scheme enables fine-grained parameter adjustment to recognition. For instance to use strictly shape similarity information set  $w_t(i) = 0$ , to use strictly translation information set  $w_s(i) = 0$ , or to let only the first and last vertices comparison contribute to translation distance in a comparison between total  $n$  comparisons, set:

$$w_t(i) = \begin{cases} 1, & i \in \{1, n\} \\ 0, & \text{otherwise} \end{cases} \quad (4.21)$$

For another example, to achieve a gradient translation distance weighting where the first and last translation distance comparisons have maximum contribution, and the

contribution decreases linearly from the two ends towards the middle comparison that has a weight  $\alpha \in [0,1]$  define:

$$w_i(i) = 1 - \begin{cases} i - 1 \frac{\alpha}{p(n)}, & i \leq n/2 \\ (n - i) \frac{\alpha}{p(n)}, & i > n/2 \end{cases} \tag{4.22}$$

where  $0 < i \leq n$ , and

$$p(n) = \begin{cases} (n/2) - 1, & n \text{ is even} \\ (n - 1)/2, & n \text{ is odd} \end{cases} \tag{4.23}$$

Figure 4.5 illustrates the resulting weighting scheme in a hypothetical matching situation for ten points and  $\alpha = 0.5$ .

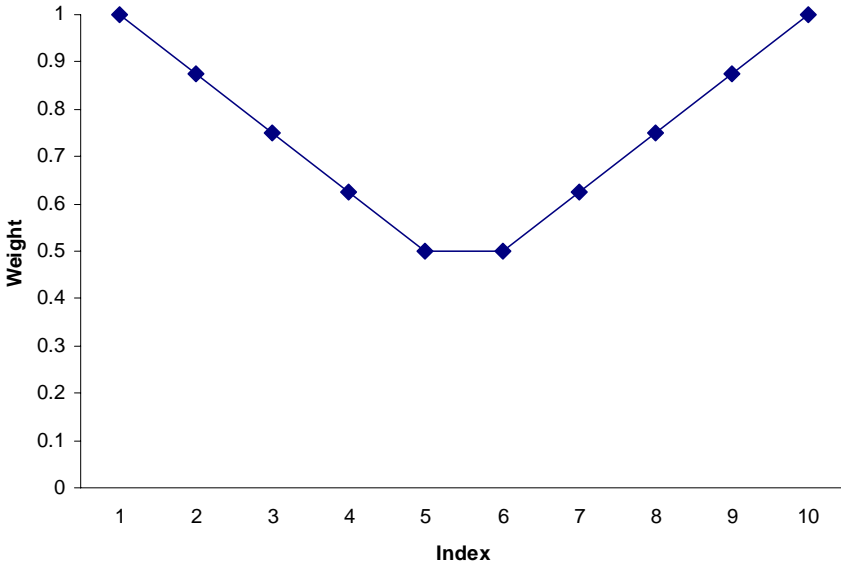


Figure 4.5. A graphical illustration of Equation 4.22. The figure plots weight as a function of index.

## 4.4 Conclusions

This chapter has presented and motivated two approaches to recognizing continuous shape writing. While there are many possible alternative approaches to continuous shape writing recognition the methods outlined here have been implemented and tested by users. Further, Experiment 3.2 and Experiment 3.3 verify that recognition works well in practice. Future work involves collecting trace data from actual users, and investigating if a statistical model, or a combination of a statistical and model-based method, can improve recognition accuracy.

The methods are also suitable for general pen-gesture recognition (simply leave out the location channel or the translation distance function), but the effectiveness of the recognition algorithms in such applications has not been formally investigated.



# Chapter 5

## Continuous Shape Writing for Control<sup>1</sup>

---

This chapter investigates how continuous shape writing can be used to control applications. The implementation of continuous shape writing for control is dubbed command strokes.

### 5.1 Introduction

---

The recurring rise and fall of pen-based computing speaks to both the promise and the difficulty of using the pen as the primary interaction device. The current generation of pen-based computing devices, including pen-based Personal Digital Assistants (PDAs), smart phones and in particular the tablet PCs, are no doubt much more advanced in both hardware and software than the early generations of pen-based computers. However, it is also evident that in comparison to the desktop or laptop computers, today's pen-based computers still exhibit a large degree of handicap to the user. Thinner and lighter hardware will certainly make the Tablet PC ever more attractive, but that alone will not suffice. To continue the current cycle of the pen-computer's rise, and keep it from falling again, user interface researchers have to develop innovative techniques that take advantage of the fluidity and dexterity of the pen, while addressing the interaction problems that handicap the user.

One of the most mundane shortcomings of today's pen-based computing devices lies in issuing commands. Understandably, the basic method of issuing commands on a pen-based computer is the same as on a desktop PC: linear and hierarchical pull-down menus. The limitations of pull-down menus on desktop computers have long been realized [Callahan, Hopkins, Weiser and Shneiderman, 1988]. Pull-down menus are much more problematic on a pen-based touch-screen computer for a number of reasons. First, the pen (stylus) and its holding hand often obscure the very items on the pull-down menu the user needs to find and select. Second, pen motion on a screen has to be one-to-one in scale. This is in contrast to other pointing devices—such as the mouse—whose control-to-display gain is rate-accelerated (popularly known as the power-mouse), so that one does not have to move over a large distance to reach a far-away menu. Furthermore, moving a pen on the screen is more fatiguing than moving a mouse supported on a desktop.

---

<sup>1</sup> This chapter is a revised and slightly extended version of Kristensson and Zhai [2007a].

What comes to linear menus' rescue on desktop and laptop computers are keyboard shortcuts. Frequent commands such as *Copy* and *Paste* have de-facto standard hotkey shortcuts across software applications. Although the number of shortcuts a user remembers might be small, the skewed distribution in use frequency, similar to the Zipf's law [Zipf, 1935] effect in the distribution of word usage in a language, elevates the percentage of hotkey use disproportionately. Without a keyboard, these shortcuts are often what the user misses the most on a pen-based computer.

This chapter presents a new and practical command entry technique for pen-computers called command strokes. Command strokes are pen-gesture traces defined on a graphical keyboard according to the letters in the commands, such as *c-o-p-y* and *p-a-s-t-e*. Command strokes offer users a complementing method of directly selecting any command without needing to browse a menu hierarchy.

The development of command strokes followed an iterative process. An early incarnation of the concept was compared with traditional pull-down menus. Encouraged by the positive results and informed by the feedback gained from the experiment a second iteration of the technique was developed: command strokes with preview.

The structure of this paper reflects the iterative development process: The first part presents the initial incarnation of command strokes and its evaluation. The second part presents the second iteration of the method and two experiments that investigate how the visual preview functionality impacts end-users. Last, the work is compared to previous research.

## 5.2 Command Strokes

---

This chapter proposes to use continuous shape writing as a command entry method. The new command entry method is called command strokes, and it is intended to complement pull-down menus. Command strokes can be divided into two classes. A short command stroke is a pen-gesture trace from one or more modifier keys, such as *Ctrl*, *Alt*, *Fn*, and *Shift*, to a letter key, corresponding to the physical keyboard hotkey combinations. A long command stroke is the pen-gesture trace from a modifier key to a sequence of letter keys based on the name of the command.

### 5.2.1 Short Command Strokes

In a traditional desktop environment the user usually selects commands from a pull-down menu. However, frequently used commands such as *Copy* and *Paste* have mnemonic key identifiers attached to remind the user of the alternative shortcut (Figure 5.1). Physical hotkeys have been directly carried over in today's tablet PCs, where users can tap the hotkeys on a software keyboard.

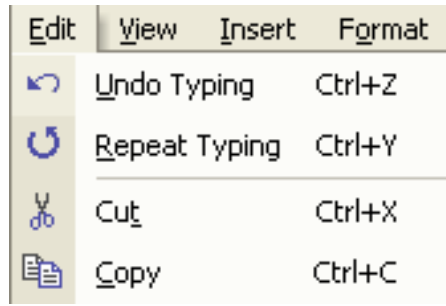


Figure 5.1. Example of commands in a pull-down menu with the hotkey assignment shown to the user to the right.

Short command strokes transform the physical keyboard-based hotkeys to a truly fluid pen-based form in order to take better advantage of the natural affordances of the pen.

As observed in Chapter 3, with continuous shape writing it is not necessary on a software keyboard to tap one key at a time in a “chicken head motion” to input information. Instead, a command can be recognized as a trajectory pattern on the keyboard, see Figure 5.2.

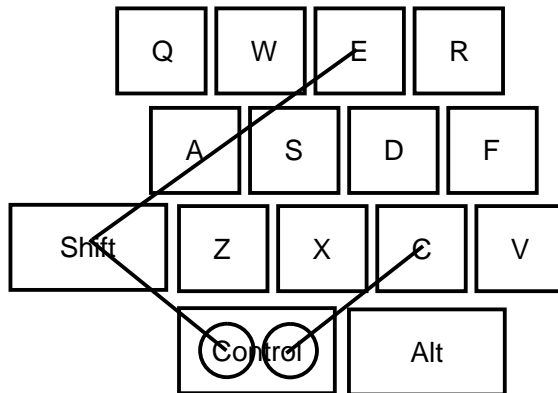


Figure 5.2. The line traces of *Ctrl-c* and *Ctrl-Shift-e* as patterns on a software keyboard with the QWERTY layout.

The short command stroke trajectory defines a pattern or pen-gesture. The user’s input can be matched against a collection of pen-gesture templates, allowing a degree of error tolerance in the interface: a user needs only to draw an approximate pen-gesture to invoke a specified action, as long as it is closer to the intended pen-gesture template than other templates in the database. A novice user starts out by tracing the key sequences. Over time, the pen-gesture builds up in the user’s memory and the user can quickly flick the pen-gesture without looking much at the keys. The technique can be expanded to any arbitrary sequence of keys. For instance, Figure 5.2 also shows the hotkey trace *Ctrl-Shift-e*.

### 5.2.1.1 Problems with Short Command Strokes

Short command strokes, being directly mapped from the physical keyboard hotkeys, do have limitations due to this heritage. Hotkey commands typically consist of a very short sequence of keys and some are designed to be easily reachable with one hand and pressed down simultaneously (e.g. *Ctrl+C* for *Copy*). This causes two problems. First, very short pattern sequences are much more confusable in the recognition process. Second, the pen-gestures of some different commands can be quite similar from a user's point of view. The pen-gestures *Ctrl-c* for *Copy* and *Ctrl-x* for *Cut* are in fact very close to each other (cf. Figure 5.2).

### 5.2.2 Long Command Strokes

Figure 5.3 shows an example of a long command stroke for the action *Copy* whose template starts on *Ctrl* and intersects the letter keys *c-o-p-y* in sequence. Since long command strokes are richer in shape features, they tend to be more error tolerant than short command strokes: as long as the user's gesture is geometrically similar to the long command pattern, the command can be recognized and executed (see Figure 5.3).

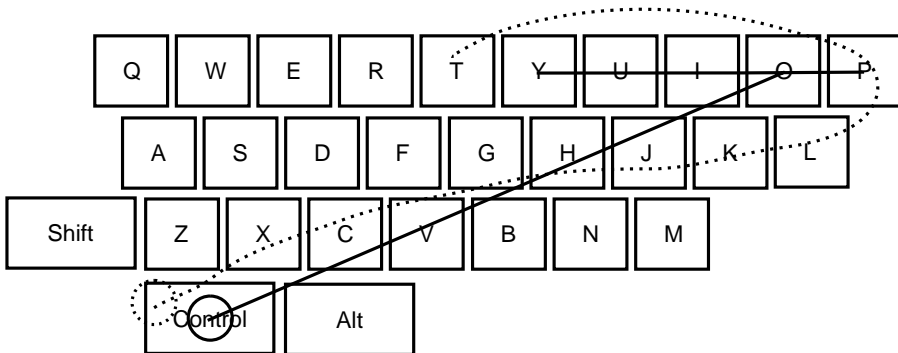


Figure 5.3. The solid lines show the long command stroke for *Copy*. The dashed spline shows a user's actual pen-gesture of the command.

There are many special design issues that must be solved to make long command strokes effective. First, an algorithm needs to be devised that can compute practical non-ambiguous long command strokes. Second, recognition accuracy is more challenging for commands that originate from the same letter key, than for ordinary text writing. Third, long command strokes need to be integrated with continuous shape writing for text. Fourth, effective visual interfaces are required. The rest of this section discusses these issues in order.

#### 5.2.2.1 Finding Practical Long Command Strokes

Articulating full command names is unnecessarily lengthy, and particularly impractical for commands with command names that are very verbose, an extreme example is *Save As Web Page*. It is possible to define long command strokes



unambiguously by abbreviations or acronyms rather than full names. For example the command *Track Changes* can probably be gestured as *Ctrl-t-r-a*. This section presents an algorithm that systematically finds practical non-conflicting non-ambiguous abbreviations and acronyms for a set of command names. See Figure 5.4 and the Visual Feedback subsection later in this section on how the user can know these shorter alternatives.

Let  $\Omega_w$  be the working set of command names initialized to a set where all command names are strings of only two symbols, e.g. the command name *Copy* is reduced to *Co*;  $\Omega_c$  is the set of command names that are confusable, i.e. easily misrecognized due to close neighbors in the recognition space, and  $\Gamma$  is the set of shortest non-ambiguous versions of the command names that are accumulated so far. The algorithm returns this set together with the command names that could not be further expanded.

---

```

function FIND-SHORTEST-COMMAND-NAMES
 $\Omega_w \leftarrow$  shortest desired initial command names
 $\Omega_c \leftarrow$  GET-COLLISIONS ( $\Omega_w$ )
 $\Gamma \leftarrow \Omega_w - \Omega_c$ 
while ( $\Omega_c \neq \emptyset$  and CAN-EXPAND-NAMES?( $\Omega_c$ )) do
     $\Omega_w \leftarrow \Gamma \cup$  EXPAND-NAMES ( $\Omega_c$ )
     $\Omega_c \leftarrow$  GET-COLLISIONS ( $\Omega_w$ )
     $\Gamma \leftarrow \Omega_w - \Omega_c$ 
end-while
return  $\Gamma \cup \Omega_c$ 

```

---

Although the term “command name” is used for ease of presentation, when implementing the algorithm a “command name”  $\omega$  is represented as a data structure  $\langle s_L, X, s_M, \sigma \rangle$ , where  $s_L$  is a string naming the command,  $X$  is the pattern of the command on the keyboard,  $s_M$  is the minimized version of the command so far, and  $\sigma$  is a flag having the value TRUE if the command name can be further grown, and the value FALSE otherwise. The algorithm repeatedly alternates between expanding the command names with conflicts (i.e. easily confusable by the recognizer) and re-computing new collisions. When there are no more collisions, or when there are collisions but the command names that have collisions cannot be further expanded, the algorithm halts and outputs the smallest command names that could be found along with the command names (if any) that could not be expanded.

The function GET-COLLISIONS returns the set of command names that have a conflict with at least one other command name (collision set). Two command names  $\omega_1$  and

$\omega_2$  are included in the collision set if it is true that  $C(\omega_1, \omega_2) < \gamma$  where  $C$  is a similarity function. Here  $\gamma$  is a threshold set by the system designer that is empirically determined. The value of the similarity function  $C$  is zero if  $\omega_1$  and  $\omega_2$  are identical and increases monotonically as  $\omega_1$  and  $\omega_2$  become more dissimilar.

The function CAN-EXPAND-NAMES? ensures the algorithm halts on all inputs. Without this predicate in the conditional loop, the algorithm would loop indefinitely if the set of collisions  $\Omega_C$  contained command names that could not be further expanded.

The function EXPAND-NAMES returns a set containing expanded command names drawn from  $\Omega_C$ . This function can be designed in many interesting ways. The simplest design handles the case of one word command names such as *Copy* and *Save*. In this case EXPAND-NAMES is designed to expand the command names gradually with one additional character at a time. In the case of command names that contain multiple words, such as the very verbose *Save As Web Page*, EXPAND-NAMES is designed to try to gradually expand one of the words in the sequence by one additional character. The word that is expanded in the sequence alternates between different calls to EXPAND-NAMES, e.g. the command *Save As Web Page* may gradually grow from *SA* to *SAWP* to *SaAWP* to *SaAsWP*. Many different schemes are possible. For example, another expansion technique could be to ignore vowels except in the beginning and only keep consonants, e.g. the command *Insert* can be grown from *In* to *Ins* to *Insr*. If the algorithm is used unsupervised a filter is probably needed here to make sure any particularly undesirable words are removed from consideration. It is important to note that a command name as long and complex as *Save As Web Page* is an extreme example of a command name, used here for demonstration purposes of the algorithm. In practice, it is expected that the vast majority of the commands being flicked as command strokes are more frequent and have much shorter names, such as *Search*, *Print*, *Save*, *Copy*, etc. To be more effective, the frequent commands in an application should be run against the algorithm first. Then, having assigned as short and non-conflicting command names as possible to these commands, the algorithm can be applied to the rest of the command corpus in the application, in consideration of the precedence already taken by the frequent commands.

#### 5.2.2.2 Increasing Accuracy in Recognition

A potential problem with long command strokes can occur if all commands are triggered by the same command initiator key. The continuous shape writing recognition engine uses two sources of information, shape and location, to infer the pen-gestures that are the best matches in the database (see Chapter 4 for more details). Both the shape and location represent the input pen-gesture as a series of ordered two-dimensional sample data points. For regular continuous shape writing the system assumes the most informative parts of the user's pen stroke is the beginning and end

segments, hence weighting the beginning and ending sample points more than the ones in the middle. However, when all commands are initiated from a small set of modifier keys (e.g. *Ctrl*, *Alt*, etc.) this is no longer true. For this reason the continuous shape writing recognition module is modified to enter a special command detection mode when recognizing long command strokes (the pen traces started on a modifier key). In this mode, the recognition process discards the first segment connecting the modifier key to the first letter key in the matching process. For example, if the user's pen trace is compared against the long command stroke template *Ctrl-c-o-p-y* the recognizer only compares the segment of the user's gesture corresponding to *c-o-p-y*, discarding the segment corresponding to *Ctrl-c*.

### 5.2.3 Detection of Command Strokes

An important aspect of command strokes is the trigger mechanism. If command strokes are used in conjunction with an ordinary software keyboard, command strokes and text input are easily separated. Since gesturing with a pen is a distinctively different event from tapping, users won't accidentally invoke commands when entering text.

If command strokes are used in conjunction with continuous shape writing text entry, the separation of commands and text input has to be carefully considered. Since a command stroke always starts from a function key, its shape and location is not likely to be the same as the geometric pattern of a regular word on the keyboard. This means that the two systems can simply be mixed. However the drawback of this approach is that it will be less tolerant to user's sloppiness / flexibility in articulating commands or ordinary text.

A more conservative method is to require command strokes to start from within the modifier key (such as *Ctrl*), which can be enforced by visual feedback. This method can clearly separate commands from text, giving the user more flexibility since the system will only search either the command set or the word lexicon. Although it is at the cost of requiring the user be more precise on the starting point of a command gesture, this method is currently preferred.

### 5.2.4 Visual Feedback

Since command strokes are recognition-based, it is important to inform users of the end result with visual feedback. Therefore visual feedback is integrated with the command stroke system. The visual feedback serves two purposes. First, the activated command is immediately displayed to the user. Second, the feedback also reveals the shortest unambiguous acronym in a distinct color. This allows a novice user who gestured *Ctrl-c-o-p-y* for *Copy* to realize that it is also sufficient to gesture *Ctrl-c-o-p* (see Figure 5.4).



Figure 5.4. The system has recognized the command *Copy* and overlays it on the keyboard. The shortest acronym the user has to articulate is drawn in a distinct yellow color.

#### 5.2.4.1 Resolving Ambiguity

A small number of command strokes may still be close to each other in shape and location after the algorithm outlined earlier in this chapter has found appropriate command names. To reduce possible user frustration, the system presents multiple candidates overlaid on the keyboard if there is more than one close match to the user's gesture (Figure 5.5). The user can either look at the alternatives to get a grasp of the commands available, or simply point or cross the desired alternative.

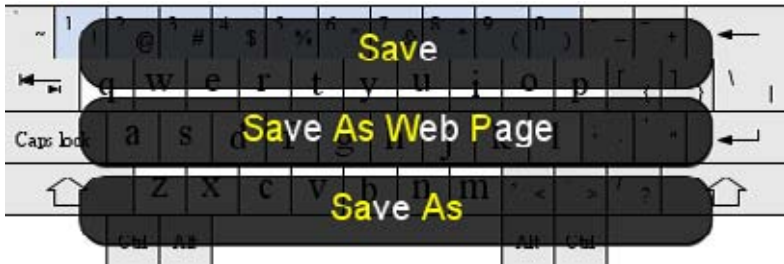


Figure 5.5. The user has entered *Ctrl-s-a* and system returns three possible choices: *Save (Ctrl-s-a-v)*, *Save As Web Page (Ctrl-s-a-s-w-p)* and *Save As (Ctrl-s-a-a-s)*. The user can select the desired command by tapping or crossing a selection and learn the SHARK commands for future use.

## 5.3 Experiment 5.1: Error and Response Time

There are many issues with command strokes that can be studied. The first experiment focused on a few basic questions: Can command strokes offer any performance advantage to complement the linear menus that is the de-facto standard technique for entering commands on a tablet PC or PDA today? What are the quantitative tradeoffs between short command strokes and long command strokes? Which technique do users prefer?

### 5.3.1 Method

#### 5.3.1.1 Design

In the experiment participants used a stylus to enter various commands in three conditions:

1. Linear menu. The participants entered commands by selecting them in a linear menu.
2. Short command stroke. The participants entered commands by gesturing their short command stroke, e.g. *Ctrl-c* for *Copy*.
3. Long command stroke. The participants entered long command strokes, e.g. *Ctrl-c-o-p* for *Copy*.

The order of the three conditions was balanced among the participants in this within-subject experiment. With each condition the participants entered 10 (trials)  $\times$  5 (command types) = 50 commands. The trials were randomly shuffled within each condition so the participant could not predict which command would appear before the trial starts. The participants were told to rest whenever they wanted between trials.

#### 5.3.1.2 Participants

12 volunteers were recruited for participation in the experiment. Their ages ranged between 20 and 50.

#### 5.3.1.3 Apparatus

The experiment was conducted on a 1 GHz tablet PC with a 12" TFT display with 1024  $\times$  768 pixel resolution. The test software was written in Java but the look and feel of the linear menus was set to the standard Windows XP look and feel which is used on the tablet PC in other applications (see Figure 5.6).

#### 5.3.1.4 Material

Shown in Table 5.1, the five types of commands tested in the experiment were drawn from the *File*, *Edit*, *View* and *Tools* menus in Microsoft Word 2002. We choose to resemble MS Word because it is a commonly used application. All participants had experience in using the pull-down menus in MS Word. All menu items assigned with hotkeys by MS Word default in these four menus were inserted into the lexicon of patterns the system could recognize.

The long command strokes' abbreviations were generated by hand before the algorithm was developed since this study was conducted in parallel to, and in fact informed, the system development.

Table 5.1. The commands used in the study.

Command	Short	Long
Copy	Ctrl-c	Ctrl-c-o-p
Print	Ctrl-p	Ctrl-p-r-i
Thesaurus	Ctrl-t	Ctrl-t-h-e
Track Changes	Ctrl-Shift-e	Ctrl-t-r-a
Find	Ctrl-f	Ctrl-f-i-n

In the linear menu condition, the commands *Copy* and *Find* were located in the *Edit* menu, *Print* in the *File* menu, *Track Changes* in the *Tools* menu and *Thesaurus* in the *Language* submenu in the *Tools* menu. Since our keyboard software did not implement the function keys the *Thesaurus* hotkey was changed from *Shift-F7* to *Ctrl-t*, which is in fact more memorable. Even though the hotkeys were taken from MS Word, *Copy*, *Print* and *Find* share hotkeys common to all MS Windows applications.

The selection of these five types of target commands in the study was biased in favor of the traditional pull-down menus. Only one of the five target commands was located in a submenu which obviously was more difficult to reach in the pull-down menu condition. Command strokes in contrast are not hierarchical so a submenu item in a pull-down menu is not necessarily more difficult for command strokes. In comparison MS Word 2002 by default contains 117 top-level menu items and 136 submenus item (56 in the *AutoText* sub-menu). Even more items are located at sub-menu levels in an Integrated Development Environment (IDE) such as Eclipse where a large number of frequently used commands such as *Indent Text* are located in submenus.

### 5.3.1.5 Procedure

As shown in Figure 5.6, to start a new trial the participant had to click on the CLICK FOR NEXT COMMAND button which brings out a new target command in the top panel of the experiment software. The participant then entered a command as fast and as accurately as possible. The command received by the software was displayed in the third panel. If a wrong command was entered the trial had to be repeated until the correct command was entered. Incorrect entries (and their repeated trials) did not contribute when we calculated the reaction and total time.

In addition to a brief introduction and explanation of each technique in the beginning of the experiment, the participants were given a “cheat sheet” (a reminder table) in all conditions to glance at in case they forgot how to do a particular command for the given condition. For example for the command *Copy* on the reminder table was “In Edit” in the linear menu condition, “Ctrl-c” in the short command stroke condition, and “Ctrl-c-o-p” in the long command stroke condition.

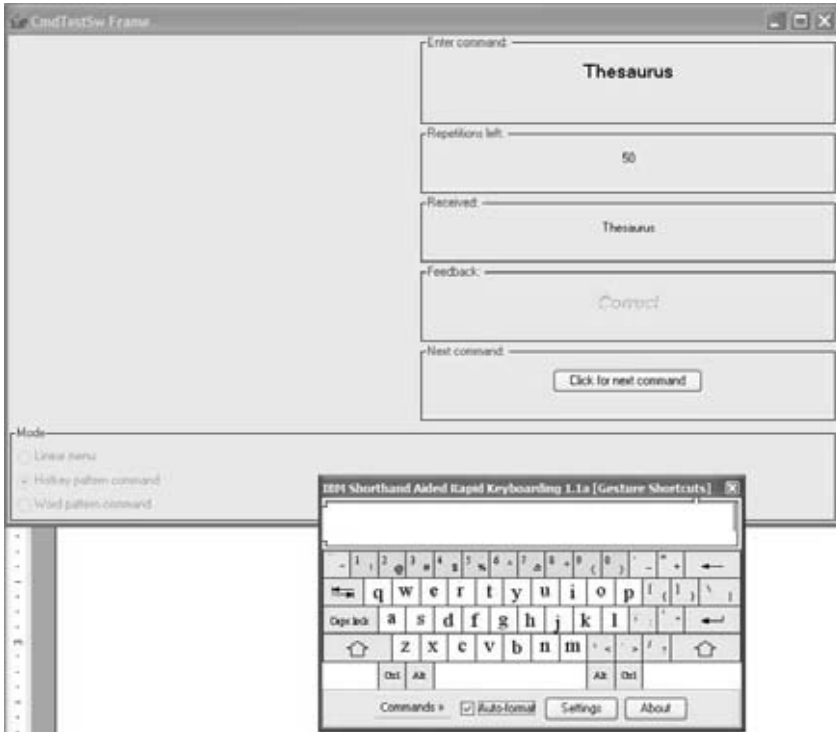


Figure 5.6. The software interface in the short command stroke task. The panels at the right display (from top to bottom): the command the user should enter, the number of trials left in the current condition, the command received from the user, whether the received the command was correct, a button that the user must click to start the next trial.

## 5.3.2 Results

### 5.3.2.1 Selection Time

Selection time was defined as the time from the user clicking on a menu to selecting (and lifting the pen) in the linear menu condition, and the time from the user presses the pen onto the stylus keyboard surface to lift it up in the command stroke conditions. Figure 5.7 shows the successful selection time with each of the three methods as a function of the trial number. Repeated measure variance analysis shows that the selection time difference between the three methods was statistically significant ( $F_{2, 22} = 99.9$ ,  $p < .0001$ ). *Post hoc* tests indicate all pair wise comparisons were significant ( $p < .0001$ ). Clearly selection time decreased with practice (Figure 5.7,  $F_{9, 99} = 23.7$ ,  $p < .0001$ ) but the differences between the three methods remained very large. Taking the average of the last four trials as example, short command strokes and long command strokes were 3.8 and 1.6 times faster than the pull-down menu method respectively.

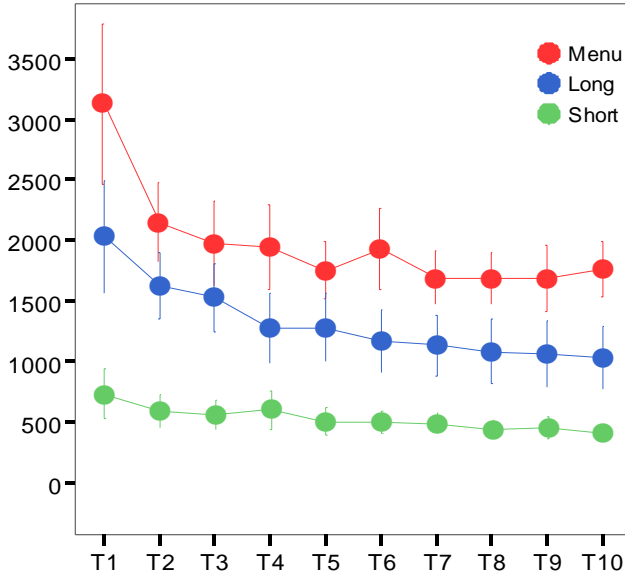


Figure 5.7. Mean and 95% confidence interval of selection time (in ms) of the three methods as a function of trial number.

There was a significant interaction between different commands and the three methods ( $F_{8, 88} = 39.38, p < .0001$ , Figure 5.8). In particular, although the linear menu was slower with all commands tested, it was particularly slow with *Thesaurus*, which was a nested sub-menu item (*Tools-Language-Thesaurus*).

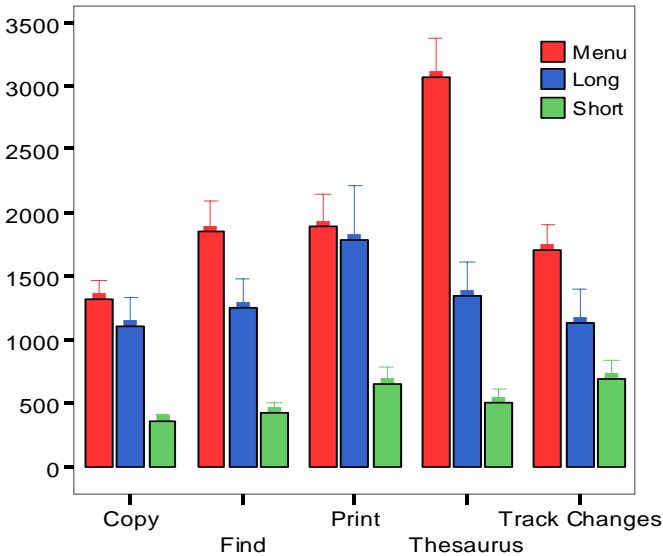


Figure 5.8. Selection time (ms) of the three methods and different commands.



The distances needed for the pen to travel (ideal traces) in the study are shown in Table 5.2. As a reference, the distance from the CLICK FOR NEXT COMMAND button in the test application to the *Ctrl* key on the keyboard was 382 pixels (measured between the centers of the buttons, Figure 5.6). Assuming that the user moves the pen at the same average speed (and excluding the initial movement from the CLICK FOR NEXT COMMAND button to the *Ctrl* key or the menu bar), short command strokes should be 1.4 times faster than linear menu, and linear menu should be 1.6 times faster than long command strokes. However, in fact long command strokes were 1.6 times faster than linear menu. Clearly movement distance is not a determinant of selection time between the three techniques.

Table 5.2. The ideal pixel distances for each command and task in the study.

Command	Menu	Short	Long
Copy	78	91	298
Print	258	277	587
Thesaurus	379	145	313
Track Changes	137	150	256
Find	217	115	297
<b>Average</b>	213.8	155.6	350.2

### 5.3.2.2 Reaction Time

Reaction time was defined as the time duration from the moment a new target command was presented to the moment of the first stylus contact with the pull-down menu or the stylus keyboard. Since occasionally the participants could be distracted or wanted to ask questions during this period, resulting in excessively long reaction time, these cases were truncated (two cases) at 32,676 ms, which was the largest number the statistics software could accept.

There was no statistically significant difference between the three methods in reaction time ( $F_{2, 22} = 0.35$ ,  $p = .71$ ). Reaction time decreased significantly with practice ( $F_{9, 99} = 20.9$ ,  $p < .0001$ ) particularly at the first two trials, and eventually stabilized at or just below 2000 ms. It is interesting to note that although novel to the participants, the command stroke methods did not take longer for the user to get to action than the linear pull-down menus.

Although the magnitude was small, there was a significant difference in reaction time between the commands ( $F_{4, 44} = 2.88$ ,  $p = 0.03$ ) and this difference did not interact with the methods significantly ( $F_{8, 88} = 2.02$ ,  $p = 0.054$ ). Fisher's PLSD *post hoc* tests show the only significantly different pair wise comparison is between *Copy* and *Track Changes* ( $p = 0.02$ ). It took somewhat longer for the user to decide what to do with a rarer and complex command such as *Track Changes*.

### 5.3.2.3 Total Trial Time

Total trial time was the sum of reaction time and selection time, from the moment a new target command was presented to the moment the system received a command.

Repeated measures variance analysis show that total trial time difference between the three methods was also statistically significant ( $F_{2, 22} = 21.57$ ,  $p < .0001$ ). *Post hoc* tests indicate all pair comparisons were significant ( $p < .0001$ ). Total trial time also decreased with practice ( $F_{9, 99} = 23.69$ ,  $p < .0001$ ) particularly during the first two trials.

The average total time of long command strokes was similar to that of the linear menu in beginning, but decreased two about two thirds of it. The differences between the three methods were quite large after the initial progress. Taking the average of the last four trials as example, short command strokes and long command strokes were 1.8 and 1.3 times faster than pull-down menu method respectively.

It is remarkable that the linear menu used in the experiment whose layout and components were identical to Microsoft Word hence should be familiar to most of our participants 1), still progressed significantly during the first few trials; 2), was still much slower than command strokes both in selection time and total time.

### 5.3.2.4 Error

The error rates were 2.8% for pull-down menus, 6.5% for short command strokes and 3.5% for long command strokes. Repeated measures variance analysis shows that error rate differed significantly across methods ( $F_{2, 22} = 6.12$ ,  $p = .0077$ ).

Fisher's PLSD *post hoc* tests indicate that short command strokes were significantly more error prone than long command strokes and linear pull-down menus ( $p < .05$ ) but the difference between long command strokes and linear menus was not significant ( $p = .64$ ).

Note that (Figure 5.9) when selecting a sub-menu command (*Tools-Language-Thesaurus*), linear menu was no better than short command strokes and worse than long command strokes in error rate. Note also that the error rate of long command strokes was comparable or better than linear menu except in the case of the *Print* command. This was because another command in the lexicon, *Go To* (*Ctrl-g-o-t-o*), forms a similar pattern as *Print* (*Ctrl-p-r-i*) on the QWERTY layout. The solution is to make *Print* more unique by expanding it with one additional letter, resulting in *Ctrl-p-r-i-n*. This result initiated the development of the algorithm that finds practical non-ambiguous command names, presented earlier in this chapter.

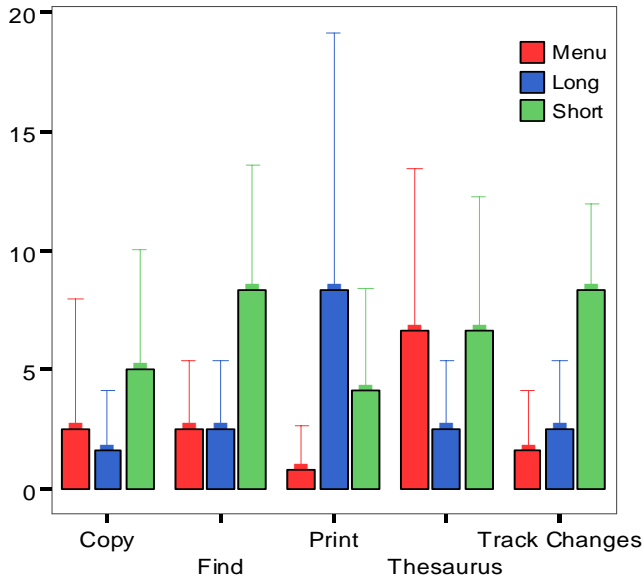


Figure 5.9. Error rates (%) in different conditions.

#### 5.3.2.5 User Ratings

No participants preferred linear menus and they all consistently rated linear menus as the most physically demanding technique. Five participants preferred long command strokes, five preferred short command strokes, and two stated that they preferred different modes for different situations, using short command stroke for very frequent actions such as *Copy*, and using long command strokes for less frequent commands. When asked for their reasons to prefer a specific technique, participants who preferred short command strokes said they were already familiar with the typing pattern of traditional hotkeys. Participants who preferred long command strokes stated better accuracy in recognition and / or the ease of remembering the commands as the main reason for preference.

## 5.4 Command Strokes with Preview

Although command strokes were rather successful in Experiment 5.1, through the experiment and additional informal user testing it was realized that novel, interesting and possibly advantageous improvements could be made. The result of the second design iteration is command strokes with preview.

### 5.4.1 Basic Idea

The overall goal in command stroke with preview is to give the user more flexibility and more certainty when using command strokes. For flexibility, each command can be entered with as long as, or as short as, a stroke on its complete path (e.g. *Command-C*, *Command-c-u* or *Command-c-u-t* can all do *Cut*), as long as the stroke

is unambiguous to the pattern recognizer. Consequently command stroke with preview effectively merges short and long command strokes. For certainty, the system displays what the command would be (a preview, see Figure 5.10) if the pen would be lifted from the current location. Through testing it was discovered that it is better to display a preview only if the stroke movement is relatively slow.

#### 5.4.1.1 Example

Suppose the user wants to issue the command *Copy*. The user starts by landing the pen on the *Command* key, and then drags it to the first letter key in the command, in this case the *c* key. Since *Command-c* matches another command (*Cut*) that is shorter and / or more frequently used, *Cut* is now previewed (Figure 5.10 top). Other commands that also match the sequence *Command-c*, in this case *Copy*, *Close* and *Comment*, are shown in a list of alternatives to the left of the center panel. To enter *Copy* the user either quickly slides the pen towards *Copy* in the leftmost box (see the Quick Pick subsection below) or continues to gesture towards the second letter key *o*. Since *Command-c-o* matches *Copy* the command *Copy* is now previewed (Figure 5.10 bottom). When the user lifts up the pen the *Copy* command is issued. It is important to remember that command stroke with preview still uses a pattern recognizer. For instance if the user is gesturing a pen trace geometrically close to *Command-p-r-i* the command *Go To* could appear instead because from the pattern recognizer's point-of-view *Command-g-o-t-o* is very similar to *Command-p-r-i*.

#### 5.4.2 Cancellation

An important feature when previewing is the ability to cancel the gesture. By dragging and releasing the pen over the semi-transparent cancellation icon (see Figure 5.10) the current gesture is cancelled and no command is executed.

#### 5.4.3 Dynamic Visual Preview

An important design goal was to make preview as unobtrusive as possible when the user already knows the gesture for a command. Therefore pattern recognition and subsequent visual preview is only triggered if the user moves the pen slower than an empirically determined threshold. In the implemented system any movement slower than 2.5 letter keys per second triggers pattern recognition and visual preview. This check is performed by the system every 20 ms (50 Hz).

#### 5.4.4 Quick Pick

Any commands shown in the alternative list to the left of the center panel (Figure 5.10) can be directly selected by quickly dragging the pen towards the command name. The system can unambiguously separate pen-gestures from alternative list selections since once the pen tip leaves the keyboard area the articulation does not constitute a valid pen-gesture. Since movement dynamics is taken into account when deciding if pattern recognition and update of the preview should be performed, the alternative list will not suddenly change despite the user gesturing over the keyboard

while heading towards the desired command. This functionality worked very well in practice, as shown in the results later.

An important aspect of all user interfaces is behavioral consistency. For this reason it is also possible to “quick pick” the currently previewed command (e.g. *Cut* in Figure 5.10 top). In other words, either lifting the pen from its current location or dragging the pen to the preview box results in the same command.

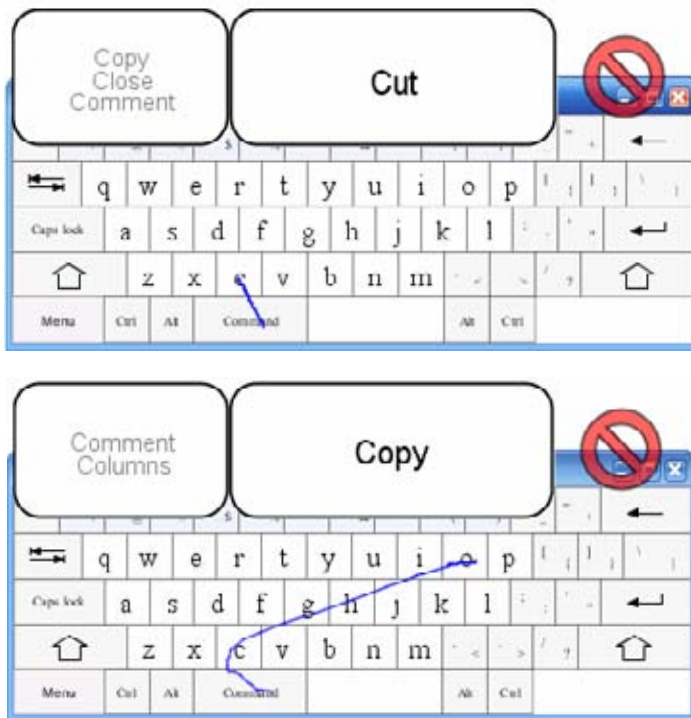


Figure 5.10. When the user gestures *Command-c* the command *Cut* is previewed (top). When the user gestures *Command-c-o* *Copy* is previewed (bottom).

#### 5.4.5 Disambiguation

Some applications have many commands that start with the same letters. For instance, in MS Word *Cut*, *Copy* and *Comment* start with the letter *c*. When the system detects such a conflict a disambiguation procedure is invoked. First the system checks if any command has priority. For instance, if *Copy* is used more in the active application than *Cut* (as determined by the user’s actions in the system) then *Copy* will be shown as the primary command previewed. If two commands have been used the same number of times the system prefers the shorter command, in this case *Cut*.

## 5.4.6 Conceptual Advantages

### 5.4.6.1 *What You See Is What You Get*

Preview allows the user to be certain about what command will be executed. This alleviates the fear of unintentionally invoking irreversible commands.

### 5.4.6.2 *Minimizes User Effort*

For many commands, one or two letters in a command would uniquely differentiate it from other commands. However without preview it is impossible for the user to discover the minimum number of letter keys needed for a given command without trial-and-error. With preview the user can learn the shortest path needed without the frustration of trial-and-error, hence the amount of motor effort needed in articulating the gestures is reduced. Command stroke with preview therefore gracefully and flexibly merges short and long command strokes.

### 5.4.6.3 *Encourages Exploration*

Without preview the user would need to try out commands to see if they existed in the active application. With preview frequently a command will appear either in the preview or in the alternative list after only one or two letters of the command is gestured. For instance, if the user gestures *Command-s* the command *Save* is likely shown in preview or in the alternative list along with the other best matching commands. If it is not, the user can continue gesturing on the path *Command-s-a-v-e* and the system will display *Save* in preview or in the alternative list at some point as long as *Save* is a valid command. If *Save* is still not displayed when the complete path of *Command-s-a-v-e* is finished, the user would know that *Save* is not available in this particular application. During this exploration the user can always stroke to the cancellation icon to abort the procedure and prevent an unintended command (if currently previewed) from being executed.

### 5.4.6.4 *Reveals the Space of Possible Commands*

A problem with the first iteration of command strokes is discovery support: how does a user know the space of available commands except by consulting co-existing pull-down menus (which most likely exist due to legacy compatibility) or the application help system first? With preview, some commands other than the intended one will inevitably be shown in the preview or the alternative list while the user is gesturing. This increases the likelihood that the user notices other available commands. For instance, if *Print* is gestured as the sequence *Command-p-r*, *Properties* might appear as the next best matching command, informing the user that this command is available in the active application.

### 5.4.6.5 *Benefits the Novice and the Expert*

Since command stroke with preview considers the movement dynamics of pen-gesturing, a true expert that knows how to quickly articulate the commands will not be disrupted with any visual feedback. If a user writes an unfamiliar command the

natural slowdown of the pen motion causes preview to be automatically displayed, aiding novices and experts alike on just how much of the command stroke path needs to be completed for the command to be accurately recognized. Furthermore, if an expert user is very certain of a command stroke, the preview display can be ignored, reverting back to basic command strokes behavior.

## 5.5 Experiment 5.2: Preview Performance

---

While there are many conceptual advantages with preview as outlined above which motivated the development of the technique, there are also concerns of possible adverse effects of preview. For example the display of preview could be distracting or over-demanding on the user's visual attention and therefore significantly slow down input speed and / or inflicting a higher error rate. Clearly an empirical study was needed to reveal any measurable performance impact of preview, particularly an adverse performance impact if any. Note that not all of the conceptual advantages outlined above, such as the ability to explore and discover, should necessarily result in measurable performance differences.

### 5.5.1 Procedure and Design

16 volunteers were recruited for this within-subject experiment. None of them had participated in Experiment 5.1. The experiment was carried out on a Fujitsu-Siemens tablet PC with a screen set to landscape orientation and with a screen resolution of  $1024 \times 768$  pixels. The QWERTY keyboard layout was used for the software keyboard.

In the experiment participants used a stylus to enter commands in one of two conditions:

1. No Preview. The participant entered commands with the preview interface (preview, alternative list and cancellation icon) disabled. When the participant lifted the pen the recognized command was displayed above the keyboard.
2. Preview. The participant entered commands with the full implementation of the preview interface as described earlier.

The experiment simulated realistic document editing in which commands were interleaved with common word processor (MS Word) tasks. The goal was to be able to observe and measure both speed and error of the two versions of command strokes. Two task scripts and instructions were developed so that the second condition would not repeat the same script. The system was made to work with real applications in MS Windows. For example, a part of the first script read: "Scroll down to the bottom of the document. Invoke *Paste*." If the participant correctly activated *Paste* the contents of the clipboard would be pasted into the active document. All 114 commands in MS Word 2002 available in the main pull-down menus (e.g. *Open*, *Paste*, etc.) and all other menu items that had an assigned hot key (e.g. *Thesaurus*, *Visual Basic Editor*, etc.) were implemented and could be recognized by the experimental system. 10

commands were used in the first script: *Open, Properties, Copy, Paste, Select All, Word Count, Date and Time, Undo, Track Changes, Print* and 10 in the second script: *New, Styles and Formatting, Symbol, Paragraph, Font, About Microsoft Word, Versions, Page Setup, Break, Close*.

If a user made a mistake in following the instructions, the user was asked to repair the mistake. For example, if the user accidentally executed the command *Styles and Formatting* to the word processor, the user was asked to close the panel that appeared.

After a brief demonstration of the system the participant was asked to follow one of the scripts with preview either disabled or enabled. After the script had been repeated 10 times, the participant was asked to follow a second script and test command strokes with the condition. The order of the two methods and the two scripts were balanced among the participants.

## 5.5.2 Results

### 5.5.2.1 Error

The average error rate with the *Preview* condition (7.5%) was lower than with *No Preview* (11.3%). However the difference was not statistically significant ( $F_{1, 15} = 2.0$ ,  $p = .178$ ).

### 5.5.2.2 Selection Time

Selection time was calculated as the time duration from pen-down to pen-up when articulating a correctly recognized command gesture. Repeated measures variance analysis showed that the difference in selection time was not statistically significant ( $F_{1, 15} = 0.207$ ,  $p = .656$ ), see Figure 5.11. There were considerable individual differences in performance. For instance, the fastest participant had an average selection time of 1126 ms with *Preview* and 1583 ms with *No Preview* while the slowest participant had an average selection time of 2451 ms with *Preview* and 3900 ms with *No Preview*.



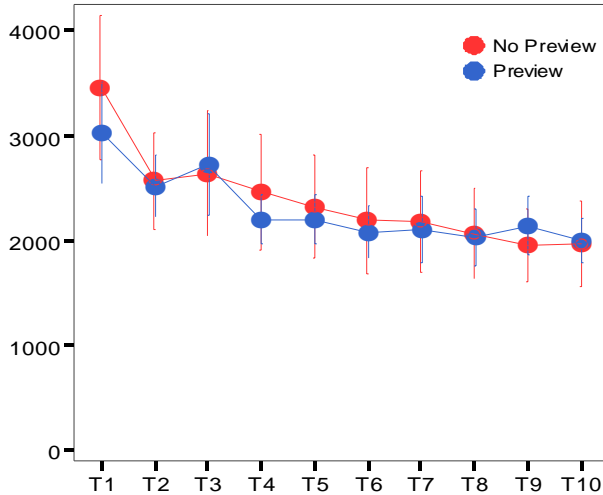


Figure 5.11. Mean and 95% confidence interval of selection time (ms) as a function of trial number.

#### 5.5.2.3 Trial Completion Time

Trial completion time was defined as the time taken to complete one repetition of one of the scripts with 10 commands. Average trial completion time was 105.2 seconds with the *No Preview* method and 105.6 seconds with *Preview*. The difference was not statistically significant ( $F_{1,15} = 0.004$ ,  $p = .952$ ).

#### 5.5.2.4 Trace Lengths

Trace length, the distance the pen traveled over the keyboard, was measured in multiples of key width. If only the minimum paths were articulated, the average trace length of the commands tested would be 9.0 keys. If the complete paths were gestured the average trace length of the tested commands would be 27.3 keys. The results showed that the traces in the *Preview* condition were significantly shorter (10.7 keys) than the traces in *No Preview* (15 crossed keys). The 40% difference was statistically significant ( $F_{1,14} = 31.7$ ,  $p < .0005$ ). Evidently participants took advantage of the visual feedback and did not over-specify the gestures much.

#### 5.5.2.5 Quick Pick

In *Preview* two participants used quick pick almost exclusively (96% and 93% of the responses respectively). One participant used quick pick  $\frac{1}{4}$  of the time. The other participants used quick pick considerably less (0-5%). Pearson's  $r$  showed no significant correlation between quick pick usage and error rate ( $r = -.139$ ,  $n = 16$ ,  $p = .608$ , two-tailed).

### 5.5.2.6 User Ratings

After each condition participants were asked to rate their confidence on a 1-7 scale (1 = “Very unconfident”, 7 = “Very confident”), and after the experiment they were asked to rate their preference of each method on a 1-7 scale (1 = “Strongly dislike it”, 7 = “Strongly prefer it”). Friedman’s repeated measures non-parametric test showed that neither confidence ( $\chi^2 = 2.273$ ,  $df = 1$ ,  $p = .132$ ) nor preference ( $\chi^2 = 2.571$ ,  $df = 1$ ,  $p = .109$ ) varied significantly between the methods.

The comments from the participants in the study were positive towards both interfaces. One participant declared “Wow! This is amazing! How can it know what I want to write?” One participant that really liked the preview version stated that “without preview I felt unsure if I was doing the right thing. With it enabled I felt I was guided [by it]” (translated from Swedish).

In summary, Experiment 5.2 did not show any adverse effect with command stroke with preview. It also revealed that participants could take advantage of some features of command stroke with preview, such as using quick pick and taking a shorter stroke path due to the guidance of the preview display. On the other hand, when the same task procedure was repeated in succession as in this experiment, hence intensifying the participants’ familiarity with the sets of command strokes to an “expert” level, neither speed nor accuracy was significantly different between the two conditions. A hypothesis is that users might take greater advantage of the preview functions when they encounter commands that are new or unfamiliar, which frequently happens in a real use situation.

## 5.6 Experiment 5.3: New Commands

---

A follow-up study was conducted with the same participants as in Experiment 5.2 to investigate how users familiar with the technique tackle new commands they have not previously gestured – with and without preview.

### 5.6.1 Procedure and Design

The same 16 participants in Experiment 5.2 were recruited the week after to take part in Experiment 5.3. They were asked to enter commands in one of two conditions: *Preview* or *No Preview* with the same properties as described in Experiment 5.2.

Two sets of 10 commands were used. The sets of commands and the experimental order of the two conditions were balanced. The commands were randomly chosen from MS Word with the constraint that they had not been previously used in Experiment 2. The commands in the first set were: *Fullscreen*, *Office Clipboard*, *Table AutoFormat*, *Macros*, *Theme*, *Text Box*, *Borders and Shading*, *Drop Cap*, *Word Perfect Help*, *Ruler*, and in the second set: *Object*, *Thesaurus*, *Draw Table*, *Reveal Formatting*, *Office on the Web*, *Heading Rows Repeat*, *Hide Gridlines*, *Bullets and Numbering*, *Find*, *Paste as Hyperlink*.

In one condition the participants entered all commands from a first set of commands and repeated the set once again. If a mistake was made the participants were asked to try again. Next the procedure was repeated in the other condition with the second set of commands.

## 5.6.2 Results

### 5.6.2.1 Error

Error rate differed dramatically between the conditions with average error rates as low as 1% in both trials in the *Preview* condition (Figure 5.12) but on average 10.5% in the *No Preview* condition. Repeated measures variance analysis showed that the difference in error rates between the conditions were statistically significant in the first trial ( $F_{1, 15} = 8.99$ ,  $p < .01$ ) as well as in the second ( $F_{1, 15} = 9.22$ ,  $p < .01$ ). The results show that users benefited from preview when executing unfamiliar command strokes.

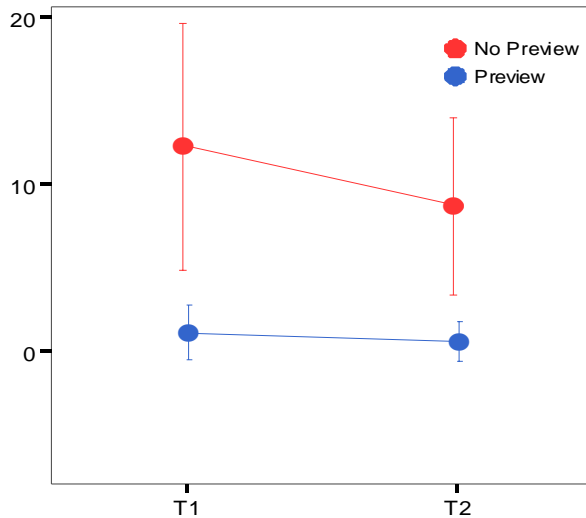


Figure 5.12. Mean and 95% confidence interval of error rate (%) as a function of trial number.

### 5.6.2.2 Selection Time

There was no significant difference found in selection time (as defined in Experiment 5.1) in either trial 1 ( $F_{1, 15} = 2.815$ ,  $p = .114$ ) or trial 2 ( $F_{1, 15} = 3.624$ ,  $p = .076$ ), see Figure 5.13. The slower selection times compared to the ones in Experiment 5.1 and Experiment 5.2 are not surprising given that the majority of the commands tested in this experiment were longer and less frequently used.

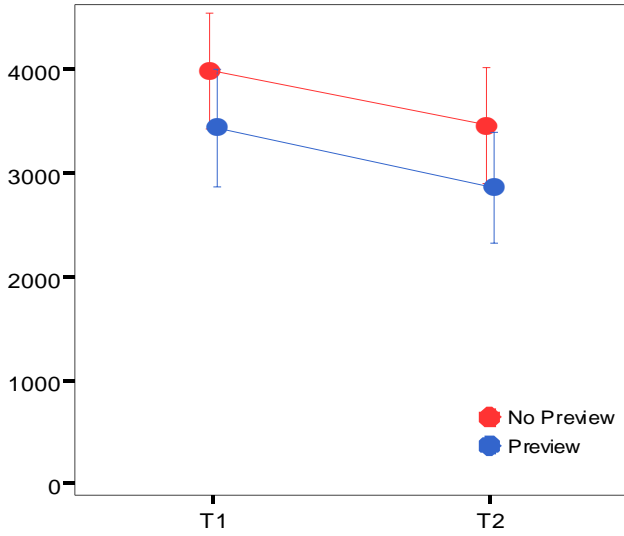


Figure 5.13. Mean and 95% confidence interval of selection time (ms) as a function of trial number.

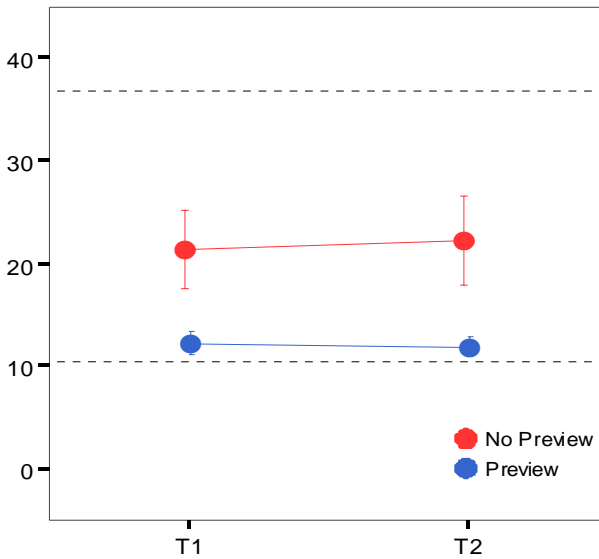


Figure 5.14. Mean and 95% confidence interval of trace length (in key widths) as a function of trial number. The dashed bottom and top reference lines indicate the minimum and maximum possible trace length.

### 5.6.2.3 Trace Lengths

The trace lengths (as defined in Experiment 5.2) in *Preview* condition were significantly shorter than in *No Preview* condition in both trial 1 ( $F_{1, 15} = 25.043, p$

<.0005) and trial 2 ( $F_{1,15} = 25.671$ ,  $p <.0005$ ), see Figure 5.14. Clearly visual preview aided the participants in gesturing shorter command strokes.

#### 5.6.2.4 User Ratings

After Experiment 5.3 the participants were asked which method they preferred and why. All participants preferred the *Preview* method. When asked to explain their preference all participants stated that with the *Preview* method they knew when they could stop and lift up the pen. One participant stated that dyslexia made it difficult for him to spell out the commands without visual guidance.

In summary, Experiment 5.3 clearly demonstrated the advantage of command stroke with preview when dealing with unfamiliar commands, as indicated by shorter stroke path and higher accuracy.

## 5.7 Discussion

---

Experiment 5.1 showed that both short and long command strokes are significantly faster than pull-down menus. Between the two, short command strokes were faster but more error prone than long command strokes. Short command strokes based on traditional keyboard shortcuts are only applicable to menu items that have a keyboard shortcut assigned, whereas long command strokes can be applied to all commands in an application. Long command strokes should also be easier to remember and use, since they are based on the name of the command instead of a randomly assigned keyboard sequence such as *Ctrl+Y* for *Redo*.

With the preview display, command stroke with preview simultaneously takes advantage of both short and long command strokes. Command stroke with preview is still based on command name traces on the keyboard, but with the visual preview the user knows how much of the entire trace needs to be gestured for a command to be recognized. Experiments 5.2 and 5.3 showed that users' trace lengths were significantly shorter when using preview (also Figure 5.14).

A critical concern with visual preview is whether it demands so much visual attention that it interferes with expert users' speed performance. Experiment 5.2 showed that the command stroke with preview mechanism, which was carefully designed not to distract fast users, did not impede users' performance for well practiced familiar commands. For unfamiliar commands tested in Experiment 5.3 the visual preview did not only leave participants' speed performance unchanged, but also significantly reduced error rates. Overall, the experiments show that visual preview has real significant benefits and does not slow down input speed.

## 5.8 Related Work

---

Many researchers have previously tackled the pen-based command selection problem. For instance, Kobayashi and Igarashi [2003] demonstrated a technique that makes

pull-down menu traversal easier when navigating through sub-menus, and Kurtenbach, Fitzmaurice, Owen and Baudel [1999] have developed a technique called Hotbox which combines linear, radial and pop-up menus to create a graphical user interface that can handle a large number of commands for the Maya modeling application.

Pie menus have been demonstrated as a competitive alternative to pull-down linear menus [Callahan et al., 1988]. Marking menus, studied and advocated by Kurtenbach, Sellen and Buxton [1993], further improve pie menus. Embodying a critical thought in user interface design, marking menus are pie menus augmented with a pen-gesture recognizer that encourages behavior transition from novice to expert use. Novice users select items in the pie menu structure as if using a regular pie menu (with delayed display). Over time, users learn the angular pen-gestures for selecting a command. This allows expert users to quickly flick the pen-gesture of the command without needing to visually traverse a pie menu hierarchy. To encourage users to learn the fast mode of gesturing commands instead of using slower visual feedback-based navigation, marking menus do not “pop-up” until after a time delay. A problem with marking menus observed by Zhao and Balakrishnan [2004] is that some selections are ambiguous when marks are articulated independent of scale. Zhao and Balakrishnan [2004] investigated the use of consecutive single line marks instead of compound marks in marking menus and found that single line marks are unambiguous and more compact. However, it remains an open question whether users will memorize a sequence of disconnected single line marks as easily as compound marks that can be perceived and remembered as a whole. Another variant of the marking menu is FlowMenu [Guimbretière and Winograd, 2000]. Originally made for wall-sized displays, FlowMenu combines the Quikwriting text entry method [Perlin, 1998] with marking menus. Although a practical method for many specialized applications, the down-side of the method is that the pen-gestures are long and complicated.

The contrast between command strokes and marking menus is interesting and multifaceted. First, marking menus replace linear menus while command strokes are designed to co-exist with or complement linear menus. With command strokes interfaces, the user can use the traditional pull-down menus to explore the existence of certain functions or commands (although with command stroke with preview one can also often discover command strokes by trying out tracing the letters of a command and observe the visual preview) but rely on command strokes to efficiently evoke known commands. Second, the learning mechanisms in the two systems are different. In command strokes the user incrementally learns the pen-gestures on the software keyboard with use, starting by slowly tracing the keys with visual guidance then, over time, gradually (partially) transitioning into open-loop gesturing by recall. The software keyboard is always present as a visual map. In contrast, using the marking menu novice-expert bridging technique, the transition is binary – either selecting with the displayed (and delayed) menu hierarchy or gesturing the mark without any visual

assistance. This may force the user to memorize the gesture faster at the cost of reduced novice performance due to the delay. On the other hand, marking menus have a relatively easy transition from browsing commands via the displayed menu to selecting commands by pen-gestures. In this regard, command strokes requires an additional conscious step by the user to transition from menu browsing to pen-gesturing command strokes, although the visual preview system in command stroke with preview may encourage exploration.

In comparison to all menu selection methods, one advantage of command strokes is consistency. Commands tend to be named in a similar manner in all user interfaces but are often placed differently in menu structures, shifting locations from application to application. Hotkeys for the same function can also vary between applications. The flat hierarchy with command strokes allows hundreds of commands to be specified directly by the user rather than being accessed from browsing a hierarchy. Large capacity in a small space is another advantage with command strokes. Since a stylus keyboard does not require much screen estate, command strokes can be used in specialized domains, such as Unmanned Aerial Vehicle (UAV) control [Quigley, Goodrich and Beard, 2004], where a large number of commands need to be entered on a size-constrained handheld computer.

Another class of command articulation techniques is free-form pen-gestures, such as the Rubine recognizer [Rubine, 1993]. Free-form pen-gestures are often “arbitrary” pen-gestures that denote different actions in a user interface. The similarity aspects of pen-gestures have also been studied [Long, Landay, Rowe and Michiels, 2000]. In general, free form pen-gestures are limited in the number a user can remember and reproduce, due to their arbitrary definition to the user. In contrast to free-form pen-gestures, command strokes use the keyboard as a mnemonic map.

It is clear that each technique has its own pros and cons. Users are familiar with pull-down menus and they are backwards compatible with virtually all existing desktop and mobile programs. Special techniques such as the Hotbox [Kurtenbach et al., 1999] can be used when an application needs to provide access to over 1,000 commands but it takes a large amount of screen space. Free-form pen-gestures are most advantageous in application domains that have a strong convention, such as copy editing, or in places where pen-gestures can be highly metaphorical, such as crossing a series of words to cause the words to be deleted.

## 5.9 Conclusions

---

Based on the conceptual analysis and the empirical studies presented in this chapter, it is expected that command strokes can be a very useful complement to pull-down menus in future mobile computing devices. Pull-down menus tend to be slow and tedious, but offer an effective way for the user to discover the commands available in an application. Users will continue to use pull-down menus to access a large number

of infrequent functions. For a known command, particular frequently used commands such as *Cut*, *Copy*, *Paste*, *Find*, and *Print*, Command strokes provide fast, fluid and convenient access when used in conjunction with a software keyboard. Experiment 5.1 shows that on average command strokes can be much faster than pull-down menus, particularly if the commands are located in sub-menus. In the first experiment a speed-accuracy trade-off was found between long and short command strokes. The latter is faster but more error prone. Command strokes were further developed with preview which conceptually could improve certainty, reduce effort, and encourage exploration and command discovery. Empirically command stroke with preview proved capable of reducing users' gesture lengths without impacting speed or accuracy for familiar command strokes (Experiment 5.2) For unfamiliar commands, preview dramatically reduced error rate (Experiment 5.3). These results indicate that as long as the input interface does not force users to look at the visual feedback, high input speed and low error rates can be obtained even though visual feedback is introduced that guides novice users towards the expert mode.



# Chapter 6

## Design Dimensions of Mobile Text Entry

This chapter serves as a literature review typically found in a doctoral dissertation. This review is usually placed in the beginning of a dissertation. However, the area of text entry is so diverse in conceptualization and as a result the traditional order of a literature review would be distracting to the flow of this thesis. Note also that each individual chapter has reviewed prior work most relevant to the build-up of the content to that chapter,

Furthermore, this chapter is also more than a literature review in that it aims at outlining the design dimensions of mobile text entry. It identifies and analyzes the most important design dimensions in mobile text entry methods. The design dimensions selected are: entry rate, error, learning curve, immediate efficacy, form factor, preparation time, localization, comfort, user engagement, visual attention, cognitive resources, privacy, single vs. multi-character entry, specification vs. navigation, one-handed vs. two-handed, task integration, robustness, device independence, computational demands, manufacturing cost, support cost and market acceptance.

The selection of and these design dimensions grew out of an extensive literature review and personal experiences in developing and analyzing text entry methods. The goal is to understand the entire space in mobile text entry method. Of course, this is impossible to achieve and I have almost certainly missed some design dimensions crucial for future text entry inventions. However, by an honest attempt in distilling the design dimensions relevant to mobile text entry as we know it today, the aim is to let the readers with varying degree of expertise in the area a sense of wide spectrum of requirements, limitations and tradeoffs that are explicitly or implicitly considered by mobile text entry method researchers and developers today. By putting all design dimensions involved in the spotlight my hope is to provide a framework that both guides the conception of new text entry methods, and help the research community in judging them for their merits.

An overview of existent mobile text entry approaches and an analysis of their typical performance are necessary to motivate and illustrate these design dimensions. Therefore this chapter begins with a survey of mobile text entry methods. There already exists surveys on mobile text entry (e.g. [MacKenzie and Soukoreff, 2002a; Isokoski, 2004a; Wobbrock, 2006]). In relation to these, this chapter's survey is

primarily designed to introduce a context for a discussion and analysis of the general design dimensions of mobile text entry that are often implicit in the body of previous text entry research.

The remainder of this chapter is structured as follows. The first section introduces basic concepts used in mobile text entry research. The second section surveys a wide array of mobile text entry methods. The third section describes and analyzes the design dimensions of mobile text entry. The fourth section selects a subset of mobile text entry methods and compares them against each other from the perspectives of the design dimensions that were previously described and analyzed. The last section concludes.

## 6.1 Introduction

---

The metrics most commonly used in the literature are entry rate (also referred to as input rate, text entry speed, text entry speed, input speed, speed) and error rate (also referred to as error, or accuracy [accuracy = 1 - error rate]). I have opted to try to list both novice and expert users' entry rates if available. Of course, both the concepts of a novice and an expert are relative. For example, operationally, novice entry rate can be defined as the entry rate achievable within 2-35 minutes by participants in a controlled experiment, whereas expert entry rate can be defined as the rate achievable after several sessions. The definitions for novice and expert entry rates are deliberately loose because there has been no consensus in the literature on experimental setup and procedure in text entry evaluation. Also note that the term error rate has different meanings in different research papers.

Text input performance evaluation inevitably involves empirical experiments. Text entry research experiments can be divided into three common classes of experimental setups.

In the first experimental setup, participants are not allowed to make any errors, i.e. participants are forced to write a completely correct sentence in order to proceed to the next. In this case the term error rate is unambiguous and refers to the number of errors participants made and corrected in the text entry process. Therefore this error rate unambiguously refers to corrected errors. In this chapter, if researchers who have conducted this type of experiment report an error rate it will be clearly marked as an error rate related to the number of corrected errors.

In the second experimental setup, participants are allowed to make errors but explicitly denied to the ability to correct their errors. Here error rate unambiguously refers to uncorrected errors. This is a problematic choice of experimental setup because entry rate is inflated by uncorrected errors and there is no telling how many errors participants would choose to correct if given the chance.

In the third experimental setup, participants are allowed to make errors but are given a chance to correct their errors. In this case the term error rate can mean two different things because there are two categories of errors: corrected errors that the participants discovered and corrected, and uncorrected errors that the participants did not discover or made an effort to correct. The two types of errors have different or even opposite effects on entry rate. As a speed-accuracy trade-off, one may gain higher entry rate by making more errors and leaving them uncorrected. The corrected errors, however, are subsumed in the entry rate because the time participants were required to devote to correct these errors decreased the entry rate. However, some researchers report error rate with this experimental setup as “total error rate” [Soukoreff and MacKenzie, 2003b]. Total error rate combines both corrected and uncorrected errors into one error rate. By doing so the error rate becomes an uninformative number in relation to entry rate because it is impossible to know from the total error rate if entry rate was inflated at the cost of an increased number of uncorrected errors. The opposite effect is also possible, entry rate can be held back by users correcting many errors.

It is methodologically more principled to force participants to correct all errors to proceed. By forcing users to correct errors, entry rates obtained with different text entry methods can be directly compared because error correction is subsumed into entry rate. Intuitively, such an approach also makes sense because in reality errors are unavoidable and error correction is an integral part of the text entry process (in fact a good text entry system should make error-correction as easy as possible). However, psychologically it can be frustrating to users to be forced to correct all errors and it has often been criticized as “artificial”. This is probably the reason most text entry research experiments do allow errors.

If some errors are left uncorrected, it is unclear how error and entry rate are related in terms of a speed-accuracy tradeoff curve. In fact, the shape of the speed-accuracy tradeoff curve in text entry is currently unknown and could vary significantly between different text entry methods. This means that two methods A and B with simultaneously different entry rate and error rate cannot be easily evaluated. Suppose A has a higher entry rate and a higher error rate than B. It is not possible to know if A would still have a higher entry rate if its error rate were held the same as B’s error rate. Text entry is an extremely complex process to study through controlled experiments. As a comparison, the much simpler process of pointing modeled by Fitts’ law has a speed-accuracy tradeoff curve that is still debated in the literature [Zhai, Kong and Ren, 2004]. For these reasons it may well be necessary for all text entry experiments to adopt the first type of set-up (all errors are highlighted and have to be corrected to proceed) so their results can be compared in the future. However the research field has not come to an agreement on this matter.

Finally, many factors other factors also affect the entry and error rates reported from an experiment. Factors include participants’ age, motor skill [Rosenbaum, 1991],

motivation, the mechanical properties of the device used, text used, the instructions given to the participants, users' native language [Isokoski and Linden, 2004] etc. For these reasons text entry methods' entry and error rates, unfortunately, are typically not comparable unless the entry and error rates were measured in the same controlled experiment.

## 6.2 Overview of Mobile Text Entry Techniques

---

This section surveys exiting text entry methods. They have been broadly partitioned into the following categories: physical keyboards, projection keyboards, tilt-based text entry, software keyboards, handwriting recognition, speech recognition, prediction, abbreviation, artificial alphabets, word-level single-stroke encoding, and hierarchic text entry methods.

Because there is an enormous amount of mobile text entry methods proposed this section will only list a select subset and makes no effort to be complete.

From here on, in this overview, the unqualified term error rate means the uncorrected error rate

### 6.2.1 Physical Keyboards

#### 6.2.1.1 *Foldable Keyboards*

A foldable keyboard (e.g. [Coulon and Malhi, 1996]) is a portable version of a desktop keyboard (typically using the de-facto standard QWERTY layout) that can be folded into two or three pieces. A foldable keyboard has capability to yield the same performance as a desktop keyboard – at the cost of portability and form factor. A foldable keyboard also requires the user being able to sit down next to a table or other flat surface in order to be effective. To my knowledge no empirical investigations of foldable keyboards have been carried out. An advantage with foldable keyboards is that users are probably already familiar with QWERTY touch-typing. On the other hand, users who cannot touch type on QWERTY only reach 20 wpm after 12 hours of practice [Noyes, 1983].

#### 6.2.1.2 *Stick Keyboards*

Green, Kruger, Faldu and St. Amant [2004] presents a “stick keyboard” where the standard QWERTY keyboard is compressed into a single home row. E.g. the letter key *A* contains all the *Q/A/Z* keys. To disambiguate between the possible letters each key users can either type the key multiple times to cycle through the letters, or activate lexicon lookup mechanism where the keyboard attempts to disambiguate among the letter key combinations. If the desired letter does not show up the user needs to navigate among the next best choices with dedicated buttons. Novice users quickly reach around 22.5 wpm with the “stick keyboard”, reaching around 40% of their desktop keyboard speed, which was used as a baseline reference [Green et al., 2004]. Error rates were not reported.

### 6.2.1.3 *Chording Keyboards*

Chording is a technique that several keys can be depressed simultaneously to yield characters (similar to the chords when playing piano). In an influential work on chording Gopher and Rajj [1988] indicates that a chording alternative to typewriting can result in faster skill acquisition than traditional QWERTY typing for users unskilled in touch-typing.

For mobile text entry the half-QWERTY keyboard [Matias, MacKenzie and Buxton, 1996] is a straight-forward implementation of a minimal chording keyboard. The traditional QWERTY keyboard is split in half hence reducing the physical size of the keyboard by a factor of 2. To access keys on the left hand side of a regular keyboard the user presses the keys on the half-QWERTY as usual. To access the keys on the right hand side of a regular keyboard the user presses the space bar while simultaneously pressing the intended letter key. Matias et al. [1996] reports that after a 50-minute session novice users typed with an average entry rate of 13.2 wpm. Matias et al. [1996] allowed errors but participants were not allowed to correct errors. The uncorrected error rate was 15.16%. After 10 50-minute sessions participants reached an average entry rate of 34.7 wpm and an average uncorrected error rate of 7.36%.

Another chording keyboard is twiddler. Twiddler is held with the user's dominant hand. The user enters text by pressing one or more keys simultaneously using the fingers on the dominant hand. Lyons, Starner and Gane [2006] shows novice users reaches around 6 wpm with a 10% error rate after 90 minutes of typing. After 13 hours of practice users reached an average entry rate of 37.3 wpm and 6.2% average error rate. Note that the error rates reported are "total error rates" [Soukoreff and MacKenzie, 2003b] that lumps together both uncorrected and corrected errors by users. Since the uncorrected error rate is not reported it is impossible to judge how much uncorrected error rates contributed to the entry rate.

Rosenberg and Slater [1999] and Mehring [2005] present variants of a chording keyboards designed for VR (virtual reality) environments. In both implementations the user wears special gloves or skeletal-structures on the hands.

In the system proposed by Rosenberg and Slater [1999] contact points are present on each finger tip. Modifier keys are present on the part of the index finger that is closest to the thumb. By pressing the thumb against the modifier keys the user can change the current key-assignment for the finger-tips' contact points. The user generates key strokes by pressing down finger tips simultaneously. After an initial practice session users typed with an average entry rate of 8.9 wpm and an average error rate of 27%. After 10 50-minute sessions the final average entry rate was 16.8 wpm and the final average error rate was 17.4%. It is unclear in Rosenberg and Slater [1999] whether error rate refers to corrected errors, uncorrected errors, or both.

In the system presented in [Mehring, 1999] each finger tip is equipped with an electronic contact point each, and the thumbs are equipped with 3 contact points. To type, the user moves a finger tip to contact with a contact point on the thumb (of the same hand). For instance, to type the letter *a* the user moves the tip of the left little finger to the center contact point of the left thumb. The mapping between finger tip and contact position on the thumb is designed to resemble the QWERTY keyboard layout to ease transfer of touch-typing skills. Kuester, Chen, Phair and Mehring [2005] presents the results of a preliminary evaluation of the method in comparison to touch-typing with a traditional keyboard. Average typing speed was 4 wpm (calculated from data reported in Section 5.2 in Kuester et al. [2005]), with a 79% error rate. It is unclear in Kuester et al. [2005] whether error rate refers to corrected errors, uncorrected errors, or both.

A technology that is very close to the same idea as what is presented in Mehring [2005] and Kuester et al. [2005] is the FJGK (finger-joint gesture keypad) [Goldstein, Baez and Danielsson, 2000]. With the FJGK the user can input keypad information (e.g. numeric input) by pressing the thumb tip against the 3 different segments on each finger. For instance, to type the number *1* the user presses the right thumb tip against the top segment of the right little finger. Goldstein, Baez and Danielsson [2000] reports an average recognition rate of 78% with the FJGK. To my knowledge no empirical evaluation of the technique's performance has been conducted.

#### 6.2.1.4 *Thumb Keyboards*

Thumb keyboards are essentially desktop keyboards shrunk down to fit the form factor of a handheld computer, smart phone or large conventional mobile phone (Figure 6.1). The device hosting the thumb keyboard is held with both hands. The user enters text by moving the thumbs concurrently on the left and right hand sides of the keyboard. From at least a strict performance point-of-view, there are reasons to believe that thumb keyboards can be quite effective for a skilled touch-typist. First and foremost the QWERTY keyboard distributes frequent letter pairs to the left and right hand side of the keyboard. With a thumb keyboard the user can move the second thumb into position concurrently, while the first thumb is pressing the first key in a letter pair. Second, novice users typing on a thumb keyboard benefit from skill transfer from touch-typing on a regular keyboard which should improve novice users' performance. On the other hand, the small packed keys may also feel difficult, uncomfortable or clumsy to type on.



Figure 6.1. A thumb keyboard. The keys have a diameter of 8 mm.

Theoretical thumb keyboard performance models based on concurrent movement of the left and the right thumb, Fitts' law [Fitts, 1954] and a character-level bigram model predict average expert entry rates of 60.51 and 60.74 wpm [Clarkson, Lyons, Clawson and Starner, 2007; MacKenzie and Soukoreff, 2002b]. In an empirical experiment novice users reached an average entry rate of 31.72 wpm and an average error rate of 6.12% after the first 20 minute session. After 20 20-minute sessions participants reached an average entry rate of 60.32 wpm and an average error rate of 8.32% [Clarkson, Clawson, Lyons and Starner, 2005]. However error rates were reported as "total error rate" [Soukoreff and MacKenzie, 2003b] that lumps together both corrected and uncorrected errors. Therefore it is impossible to know if uncorrected errors affected entry rate, and if so, to what degree.

There appears to be a relationship between the size of the thumb keyboard and entry rate. Clarkson et al. [2005] compared two different thumb keyboards of different sizes and found a statistical significant difference. The larger thumb keyboard was faster [Clarkson et al., 2005]. A related study by Curran, Woods and Riordan [2006] reports that novice users entering four phrases of varying difficulties had an average entry rate of 14.36 wpm for a large thumb keyboard attached to a flip-up personal organizer, and 8.05 wpm for a thumb keyboard attached to a smaller handheld computer. In a controlled study Roeber, Bacus and Tomasi [2003] found that participants had an average entry rate of 27.6 wpm and an average error rate of 2.2% when typing the pangram "The quick brown fox jumps over the lazy dog" repeatedly for 2 minutes. This is similar to the level of performance as reported in Experiment 3.2 in this dissertation, in which the participants typing on a thumb keyboard reached an average entry rate of 29.6 wpm and an error rate (uncorrected errors) of 1.3% after 35 minutes of practice, which is slower than Clarkson et al. [2005]. However, it is unclear in Roeber et al. [2003] whether error rate refers to corrected errors, uncorrected errors, or both.

In summary, thumb keyboards can be quite fast, at least based the experimental data typically contributed by young adults. Novice users can reach an average entry rate around 30 wpm. After many sessions of practice users can reach an average entry of

60 wpm [Clarkson et al., 2005]. Since Clarkson et al. [2005] only reports total error rate it is impossible know how high the uncorrected error rate was.

Thumb keyboards have the downside that they require a relative large portion of the mobile device devoted to typing. Persons with larger thumbs or reduced motor control ability may also have trouble with precise typing because the thumb can easily hit a key adjacent to the intended key by mistake.

#### 6.2.1.5 Keypads

Mobile phones usually have a physical telephone keypad in close proximity to the screen. Due to the small size of a mobile phone, and the legacy of touch-tone phones, the key set is reduced, see Figure 6.2. Since the mapping from the telephone keypad keys to the Latin alphabet is one-to-many an alternate mapping scheme between the user and the computer has to be used. The most common mapping scheme is known as multi-tap (sometimes called multi-press, e.g. in Isokoski [2004a]). To multi-tap the user repeatedly presses a keypad button to cycle through the valid key states for the button. For instance, to type the character *a* the user presses the 2ABC button once, to type *b* the user presses the 2ABC button two times and thereby cycles to next key state, from *a* to *b* (Figure 6.2). A result of this input scheme is that the user's intention when a key is pressed is ambiguous. A user can either intend to cycle to the next state of the key, or intend to input a new character. To disambiguate between these actions a timeout is used. If the user presses a key before the timeout, the key press is interpreted as an intention to cycle to the next key state. Otherwise, the key press is interpreted as inputting a new character.

Two factors limit multi-tap performance. First, multiple key presses are required to type a majority of the characters. Second, the timeout disambiguation mechanism forces users to wait unnecessary long to repeatedly input characters assigned to the same key. If a timeout is not used users must press an auxiliary key to signal that the next letter will be inputted.





Figure 6.2. The ISO/IEC 9995-8:1994 standard telephone keypad layout.

In addition, the standard telephone keypad is not designed to overcome these limitations because the characters are distributed among the keys based on the standard ISO/IEC 9995-8:1994 alphabetic convention (see Figure 6.2). Expert keypad performance can be theoretically modeled by a Fitts' law [1954] movement model and a character-level bigram model [Silfverberg, MacKenzie and Korhonen, 2000]. Silfverberg et al. [2000] estimates the theoretical expert speed of multi-tap to 25 wpm for one-handed thumb input and 27 wpm if the two index fingers are used. The estimate is made on the assumption that timeout is not used and users explicitly specify when the next letter will be inputted. If timeout is used the estimates should be reduced by 4 wpm [Silfverberg et al. 2000]. Pavlovych and Stuerzlinger [2004] presents a theoretical model of novice performance with multi-tap that predicts average novice entry rate to be 6.53 wpm. James and Reischel [2001] conducted an experiment where participants were divided into experts and novices depending on if they had used text messaging before or not. Participants wrote four sentences from newspaper corpora. Novice users reached an average text entry of 5.59 wpm with multi-tap while expert users reached an average text entry rate of 5.33 wpm. Unfortunately James and Reischel [2001] reports absolute errors and not error rates. This study was replicated by Butts and Cockburn [2002] who found that multi-tap with timeout resulted in an average text entry rate of 6.4 wpm, and multi-tap with a NEXT key instead of timeout resulted in an average text entry rate of 7.2 wpm. The error rate is not reported.

A variation of multi-tap is to re-arrange the letter ordering on each key to minimize the average number of key presses for words [Pavlovych and Stuerzlinger, 2003]. Pavlovych and Stuerzlinger [2004] estimates novices users' average text entry rate to be 6.53 wpm. In a controlled experiment comparing this method against multi-tap a 9.5% increase in entry rate was observed. Novice users typing for 20 minutes reached an average entry rate of 6.8 wpm with multi-tap in comparison to 7.2 wpm when letters were re-arranged on each key. Users that typed 40 minutes (grouped in 2 sessions) reached an average entry rate of 7.4 wpm with multi-tap in comparison to 8.0 wpm with letters were re-arranged (these numbers are estimated from Figure 6 in

[Pavlovych and Stuerzlinger, 2003]). Error rates for both techniques were consistently < 1% (see Figure 7 in [Pavlovych and Stuerzlinger, 2003] for details). It is unclear in Pavlovych and Stuerzlinger [2003] whether error rate refers to corrected errors, uncorrected errors, or both.

Another variation of multi-tap is to re-arrange the distribution of letters on the keypad while maintaining alphabetic ordering. For example, the keypad key 5JKL can be changed to 5MN. Gong and Tarasewich [2005] uses a computational model to derive such a keypad design that minimizes the average number of key presses required to compose words. Their evaluation shows that novice users that reached an average entry rate of 7.89 wpm with a re-distributed keypad in comparison to 8.22 wpm for a traditional keypad after one session. It should be noted that in this comparison the novice users had already been exposed to typing with re-distributed keypad in one earlier session (comparing two different designs of re-distributed keypads). Unfortunately the time durations for the sessions are not reported in Gong and Tarasewich [2005]. The errors are reported in a non-standard format as the number of times the user pressed the BACKSPACE key per sentence. Novice users pressed backspace 0.47 times when using a re-distributed keypad and 0.42 times when using a traditional keypad.

Smith and Goodwin [1971] realized that for many words multi-tap is highly redundant. Not all key combinations on a keypad are legitimate words in a language. The few letter combinations that do form valid words can be captured in a dictionary. For instance by typing the keys 2ABC, 6MNO and 3DEF in sequence, the system can recognize that among the possible letter combinations (for example *amd*, *and*, *anf*, *bmd*, etc.) that the word *and* is most likely the intended word. Such a system is called a dictionary-based disambiguation method.

Several methods and strategies have been proposed in the literature. Smith and Goodwin [1971] propose the perhaps earliest known method. Other methods such as Tegic Communication's (now a subsidiary of America Online Inc.) T9, ezText and Motorola's iTap have been licensed to mobile phone manufacturers and used in commercial products. Silfverberg et al. [2000] estimates expert performance to 41 wpm when a single thumb is used, and 46 wpm if both index fingers are used. However this assumption is based on a perfect disambiguation algorithm, which is never the case in practice. The unavoidable error correction process involved in the input process is not modeled or taken into account. Pavlovych and Stuerzlinger [2004]'s model predicts novices to reach an average entry rate of 7.58 wpm. James and Reischel [2001] conducted an experiment where participants were divided into experts and novices depending on if they had been text messaging before or not. Participants wrote four sentences from newspaper corpora. Novice users reached an average text entry of 7.21 wpm while expert users reached an average text entry rate

of 15.0 wpm. Unfortunately James and Reischel [2001] reports absolute errors and not error rates.

LetterWise [Gutowitz, 2001] is another alternative to multi-tap. LetterWise uses a statistical model to predict the user's intended letter when the user presses a key. The statistical model is composed of a set of prefix substrings of the most common words in the language. The difference between LetterWise and dictionary-based methods is that LetterWise does not rely on a dictionary which means the user can always type a word even if the word is not explicitly modeled by the language model. LetterWise attempts to guess the current key press immediately, in comparison to dictionary-based methods such as T9 that treats each key press as a step towards the completion of a word. If the user does not see the intended letter with LetterWise the user presses a NEXT key until it appears. In an empirical experiment, participants typing with LetterWise achieved 7.3 wpm in comparison to 7.2 wpm with multi-tap after 25-30 minutes [MacKenzie, Kober, Smith, Jones and Skepner, 2001]. The error rate was 7.6% for LetterWise and 5.5% for multi-tap (these numbers are estimated from Figure 7 in MacKenzie et al. [2001]). After 20 25-30 minute sessions participants reached 21 wpm with LetterWise in comparison to 15.5 wpm with multi-tap. The uncorrected error rate after session 20 was 6% for LetterWise and 4.2% for multi-tap (these numbers are estimated from Figure 7 in MacKenzie et al. [2001]). Participants were not allowed to correct their errors in MacKenzie et al. [2001]. A possible downside of LetterWise is the unintuitive process that decides which letter will be outputted when the user presses a key. Because the letter that will be outputted depends on previous writing the letter ordering on the keys no longer match. Gong, Haggerty and Tarasewich [2005] presents an extension of LetterWise where the next letter key is highlighted on the keyboard (implemented as a software keypad). Gong et al. [2005] reported error rate in a non-standard format as the number of times participants pressed the BACKSPACE key. Their evaluation shows a significant 42% reduction in error rate (BACKSPACE key presses) in comparison to LetterWise without highlighting [Gong, Haggerty and Tarasewich, 2006].

Another method of text entry using key pads is known as the "island technique" [MacKenzie and Soukoreff, 2002a]. The process of entering a letter using the island technique is partitioned into two separate steps. One step is a selection step where the user presses the key that contains the intended letter (among others). The other step is a disambiguation step where the user presses a key to indicate the specific letter intended.

The implementation and order of these steps can be varied according to several different strategies. In one strategy the letter groups are defined by partitioning the letters according to their ordinal position on the keys [Detweiler, Schumacher and Gattuso, 1990]. Using this strategy the first input step is to press the key that contains the desired letter. The second input step is to press a key that specifies the ordinal

position of the intended letter. For example, the top row with the 1, 2ABC and 3DEF keys specifies the first, second and third ordinal position respectively. As an example, pressing the key sequence 2ABC 4GHI results in the letter *h*. Other variants can be easily derived. For example, the same row that is used in the specification step can be used in the disambiguation step, instead of the top row [Detweiler et al., 1990]. Another variation is to reverse the disambiguation and specification step. Yet another variation is to complement the island technique with visual preview [Ingmarsson, Dinka and Zhai, 2004].

In comparison to multi-tap where the number of required key presses for a letter varies between 1 and 4, the island technique has a uniform distribution of two key presses per letter. Therefore the delimitation of separate intended letters is unambiguous to the system. In comparison to multi-tap there is no need for a timeout or an explicit delimitation button.

In a study examining novice users' typing performance for text entry on a television set Marshall, Foster and Jack [2001] found that when users' were asked to type a few short messages, multi-tap was on average faster than any island technique variation while maintaining the same error rate. Note that their study was very brief, encompassing only a few sentences. Butts and Cockburn [2002] found that when participants were asked to type 5 sentences from [James and Reischel, 2001] the island technique resulted in an average entry rate of 5.5 wpm. Errors were not reported in [Butts and Cockburn, 2002]. In a comprehensive study of the visual island technique [Ingmarsson, Dinka and Zhai, 2004] as a text entry method for television sets, novice users reached an average entry rate of 17 wpm after 10 45-minute sessions [Ingmarsson, Dinka and Zhai, 2004]. In Ingmarsson et al. [2004] participants were not allowed to correct their errors. Silfverberg et al. [2000] estimates that an expert user can expect an entry rate 22 wpm when using the island technique with a single thumb, and 25 wpm using both index fingers.

Levy [2002] presents a technique called Fastap. With Fastap an extra layer of physical keys are superimposed onto the traditional keypad. The second keyboard consists of small raised keys along the corners of the regular keys. To input a letter the user presses the raised small keys next to the regular key. For example, to enter the letter *d* the user presses a small raised key next to the key 2ABC. Cockburn and Siresena [2003] shows that Fastap can be used without training, and is faster than T9 for novice users. For expert users the performance is about the same as T9, with an average entry rate of 10.8 for T9 in comparison to an average entry rate of 9.8 wpm for Fastap. As a reference point, multi-tap was found to be much slower with an average entry rate of 5.6 wpm. The error rates were not reported.

Wigdor and Balakrishnan [2004] presents a chording method for keypads. The device is equipped with three additional keys on the back. When the user enters a letter the

user simultaneously presses a key on the back of the device and a key on the keypad. E.g. to write the letter *a* on the 2ABC key the user simultaneously presses the topmost key on the back of device together with the 2ABC key. After 20 minutes of practice participants reached an average entry rate of 11.8 wpm with the chording method, in comparison to 9.2 wpm with multi-tap (these numbers are derived from Figure 3 in Wigdor and Balakrishnan [2004]; the multi-tap figure refers to the two-handed multi-tap method). After 150 minutes of practice participants reached an average entry rate of 16.06 wpm with the chording method in comparison to 12.04 wpm with multi-tap. In the experiment participants were required to correct errors to proceed.

Keypads with a limited amount of keys have also been investigated (e.g. [MacKenzie, 2002]). Wobbrock, Myers and Rothrock [2006a] presents an input method where four keys are arranged in a square. To enter a letter the user presses a series of keys in sequence. An empirical experiment in Wobbrock et al. [2006a] shows that initial entry rate is on average 7 wpm. After 10 sessions the average entry rate was 15.5. These numbers are estimated from Figure 5 in Wobbrock et al. [2006a]). The grand mean error rate was 1.7%. In each session the participant typed ten phrases for each method investigated. However the two initial phrases were discounted as practice (20% of the session) and did not contribute to the reported entry or error rates. A visual chart showing the key combinations required for the letters was presented for the two initial practice phrases before the eight test phrases.

In summary, keypad-based methods are relatively slow. Regardless of technique, novice users tend to reach an entry rate around 10 wpm. After many sessions of practice the average entry rate for the best keypad-techniques peaks at around 25 wpm. Another downside with keypad-based methods is that they are primarily designed for a physical keypad. On the other hand, almost all mobile phones already have a keypad, and users have become familiar with the commercially successful keypad-based methods such as multi-tap and T9.

### 6.2.2 Projection Keyboards

A projection keyboard is a keyboard where the keyboard surface is projected on any solid surface. The user types text by moving the finger over the intended keys in the projection. The user's intended key presses are detected using sensors. Here the term "projection keyboard" is broadened to also include keyboard mechanisms where users can type on any surface but do not necessarily utilize a keyboard projection, such as methods where the user wears special gloves or straps.

Assuming perfect sensing technology what would be users' entry rate performance? Goldstein, Book, Alsiö and Tessa [1999] found that participants equipped with gloves covered with electrodes that sensed the finger movements could keep up with a dictation rate at 45 wpm. The error rate was 12.3%. Participants wrote approximate 1,000 words. Unfortunately the entry and error rates for novice users are not available.

It is also unclear how much the dictation task affected entry rates positively at the cost of increased error rates.

One commercial implementation is Senseboard developed by Senseboard AB. To type text the user attaches electronic straps to the wrists. The straps contain sensors that detect finger and hand motion. Using the sensor data a software system infers the user's intended key presses. Senseboard can according to Senseboard AB be used on any flat surface. There is no also need for a keyboard projection if the user can touch-type. Senseboard is currently not for sale and there are no publicly available results from an empirical experiment.

A competing technology is the Canesta Virtual Keyboard. It projects a laser keyboard surface and uses a 3D camera coupled with a motion-based pattern recognition system to detect the user's intended key presses [Tomasi, Rafii and Torunoglu, 2003]. Roeber et al. [2003] shows that participants can reach an average entry rate of 46.6 wpm and an average error rate of 3.7% when repeatedly typing "The quick brown fox jumps over the lazy dog" for 2 minutes. It is unclear in Roeber et al. [2003] whether error rate refers to corrected errors, uncorrected errors, or both.

Several other virtual laser keyboards exist commercially, for example I-Tech, VKB and VKey. Typically virtual laser keyboards project a visible keyboard area on a flat surface using a red laser diode. An infrared plane is projected parallel and slightly above the visible keyboard area. The user's intended key presses are detected by an infrared light sensor when the user's fingers intersects the infrared plane [Carau, 2001].

In summary, conducted studies of fully-implemented projection keyboards are limited in scope. Assuming that high entry rates and low error rates are easily achievable, the downside with virtual keyboards is the preparation time required to setup the keyboard. Also, the user has to be sitting down and have a flat surface available for the technique to be practical.

### 6.2.3 Tilt-Based Text Entry

Sazawal, Want and Borriello [2002] describes an accelerometer-based text entry method where the user tilts the device in various directional sequences to produce letters. A preliminary user study indicates that users do manage to write simple phrases with the system. No further experimental results are reported [Sazawal et al., 2002]. A further development of the system presented in Sazawal et al. [2002] is TiltType [Partridge, Chatterjee, Sazawal, Borriello and Want, 2002] where in addition to tilting the user presses special buttons to help the system to disambiguate the intended letter. No experimental results are presented [Partridge et al., 2002].

Wigdor and Balakrishnan [2003] uses tilt information to disambiguate keypad presses. The user disambiguates among the intended letters of a key by tilting the

device in one of four directions, e.g. tilt left for the first letter (for example *a* on the 2ABC key), tilt forward for the second, and so on. After approximately 20 minutes participants reached an average entry rate of 7.42 wpm with the tilt method, in comparison to 7.53 wpm with multi-tap (the baseline). After approximately 250 minutes of practice participants reached 13.57 wpm with the tilt method in comparison to 11.04 wpm with multi-tap. Errors were not allowed, i.e. users were required to correct their errors to proceed.

#### 6.2.4 Software Keyboards

A software keyboard is here defined as a keyboard with two specific properties. First, the visual representation of the keyboard is presented on a touch-sensitive screen. Second, to press a key the user touches a graphical representation of the key on the screen.

In the literature software keyboards are sometimes also known as graphical keyboards, virtual keyboards, or stylus/finger keyboards depending on context.

Software keyboards are popular on devices that are operated primarily with a pen or finger. Because software keyboards are used on a flat touch-screen the user does not perceive the same rich tactical sensation when using them as when using a regular desktop keyboard. From users' point-of-view primarily three use-quality properties are lost.

First, the tactile sensation that confirms that the key is pushed down is lost. The keys of a regular desktop keyboard lower when pressed. In contrast, a software keyboard surface is static in the sense that it does not move when a key is "pushed down". As a result the tactile feedback to the user when a key is hit is much lower, especially if the fingers are used as pointing devices. Styli are probably better in this regard since the user's sensation of when the top of the stylus has reached the touch-screen is more abrupt and as a result most likely stronger than the user's sensation of when the finger has come to contact with the touch-screen.

Second, the certainty that the key press was registered is lost. This problem is more acute with finger-operated touch-screens. Touch-sensitive displays use sensors to detect when the user touches the surface. If the touch-detection system fails the key press is not registered by the software. Registration failure happens if the sensor data is of low quality or no sensor data is detected. Low quality sensor data can easily occur with pressure-sensitive touch-screens if the user applies too little pressure or moves the finger too quickly. With capacity-based sensors [Gungl, 1989] contact failure can be caused by insulation of the fingers. In contrast, the probability that a desktop keyboard fails to register a key press is miniscule.

Third, the sense of the boundaries of the keys is lost. With a standard desktop keyboard the user feels the boundaries of the nearby keys and the boundary of the intended key.

The standard keyboard layout for software keyboards is QWERTY. This is also the only keyboard layout for software keyboards that has been widely accepted in the market. Zhai, Sue and Accot [2002] estimates the average expert text entry rate for a QWERTY software keyboard to be 34.2 wpm.

MacKenzie and Zhang [1999] found that after a 20-minute transcription task on average participants typed 28 wpm and had an uncorrected error rate of 3.2% with QWERTY. After 20 20-minute sessions the average entry rate was 40 wpm and the uncorrected error rate 4.8%. Participant could not correct errors in the experiment.

#### 6.2.4.1 Optimized Software Keyboards

It has since long been observed [Getschow, Rosen, Goodenough-Trepagnier, 1986] that the QWERTY keyboard layout is suboptimal when the contact point is single (e.g. a single finger or a single pen). Originally the QWERTY keyboard layout was designed to minimize mechanical jamming of the keys in typewriters [Yamada, 1980]. Since jamming is caused by two or more keys' arms getting stuck together, the likelihood of jamming increases when two nearby keys are hit. As a result, frequent pairs of pressed keys were distributed on the opposite sides of the keyboard. When two or more contact points are used (e.g. all 10 fingers, or both thumbs) the QWERTY keyboard layout is not optimal but still good enough for practical use.

Lewis, Kennedy and LaLomia [1992] and Lewis, Potosnak and Magyar [1997] use a model based on Fitts' law [Fitts, 1954] and character-level bigram statistics to find more efficient keyboard layouts. MacKenzie and Zhang [1999] uses a similar model to find a key arrangement they call the OPTI keyboard layout (see Figure 6.3).

Q	F	U	M	C	K	Z
		O	T	H		
B	S	R	E	A	W	X
		I	N	D		
J	P	V	G	L	Y	F1

Figure 6.3. The OPTI keyboard layout [MacKenzie and Zhang, 1999].

In a study by MacKenzie and Zhang [1999] participants initially typed 17 wpm with OPTI and had an average uncorrected error rate of 2.1%. After 20 20-minute sessions



the average entry rate was 45 wpm with an uncorrected error rate of 4.2%. Participants could not correct errors. Zhai, Sue and Accot [2002] estimates the average expert entry rate performance of OPTI to be 42.8 wpm.

Instead of relying on heuristics and trial-and-error when developing a new software keyboard layout, Zhai, Hunter and Smith [2000] uses a metropolis random walk algorithm to find the METROPOLIS keyboard layout (the layout is shown in Figure 6.4). The METROPOLIS layout is estimated to enable an average expert text entry rate of 46.6 wpm [Zhai, Sue and Accot, 2002].

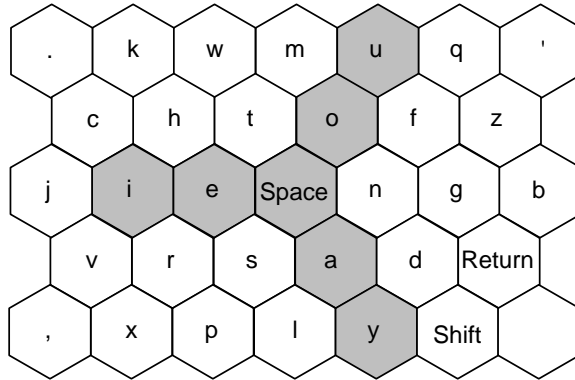


Figure 6.4. The METROPOLIS keyboard layout [Zhai, Hunter and Smith, 2000].

Another algorithmically derived keyboard layout is ATOMIK [Zhai, Smith and Hunter, 2002], see Figure 6.5. Using a movement efficiency model with Fitts' law, letter bigram frequency and alphabetical ordering tendency components they used simulated annealing to find a near-optimal software keyboard layout configuration within the model [Zhai, Smith and Hunter, 2002]. Its average expert entry rate is estimated to 45.3 wpm [Zhai, Sue and Accot, 2002]. The slight reduced average expert entry rate estimate is due to the alphabetic tendency of the letter key ordering. To introduce alphabetic tendency appears to be a reasonable tradeoff though given novice users find it easier to find the keys on a software keyboard with alphabetic order than without [Smith and Zhai, 2001].

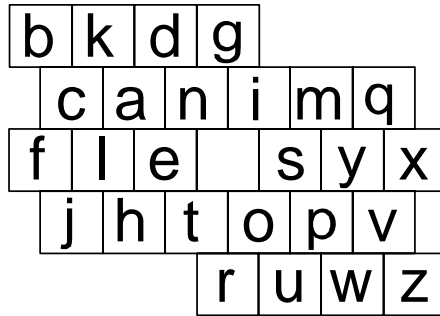


Figure 6.5. The ATOMIK keyboard layout [Zhai, Smith and Hunter, 2002].

A problem with optimized software keyboards is that users have to learn a completely new layout. Specialized practice software using expanding rehearsal interval [Landauer and Bjork, 1978] has been shown to improve learning rate [Zhai, Sue and Accot, 2002]. Another alternative is proposed by Magnien, Bouraoui and Vigouroux [2004] where the most likely keys that follow the users' previous text are pre-highlighted by the software keyboard. The hypothesis is that users' search time is reduced when practicing the new optimized layout. Magnien et al. [2004] show some evidence that input rates are higher for novice users when the most likely keys are highlighted. In their experimental design participants entered 50 words with or without prediction. To keep users in a novice state the keys were re-arranged before each word. Unfortunately details such as whether the conditions were balanced in the within-subject design and calculations of statistical significances are omitted in the paper. It is therefore hard to make any conclusive statements of the benefit of the technique. One interesting advantage with the proposed training technique is that the teaching aid can be enabled while simultaneously allowing the user to use the text entry method for daily work.

#### 6.2.4.2 Error-Correcting Software Keyboards

A problem with software keyboards is the lack of tactile feedback from the keys. Using a pen or finger it is easy to slightly miss keys, increasing error rate. Another problem is that imperfect touch-sensors result in some contact points being lost altogether.

Goodman, Venolia, Steury and Parker [2002] proposes a software keyboard that automatically corrects typing errors using a model that combines the probability of the pen's hit point location with a character-level language model. Goodman et al. [2002] found that after typing 1000 characters users had the same entry with or without correction (19.8 wpm vs. 20). The uncorrected error rate was reduced by a factor between 1.67 and 1.87 in comparison to non-corrected standard typing on a QWERTY software keyboard. Participants were not allowed to correct errors.

An alternative word-level solution is the discrete continuous shape writing method. The reader is referred to Chapter 2 in this dissertation for in-depth information on this method.

#### 6.2.4.3 *Double-Tapping Software Keyboards*

Nesbat [2003] presents a software keyboard based on the 12-key telephone keypad. Letters are assigned to keys based on their frequency distribution in the language. Frequent letters are accessed by double-tapping the pen while others are accessed by double-tapping two different keys. The assignment of letters to keys also depend on the probability that one key will be followed by another. This is to maximize movement efficiency in the spirit of Getschow, Rosen and Goodenough-Trepagnier [1986]. To my knowledge no formal evaluation of the text entry method has been conducted. Nesbat [2003] presents a theoretical model that predicts an average text entry rate of 50.1 wpm for expert users. As a reference the METROPOLIS software keyboard [Zhai, Hunter, Smith, 2000] layout has an estimated average text entry rate of 46.6 wpm, but using METROPOLIS keys are only tapped once.

#### 6.2.4.4 *Menu-Augmented Software Keyboards*

Isokoski [2004b] presents a software keyboard that is augmented with a marking menu [Kurtenbach, Sellen and Buxton, 1993]. When the user presses the pen on a key the user can instead of simply lifting the pen, flick the pen in one of eight directions. Depending on the direction of the flick a second letter is emitted. To support novice users the system uses a marking menu with delayed pop-up. If the user lingers with the pen for a preset interval a pie menu pops up. Importantly, the letters for each flick direction are constant and do not change depending on which letter key on the software keyboard the user selected, which should enable faster learning.

Isokoski [2004b] finds that after a 20 minute session users have an entry rate of 10 wpm with menu, in comparison to 17 wpm without. After 20 20-minute sessions the performance of both methods are about the same, around 25 wpm. The grand mean error rate was 0.96% with menu and 0.59% without. Interestingly, entry rates reported in Isokoski [2004b] for regular QWERTY software keyboard typing (the baseline) is much lower than what is reported in MacKenzie and Zhang [1999], where users reached an entry rate of 40 wpm on a software keyboard after 20 20-minute sessions.

#### 6.2.4.5 *Concluding Remarks*

In summary, software keyboards can reach competitive entry rates. An advantage with the software keyboard is that users immediately know what to do. There is also a certain degree of skill transfer from using a desktop QWERTY keyboard to using a QWERTY software keyboard even though from a motor control point-of-view the tasks are different.

A downside with software keyboards is the need for the system to include a touch-screen. If the software keyboard is driven by a pen, the pen needs to be picked up by

the user which therefore demands preparation time from the user. Another downside is that users tapping words repeatedly with a pen inflict forces concentrated to small areas of the touch-screen, which in the long run lead to screen damage and increased support cost for the manufacturer. In a study by Zhai, Sue and Accot [2002] it was also discovered that participants thought it was tedious and boring to type on a software keyboard.

### 6.2.5 Handwriting Recognition

Handwriting recognition has evolved considerably during the late 20<sup>th</sup> century [Tappert, Suen, Wakahara, 1990; Plamondon and Srihari, 2000]. Handwriting recognition can roughly be characterized into being either on-line [Tappert et al., 1990; Plamondon and Srihari, 2000] or off-line [Nafiz and Yarman-Vural, 2001; Koerich, Sabourin and Suen, 2003]. The differentiating factor between the two is the information available to the recognizer. An on-line recognizer receives users' handwriting as at least an ordered sequence of spatial sample points. An off-line recognizer receives users' handwriting trace indirectly embedded in a bitmap image.

Common approaches to on-line handwriting recognition are primarily data-driven statistical methods that rely on collecting and classifying vast amounts of users' handwriting. Efficient features and classification methods are still subject to considerable research [Plamondon and Srihari, 2000]. Another, less popular approach, relies on template-matching, for instance energy-deformation or elastic matching methods [Tappert, Suen and Wakahara, 1990].

A problem with handwriting recognition is recognition errors. A wizard-of-oz study suggests that an error rate of 3% is acceptable to users in most cases [LaLomia, 1994]. However, when writing "business critical" information, users demand an error rate <1% [LaLomia, 1995]. Frankish, Hull and Morgan [1995] studied the effect of recognition errors on users' perception of the text entry method. They found that in general handwriting recognition was not good enough for practical use. In addition, they found that users' perception and acceptance of handwriting recognition depended on the task. When accurate input did matter, e.g. filling in the forms of a fax request, users were frustrated by recognition errors. In another task that involved free-text writing in a diary, users were not as critical to recognition errors [Frankish et al., 1995]. Note that Frankish et al. [1995] tested isolated character recognition in commercially available handwriting recognizers at the time. Handwriting recognition methods have undergone tremendous development since the year 1995 (e.g. [Plamondon and Srihari, 2000]).

As a mobile text entry method, handwriting recognition has the advantage that users already know how to write. The learning curve should be minimal. The disadvantage with handwriting recognition in comparison to handwriting on pen-and-paper is that it is recognition-based, which always implies several limitations: recognition errors and

a limited system vocabulary risk impeding users' efficacy. Another disadvantage is the relative low speed that is achievable with natural handwriting. Hand printed character writing has an entry rate limited to around 15 wpm [Card, Moran and Newell, 1983]. Surprisingly, it appears that no comprehensive evaluation of cursive script handwriting recognition entry and error rate performance is available in the literature.

### 6.2.5.1 Recognition of Pitman Script

An alternative to handwriting recognition is automatic online transcription of stenography. After all, stenography systems were invented to overcome the limitations of long hand handwriting. Melin [1927; 1929] presents a comprehensive compilation of stenography systems in the Western world throughout history. The most common systems in the world were Pitman shorthand in United Kingdom and the colonies, and Gregg shorthand in the United States of America [Melin, 1927]. Overall, little research effort has been concentrated on automatically transcribing online stenography. The reason is probably because it is a very challenging task. Stenography is generally sloppy and incomplete, relying deeply on the writers' knowledge of the text written to save writing time. In most instances, transcription of shorthand to longhand text had to be completed immediately because the writer tended to forget the context of the text, making later transcription hard, or even impossible [Melin 1927].

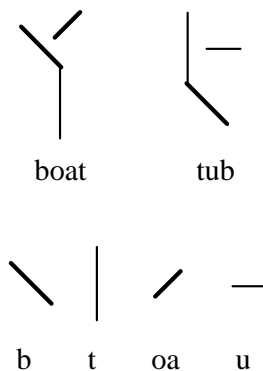


Figure 6.6. Examples of Pitman shorthand script. The thick strokes are written with more intense pressure with a pen, or at a different angle with a quill.

To my knowledge the first attempt to automatically recognizing shorthand is described in Brooks and Newell [1984]. At the time the hardware was insufficient. Hwte, Higgins, Leedham and Yang [2005] presents a recent attempt of a recognition architecture for recognizing online Pitman script (see Figure 6.6 for an example of Pitman script). No controlled text entry experiment is conducted. Hwte et al. [2005]

reports a recognition error rate of 7.14%. However, the circumstances around data collection are unclear. For example it is not mentioned how fast participants were writing Pitman script, and if the participants were experienced Pitman writers or not.

Assuming low error rates online Pitman script has the capacity to perhaps enable entry rates as high as 100 wpm [Hwte et al., 2005]. The downside is that users need to learn highly specialized form of stenography, which may take weeks or months of dedicated training [Melin, 1927].

### 6.2.6 Speech Recognition

A possible portable text entry solution could be speech recognition [Rabiner and Juang, 1993; Rudnicky, Hauptmann and Lee, 1994; Padmanabhan and Picheny, 2004]. An unavoidable side-effect of speech recognition is the lack of privacy and interruption to others when used in a shared environment.

Users can speak very fast, up to 200 wpm [Rosenbaum, 1991]. However just because a user can speak and be understood by other people, it does not necessary imply a computer system can accurately decode a user's utterances into text [Moore, 2004]. Karat, Halverson, Horn and Karat [1999] found that participants could only effectively transcribe text at speed of 13.6 wpm. In Karat et al. [1999] participants were instructed to correct errors to make sure the final text was correct.

In practice, the recognition process of converting the user's acoustic signal to text has undergone much research (see [Rabiner and Juang, 1993; Padmanabhan and Picheny, 2004] for overviews) but accuracy is still far from perfect [Karat et al., 1999]. Error correction in speech interfaces is also cumbersome and users tend to create cascading errors as a result of error correction difficulties [Karat et al., 1999]. For speech recognition on a mobile device, a multimodal correction interface via a software keyboard can to some extent alleviate this problem [Price and Sears, 2005].

When dictating it is hypothesized that speech drains cognitive resources and speech may be more suitable as a command input language than as a transcription interface [Shneiderman, 2000]. In contrast to writing, natural speech tends to be informal and conversational. As Shneiderman [2000] notes, an effective human-human interface does not necessarily imply an effective human-computer interface.

### 6.2.7 Prediction

To reduce keystrokes various forms of prediction can be used. For text writing in English, prediction-based methods are most commonly used for keypad-based methods, e.g. the dictionary-based methods and LetterWise that are discussed in the Physical Keyboards subsection. Several predictive models have also been proposed for use in text entry devices where the number of keys is significantly reduced to four keys or even less (e.g. [Tanaka-Ishii, Inutsuka and Takeichi, 2002]). Adaptive prediction models that attempts to continuously model the user's text writing have

also been investigated (see Tanaka-Ishii [2006] for a recent review). It is also important that the corpora used for language modeling accurately reflect users' text and editing operations [Soukoreff and MacKenzie, 2003a].

#### 6.2.7.1 *Software Keyboard Prediction*

Masui [1998] presents a system that combines a software keyboard with a pop-up prediction list. As the user types on the software keyboard a pop-up list shows the most likely words that follow from the user's input. A closely related system is the reactive keyboard [Darragh, Witten and James, 1990] that provides similar predictions for desktop keyboarding. Masui [1998] reports that a selected group of volunteers reached an average text entry rate of 8 wpm when writing a Japanese text consisting of 53 Kanji or Kana characters. No error rates are reported. A variant of menu-based prediction is commonly implemented in off-the-shelf handheld computers.

#### 6.2.7.2 *Unistroke Prediction*

MacKenzie, Chen and Oniszczak [2006] presents a similar system to Masui [1998]. The difference is that the user inputs Unistroke [Goldberg and Richardson, 1993] pen-gestures instead of typing on the software keyboard. Novices reached an average entry rate of 10 wpm and an error rate of 1.8% after an hour of practice of writing with regular Unistrokes (these numbers are estimated from Figure 6a and 6b in MacKenzie et al. [2006]). After 1 hour additional practice with the prediction system users reached an average entry rate of 12.8 wpm and average error rate of 1% [MacKenzie et al., 2006]. To simulate expert performance 3 of the original 10 participants were selected to perform an additional session where only the pangram "The quick brown fox jumps over the lazy dog" was repeated throughout. This test is identical in execution and spirit to the expert speed estimate test introduced in Kristensson and Zhai [2004] and Kristensson and Zhai [2005]. MacKenzie et al. [2006] found that these expert estimates led to an average entry rate of 22.5 wpm. If one of the authors of the paper is included (i.e.  $n = 4$ ) the average entry rate rises to 25.6 wpm.

#### 6.2.7.3 *Touch-Wheel Prediction*

Proschowsky, Schultz and Jacobsen [2006] presents a predictive text entry method for touch-wheels, such as the touch-wheel in the Apple iPod portable music player. The letter keys are distributed in alphabetic order along a circle on the touch screen. To type a letter the user touches the touch-wheel in proximity of the intended letter. Because of the low accuracy of touch and the high-density of letters on the circle, the user's intended letter is easily misinterpreted by the system. Proschowsky et al. [2006] proposes the use of a prediction system where the user's intended letter key is inferred by a character-level language model. The language model's influence is dynamically weighted depending on how fast the user re-positioned the finger. If the prediction outcome is wrong, the user needs to slide the finger clockwise or counter-clockwise to

scroll through the letters to eventually highlight the intended letter. When the finger tip is released from the touch-wheel surface a letter is emitted by the system.

Proschowsky et al. [2006] evaluated the text entry method in relation to two other methods. The first was the same system as the one described above except prediction was turned off. The second was a baseline method where the user slides the finger clockwise or counter-clockwise to move a selection cursor among letters lined up in alphabetic order on graphical display. Proschowsky et al. [2006] evaluated the text entry methods in a study with three sessions. However because each session was relatively short and only involved typing five phrases, and sessions were only separated with 20 minutes breaks; I treat the three sessions as a single initial session. The prediction method was significantly faster with a grand average entry rate of 5.6 wpm. In comparison, the grand average entry rate was 5 wpm when prediction was turned off and 4.5 wpm for the baseline (these numbers are calculated from Table 1 in Proschowsky et al. [2006]). The error rate was 1.7% with prediction, 2.4% without prediction, and 2.6% for the baseline. Proschowsky et al. [2006] reports that only the entry rate differences were significant. It is unclear in Proschowsky et al. [2006] whether error rate refers to corrected errors, uncorrected errors, or both.

#### 6.2.7.4 *The Cost of Prediction*

There are reasons to believe word-prediction is rather slow. For prediction to work the user has to select the intended word from a graphical list. This action involves simultaneously visual, cognitive and motor components. If the user is experienced with the system and knows where to expect certain words to appear the user can use a fast linear strategy where the user starts looking at the top choice and moves linearly along the graphical menu. If the user is not experienced the user must first notice that a graphical menu with predictions pop-up. Then a search is initiated where the alternatives are examined by the user to determine if any alternative is the desired word. If the intended word is found, the user can then proceed to select it.

#### 6.2.8 **Abbreviation**

Instead of predicting the users' intention, the users themselves can minimize input redundancy by abbreviating or compressing their input according to predefined schemes. Abbreviation is an old idea. For example, in monasteries in medieval Europe it was common practice to abbreviate words to save page space when copying bibles. For example, the Latin word *uerbum* ("word") was often compressed into *ūbū* [Jansson, 2004]. Vanderheiden and Kelso [1987] surveys many different abbreviation schemes for modern text writing on a computer.

Shieber and Baker [2003] and Shieber and Nelken [2007] propose a specific abbreviation scheme for modern text writing on size-constrained devices. The idea is that the user writes compressed text that is automatically decompressed by the text entry system. Therefore less keystrokes are necessary to compose the same text mass.



In comparison to prediction abbreviation does not require cognitive visual attention from the user to process prediction [Shieber and Baker, 2003; Shieber and Nelken, 2007].

The abbreviation model proposed by Shieber and Nelken [2007] is based on a compression scheme where all vowels (y is treated as a consonant) are dropped except when a vowel initiates a word. For example the word *association* is abbreviated as *asctn* [Shieber and Nelken, 2007]. A controlled experiment showed that participants reduced their input speed with 1% when abbreviating their input if the time to correct abbreviation mistakes is taken into account [Shieber and Nelken, 2007]. Therefore it appears the abbreviation is not very effective in practice.

Abbreviation could be used in conjunction with character-level methods such as Unistrokes [Goldberg and Richardson, 1993], thumb keyboards, etc. A possible weakness to abbreviation is the difficulty in modeling typing mistakes. Because users' typing is expanded, a single typing error may result in the entire word being wrong. Empirical investigations of thumb keyboard typing have revealed that participants tend to produce many typing errors [Clarkson et al., 2005]. In Experiment 3.2 in in this dissertation it was discovered that almost every 4<sup>th</sup> word had one typing mistake. On the other hand, abbreviation could be used to reduce typing stress, and increase the comfort in using a thumb keyboard by reducing the amount of necessary key presses.

### 6.2.9 Artificial Alphabets

Goldberg and Richardson [1993] proposes the influential text entry system called Unistroke. Unistrokes is designed for pen-based mobile devices and can be said to be motivated by three primary design goals: 1) efficiency, 2) accuracy and 3) minimize visual attention.

To achieve efficiency the alphabet is designed with a minimal number of strokes to disambiguate the letters. Also, more common letters such as *a* or *e* have the shortest and most easily reproduced strokes, see Figure 6.7. Accuracy is attained by designing the alphabet so that individual pen-gestures are not easily confusable by the recognizer, even though Goldberg and Richardson [1993] notes that there were recognition accuracy problems with the letters *m* and *n*. Minimizing visual attention is possible because the pen-gestures are articulated in-place on top of each other and recognized invariant of small scale, translation and rotation perturbations.

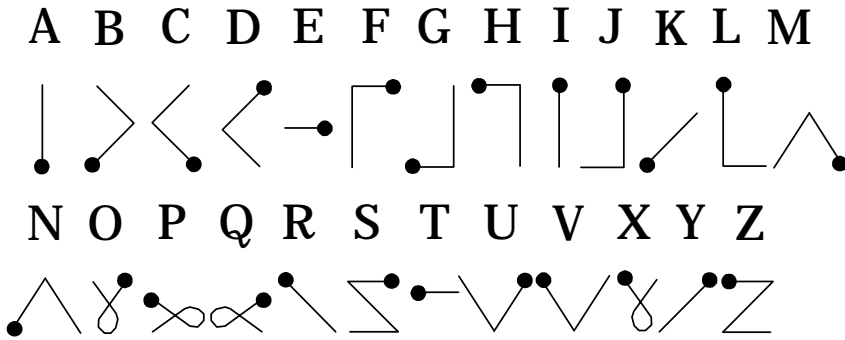


Figure 6.7. The Unistroke alphabet [Goldberg and Richardson, 1993].

Graffiti is a unistroke variant that was developed and marketed by Palm Inc. Graffiti is heavily inspired by Unistrokes but has an alphabet where the individual strokes resemble the corresponding letters more. Around 40% of the lower and 70% of the upper Graffiti letters are identical to the Latin alphabet [MacKenzie and Zhang, 1997]. MacKenzie and Zhang [1997] found that users can write Graffiti with an error rate around 3% after a few minutes of practice.

Jot is variation of Graffiti that is so close to the idealized Latin alphabet that it can essentially be regarded as an isolated character-recognition based text entry method. Some characters are multi-stroke in Jot, for instance the capital *T*. Sears and Arora [2002] compared novice users performance of Jot and Graffiti. They divided the experiment into six different tasks. However, only one task (Task 6) can be regarded as a real text entry task where participants entered a series of phrases. The other tasks were concerned with writing addresses and other forms of auxiliary text. Therefore only results from Task 6 in [Sears and Arora, 2006] are reported here. Using Jot, participants reached an average entry rate of 7.74 wpm and an error rate of 15.4%. The error rate may be 0.154% because the unit is not noted in Table 5 in [Sears and Arora, 2006]. However judging from the discussion part in the paper it appears more likely that the error rate is 15.4%. It is important to note that participants were exposed to 10 minute of practice before the test. Results obtained during practice are not reported. Unfortunately there appears to be an error in Table 3 that makes it impossible to know what the average entry rate for Graffiti was in the text entry task (Task 6, Table 3 in [Sears in Arora, 2003]). However, the authors imply average entry rates were similar or lower with Graffiti than average entry obtained with Jot. In a controlled experiment Költringer and Grechenig [2004] found that when novice users were told to write three phrases Jot was significantly slower than the software keyboard. Jot resulted in an average entry rate of 9.24 wpm and the software keyboard resulted in an average entry rate of 13.64 wpm. The average error was 19.35% for Jot and 4.11% for the software keyboard. Költringer and Grechenig report error rate as “total error rate” [Soukoreff and MacKenzie, 2003b] that lumps together uncorrected and corrected error rate.

EdgeWrite [Wobbrock, Myers and Kembel, 2003] is another alphabet variation. The distinct feature of EdgeWrite is that the system is designed for robustness. The EdgeWrite alphabet is meant to be used within a square area surrounded by edges. The text entry method is intended to lead to stability and minimize error. To that effect the alphabet is designed so that all letters are written by sliding the stylus next to the edges. After typing 12 sentences novice users reached an average entry rate of 6.6 wpm in comparison to Graffiti (the control condition) which reached 7.2 wpm. Error rates are not reported but are presumably low because the main motivation behind the paper is to introduce an alternative to Graffiti with lower error rates [Wobbrock, Myers and Kembel, 2003].

EdgeWrite can also be inputted with an isometric joystick in addition to a stylus [Wobbrock, Chau and Myers, 2007]. Wobbrock et al. [2007] shows that participants that used EdgeWrite via an isometric joystick reached 4.2 wpm after the first 20 minute session (this number is deduced from Figure 6a in [Wobbrock et al., 2007]). The error rate for individual sessions is not reported. Note that the procedure included a practice session before each testing session and the results from the practice sessions are not reported, which makes it impossible to understand the true initial effectiveness of the method. After 15 20-minute sessions the participants reached an average entry rate of around 10 wpm. The control condition was multi-tap which also reached around 10 wpm. EdgeWrite had a grand average error rate of 1.34% in comparison to 0.52% for multi-tap.

A further study of EdgeWrite involved combining EdgeWrite with completions. Essentially the alphabet is extended to include common words. For example the alphabetic notation for the word *the* is defined as a continuation of the EdgeWrite gesture for *t* [Wobbrock, Myers and Chau, 2006b]. Wobbrock et al. [2007] found that joystick-controlled EdgeWrite with completions reached 10 wpm after the first 20 minute session (this number is derived from Figure 6b in [Wobbrock et al., 2007]). The error rate for individual sessions is not reported. After 15 20-minute sessions participants reached an average entry rate of 15.15 wpm for EdgeWrite completions in comparison to 15.75 wpm for T9 (the control condition). The grand average error rate was 0.18% for the EdgeWrite with completions, and 0.26% for T9.

#### **6.2.10 Word-Level Single-Stroke Encoding**

Perlin [1998] proposes a pen-based text entry technique called Quikwriting where the central idea is that the user should not be required to lift up the pen between inputted letters until a word is completed. In this sense it is a word level single stroke method, although each letter in a word still requires two to more individual (but connected) movements. With Quikwriting letters, numbers and auxiliary keys are distributed along the edges of a square (Figure 6.8). The square input area is divided into zones that group the letters. The center area is called the resting zone. The input process for a single letter is a three-step process. First, the user draws a pen stroke from the

resting area into the zone that contains the desired letter. Second, the user moves the pen stroke into the zone that topologically maps to the intended letter's position in its zone. Third, the user moves the pen stroke back to the resting zone, from where the process can be repeated for the next letter. For an example, see Figure 6.8. Isokoski and Raisamo [2004] found that novice users reached an average entry rate of 5 wpm and an average error rate of 0.78% after entering text for 15 minutes (these numbers are derived from Figures 4 and 5 in Isokoski and Raisamo [2004]). After 20 15-minute sessions participants had an average entry rate of 15 wpm [Isokoski and Raisamo, 2004] and an average error rate of 0.32% (this number is derived from Figure 5 in Isokoski and Raisamo [2004]).

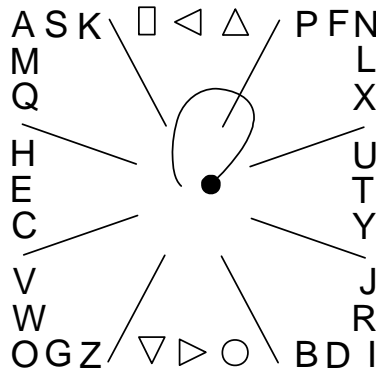


Figure 6.8. Writing the character “f” with the Quikwriting method [Perlin, 1998]. The dot marks the beginning of the pen-gesture.

Mankoff and Abowd [1998] proposes a pen-gesture text entry method called Cirrin (Circular input). To enter text with Cirrin the user moves the pen from key to key. The system simply records all keys that are visited and therefore the keys must be laid out in such a way that it is possible to reach the next key from a previous key without intersecting a third key. A solution is to lay out the keys in a circle, but other closed shapes such as a hexagon or rectangle could possibly be used. To my knowledge, Cirrin has never been properly evaluated in a controlled study. Since the keys are distributed along a circle the pen-gestures tend to be long despite the attempt in Mankoff and Abowd [1998] to align frequent letter-pairs. As a result, fast text entry rate cannot be expected.

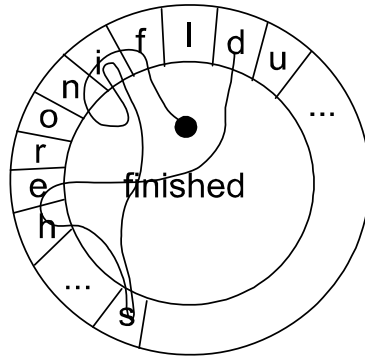


Figure 6.9. Writing the word “finished” with Cirrin [Mankoff and Abowd, 1998]. The dot marks the beginning of the pen-gesture. Not all keys in Cirrin are shown in the figure.

Finally, another word-level single-stroke encoding is continuous shape writing. For a description of continuous shape writing the reader is referred to Chapter 3.

### 6.2.11 Hierarchic Text Entry Methods

Venolia and Neiberg [1994] presents T-Cube. It is inspired by the work of Goldberg and Richardson [1993] on Unistrokes and the work of Kurtenbach, Sellen and Buxton [1993] on marking menus. T-Cube consists of a two-layered pie menu. The second tier pie menus group letters, numbers, punctuation symbols, etc. The single first tier pie menu trigger individual second tier pie menus. The initial pie menu is always displayed to the user (see Figure 6.10, right). To enter a character the user presses the pen down on a slice in the initial pie menu. This action triggers a second pie menu to pop up with its geometric centroid at the user’s pen position (Figure 6.10, left). The user selects a character in the second pie menu by flicking the pen in the in the direction of the intended character. To help novice users T-Cube can be set to a training mode where the secondary pie menu is revealed whenever the pen is hovered over a slice.

The display of the second pie menu is only a visual guide to the user and not necessary for the system to work. Similar to marking menus [Kurtenbach, Sellen and Buxton, 1993] the second pie menu is only displayed after an initial delay to encourage users to quickly flick in the direction of the intended letter without visual guidance.

In a learning curve study with T-Cube novice users reached an average entry rate of 3.2 wpm after 20-30 minutes (this number if calculated from Figure 7 in Venolia and Neiberg [1994]). After nine 20-30 minute long sessions the average entry rate was 21.6 wpm (from Figure 7 in Venolia and Neiberg [1994]). The error rate is not reported.

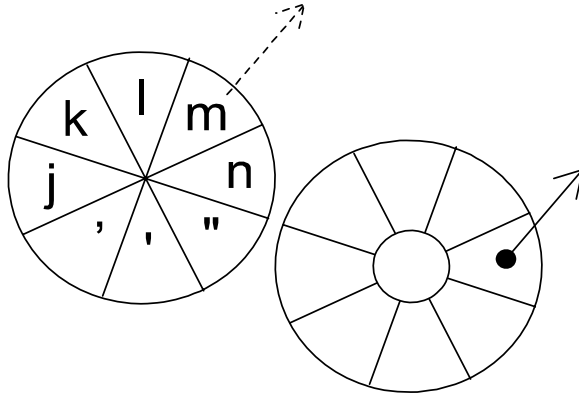


Figure 6.10. Using the T-Cube to enter the character *m*. First, the user selects a slice in the leftmost pie menu, thus triggering the popup of the rightmost pie menu. Second, the user flicks in the direction of the intended character. The dot marks the beginning of the pen-flick.

Martin [2006] presents VirHKey (Virtual Hyperbolic Keyboard) which is a character-level text entry method. The user specifies individual characters by navigating a hyperbolic projection. Similar to T-Cube the VirHKey alphabet is self-disclosing and the pen-gestures fixed. Therefore, after practice users learn the pen-gestures. After an initial 20-minute session average entry rate was 6 wpm (this number is estimated from Figure 15 in [Martin, 2006]). The error rate was 2.8% (participants were not allowed to correct errors). Average entry rate was 23 wpm after 20 20-minute sessions of practice and the error rate had increased to 5.9% [Martin, 2006].

Another hierarchic text entry system is Dasher [Ward, Blackwell and MacKay, 2002]. In Dasher the user navigates a cursor through hierarchical graphical boxes that contains the desired characters (Figure 6.11). The boxes' sizes and hierarchic structures are dependent on a probabilistic language model [Ward et al., 2002]. In a sense, with Dasher the user “drives” the cursor to the intended destination – the desired sequence of words. For common words and sequences of words, the driving will be efficient because the probabilistic language model attempts to align the most likely character and word sequences along a straight path for the user. For less common words, the driving will be more involved and the user has to “turn” the cursor towards small areas.

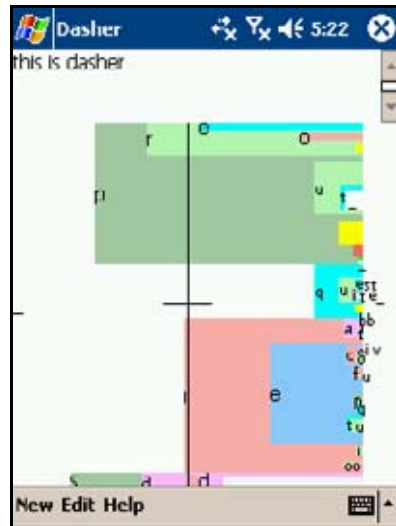


Figure 6.11. Dasher [Ward et al., 2002].

Ward et al. [2002] performed an experiment investigating users' performance in a dictation task. Note that the experiment was carried out on a regular, not a handheld, computer, but the results should apply for a handheld computer as well. Novice users reached a text entry rate ranging from 6 wpm to 14 wpm after five minutes of dictation. The word-level error rate ranged from 0.5% to 1.8%. After total one hour of dictation the text entry rate was 12-26 wpm. The word-level error rate ranged from 1.6% to 3.8% (these numbers are estimated from Figures 14 and 15 in [Ward et al, 2002]). Note that Ward et al. [2002] reports entry rate as actual words entered.

In the experiment [Ward et al., 2002] Dasher used a language model trained on the novel *Emma*. Excerpts from the book not used when training the model were used as stimuli in the experiment (e.g. test data). It remains an open research question how well Dasher would perform on open text outside the language model's training data domain.

Ward et al. [2002] also investigated how screen size in pixels affected entry rate. A single experienced participant reached an average entry rate of around 30 wpm (Figure 19 in Ward et al. [2002]) when the screen size was set to  $200 \times 200$  pixels ( $6.4 \times 6.4$  cm on the device tested). A larger screen size did not improve performance. A lower screens size of  $100 \times 100$  ( $3.2 \times 3.2$  cm of the device tested) pixels reduced performance (average entry rate around 17 wpm, Figure 19 in Ward et al. [2002]). It appears Dasher can be a competitive mobile text entry method for some users if the handheld device can dedicate more than  $6.4 \times 6.4$  cm of the display area to the text entry method.

## 6.3 Design Dimensions

---

This section identifies and analyzes the design dimensions of mobile text entry methods. Note that these design dimensions reflect different aspects of text entry methods from the text entry method designers' point-of-view. The design dimensions are not meant to be directly comparable against each other. As such, some design dimensions are dependent variables, some independent variables, and other are neither. In fact, even ranking of these design dimensions is problematic since the relative importance of these dimensions is highly subjective and dependent on the ultimate goal a text entry method is trying to achieve. Again, the design dimensions are intended to visualize the design space in text entry and should not be interpreted as a conclusive systematic taxonomy or rulebook on how to design a new text entry method.

### 6.3.1 Entry Rate

Entry rate is one of the most important design dimensions of any text entry method. Clearly a text entry method must be effective enough to allow users to perform their tasks with minimal disruption. A too slow method frustrating to use and disrupts the work-process flow for the user writing text. The gold standard is the regular desktop keyboard which is generally regarded fast enough for all-purpose text writing. Ideally, average text entry rate performance is at the level of a regular desktop keyboard.

Entry rate is often used as the most important, or sole, metric in text entry system design and research. This could be due to a lack of a more comprehensive understanding of the total design space. It could also be based on an implicit and hypothetical assumption that entry rate is an end result of most of the other dimensions to be discussed later.

However there are many aspects to entry rate. For most methods, users tend to become increasingly proficient with the text entry method they use. Thus the average entry rate is not meaningful when taken as the average of a large time span. In the literature researchers frequently recruit a sample subset from the population and conduct controlled experiments with a set of participants. In this situation average entry rate tends to refer to the average of the entire sample for a session (typically the last). This is usually the number cited by other researchers when comparing text entry methods.

A problem with this habitual comparison of average entry rate is that the average entry rate is not comparable between methods outside the controlled experiment where the original numbers were derived. First, different experiments have different sets of participants, and text entry is shown in several experiments to be highly dependent on individual skill (e.g. [Matias, E. and Buxton, W, 1996; Lyons, K, Starner, T. and Gane, B., 2006]). There also exist factors such as participants' age that are known to affect typing entry rate [Rosenbaum, 1991] and whether or not



participants are inputting text in their native language [Isokoski and Linden, 2004]. Second, there is no standard on how experimental procedures for evaluating text entry methods should be designed.

The term “average entry rate” washes out the fact there exists both a minimal and maximal entry rate for every technique. Here two complementing terms are introduced: the floor entry rate and the ceiling entry rate.

#### 6.3.1.1 *Floor Entry Rate*

The floor entry rate is defined as the lowest entry rate anticipated for a method. It is the initial performance expected from participants writing open text for some period of time, e.g. 10-60 minutes. Often the floor entry rate can be measured immediately and trivially by letting a complete novice user type a few test sentences with the text entry method under investigation. In fact, many text entry researchers report results where only a few sentences are written by participants. Such entry rates are considered floor entry rates. The measured average floor entry rate is important since it is a component of immediate efficacy (to be discussed later) of the text entry method. A high floor entry rate means users can benefit from the text entry method from the start.

#### 6.3.1.2 *Ceiling Entry Rate*

The ceiling entry rate is defined as the highest entry rate physically achievable by users writing a portion of realistic text with a text entry method. Unlike expert entry rate, ceiling entry rate is the rate where it is physically almost impossible to surpass.

For ceiling entry rate to be meaningful it needs to be empirically validated through an experiment with a random sample from the user population. How to define a standard experimental procedure to derive ceiling entry rate is an interesting methodological question concerning a number of components that must be isolated and controlled.

First, skill transfer varies between text entry methods. For example a thumb keyboard benefits from skill transfer from desktop keyboard skills, keypad-based methods benefit from skill transfer from multi-tap / T9 keypad skills. Other methods do not benefit from skill transfer, such as Unistrokes [Goldberg and Richardson, 1993], where the user needs to learn a new alphabet.

Second, character-level methods can be faster for participants to completely memorize in muscle memory than word-level methods. This is because character-level methods involves only 26 or so unique input patterns and therefore require less learning effort. Therefore, to measure the ceiling entry rate of word-level methods users may require more training. It may also be necessary to create a specialized experimental setup, such as letting users repeatedly write a single phrase over and over to saturate learning (e.g. Experiment 3.3).

Third, participants may need explicit direction to realize how to excel in entry rate with a text entry method. It may be that it does not suffice to use the phrase “proceed quickly and accurately” when measuring ceiling entry rate. For novel text entry methods, participants may never realize how quickly they can write unless one explicitly shows them. As an example, with continuous shape writing it is possible to become extremely fast when writing phrases whose motor patterns are completely memorized in muscle memory (over 98 wpm, see Experiment 3.3 for details). However, to achieve these entry rates the participant needs to realize that it is not necessary to look at all at the keyboard and rather merely move the hand from memory. A hypothesis is that for innovative text entry methods some participants require explicit demonstration to reach expert performance. This is not surprising if one consider the history of typewriting. Experts in typewriting touch type text without looking at the keyboard. However, touch typing was not discovered until decades after the typewriter was invented [Yamada, 1980]. Apparently prolonged usage is not always enough for users to self-discover the most effective way to use a text entry method. In continuous shape writing, when a user truly “shape writes” text the user articulates pen-gestures from muscle memory rather than attends visual-guided tracing of the letter keys. Future research is required to properly define an experimental procedure for finding ceiling entry rates for text entry methods.

Finally, ceiling entry rate should not be confused with expert entry rate and average expert entry rate. Ceiling entry rate is maximum performance measured as an average within a sample of participants. In the process of finding the ceiling entry rate participants will make many mistakes where the input has to be discarded. In comparison, expert entry rate is the entry rate obtained when participants highly skilled and proficient with the text entry method are writing real text. Expert entry rate is only meaningful together with an error rate measure since there is a speed-accuracy tradeoff in text entry tasks in general. Average expert entry rate is the average of several participants’ entry rates. Typically expert entry rates are obtained through a longitudinal learning curve experiment, or by measuring text entry rates of highly proficient users of the method (e.g. the researchers who has worked with the text entry method for a long time).

### 6.3.2 Error

Entry rates are meaningless when not accompanied with some measurement, control and reporting of error rates. Human performance inherently involves speed-accuracy tradeoffs. For the same task, a sloppy and erroneous performance tends to be faster than a careful and precise performance.

It is important for a text entry method to attempt to restrict errors. A method that results in too many errors can be perceived as tedious and untrustworthy by users. On the other hand text entry is a skill-based activity where errors will inevitably occur.

Handling these errors is an integral and inseparable part of text composition in practice.

The literature is sparse on studies of users' acceptance of errors. LaLomia [1994] found that in the context of handwriting recognition writers generally only accept a 1% error rate on the word-level in business communication. For more general text an error rate of 3% was acceptable. However it is unclear in that study whether the errors were errors that writers discovered and corrected, or undiscovered errors in the composed text. It is plausible that typing errors are generally more accepted in informal email and instant messaging conversations.

Errors can be counted on both the character and the word level. Thumb keyboards, for example, causes errors on the character-level. Cursive handwriting and continuous shape writing causes errors on the word-level. An advantage of character-level errors is that they most likely do not form valid words. Therefore, they can be captured by spell checking algorithms in the text composition software. Word-level errors cause an entire unintended word to be wrong, which may be harder for spell checking software to detect (unless new spell checking software is developed at a sentence or phrase level). On the other hand, word-level errors are more easily detectable for the user because they break the logical flow in the text.

### **6.3.3 Learning Curve**

Mobile text entry methods are skill-based. Generally text entry skill is a function of practice and as a central tendency text entry skill learning tends to follow a power-law curve [Crossman, 1959]. This tendency has also been observed in many of the studies that have been discussed earlier.

The learning curve of a mobile text entry method can be an important dimension to consider if high text entry rates require extended amounts of practice to be reachable. If a text entry method is fast from the beginning, e.g. 20-25 wpm, the rate of increase in learning is less critical. Frequently, text entry methods are rather slow in the beginning with average entry rates < 10 wpm (e.g. the twiddler that has an average entry rate of around 6 wpm and increases over time to over 30 wpm [Lyons et al., 2006]). In such case the rate of increase along the initial stage in the learning curve is more important because the text entry method must enable users to quickly reach an acceptable text entry rate.

### **6.3.4 Immediate Efficacy**

Ideally a novice user is able to use a mobile text entry method effectively immediately without any training. Effective use involves a high enough text entry rate and low enough error rate that the method is not perceived as too tedious, frustrating, or cumbersome by users.

To my knowledge there are no studies on immediate efficacy and what entry and error rate levels users accept. There may not be a simple standard either since rationally the amount initial learning effort justified depends on the eventual payoff. A common example is the comparison between tricycles and bicycles. While the latter does take serious learning to get started, its eventual performance payoff attracts most people to learn it. In the context of mobile text entry, most likely users understand the fact that mobile devices are limited in form factor and cannot provide a satisfactory desktop-level experience. The Graffiti text entry method is an example of the immediate efficacy effect. Prior to Palm Computing's Graffiti text entry method traditional hand printed character recognition was often used. The Apple Newton provided both hand printed and cursive handwriting recognition. However, the accuracy of handwriting recognition was too low in the early days, which is believed to have contributed to the Newton's failure [MacNeill, 1998]. Palm Computing's handheld computers were on the other hand a tremendous success. One reason that was often referred to by trade press journalists and analysts was the low-error rate Graffiti text entry method [Butter and Pogue, 2002]. An empirical investigation by MacKenzie and Zhang [1997] revealed that users could learn the Graffiti alphabet very quickly – within five minutes of use. Therefore, while not having stunning entry rates, Graffiti due to its low error rate proved to be a practical method with high immediate efficacy.

An easy method to judge immediate efficacy is to examine entry and error rates that users obtain when typing open text within the first ten minutes of use. If either entry or error rates are unacceptable, the method lacks immediate efficacy.

### 6.3.5 Form Factor

For almost all text entry methods the form factor of the mobile device plays an important role. The limited physical size of a device may prohibit a thumb keyboard that requires more space than the standard telephone keypad. If the device consists of only a large touch-screen the text entry method must rely on finger, pen or speech input – ruling out many physical key-based methods.

Some mobile text entry methods are more adaptive than others. For instance EdgeWrite [Wobbrock et al. 2003] has been adapted to both pen and joystick [Wobbrock et al., 2007]. The joystick version can theoretically be used on a mobile device with a very small size [Wobbrock et al., 2007].

### 6.3.6 Preparation Time

Preparation time is here defined as the time interval from the user's intention to start writing text when the text entry method is not set up, to the point in time when the user is able to start writing. Methods that are integrated into the system such as permanently attached thumb keyboards and keypads have a startup cost of zero. Systems such as projection and fold-up keyboards have in comparison a high long startup cost because the user is required to put down the primary device (for example

a smart phone or handheld computer) and pick up and install the text entry system. Pen-based systems have a much lower preparation time but may still require the user to fetch the pen for the device. In that sense, the preparation time is not zero for pen-based devices unless the user is already actively using the pen with the system.

Since mobile text entry is supposed to be always available the impact of a high preparation time should not be neglected. Such methods need to be complemented with a secondary mode of input. If users do not appreciate the preparation time there is a risk that the secondary text entry mode becomes the user's primary method in the end. At worst, a mobile text entry method with high preparation time may result in the user not bothering writing text on the device.

### **6.3.7 Localization**

Not all methods are equally suitable to the vast amount of languages in the world. Almost all text entry methods in the literature have focused on (U.S.) English. Some methods, such as the keyboard, can easily be extended to other Western languages such as German and French by introducing the additional accented letters as additional keys. In Chinese, the keyboard is not enough in itself and complemented with menu-based systems where the Latin characters entered via the keyboard (pinyin) are disambiguated into Chinese characters via a graphical menu system [Wang, Zhai and Su, 2001].

Systems that depend on language modeling need to adapt language resources such as corpora and dictionaries. In some systems, such as Dasher [Ward et al., 2002] and continuous shape writing additional inventive changes are needed for the system to support languages with dramatically different structure. The reader is referred to the Localization section in Chapter 3 for additional information on how continuous shape writing was adapted to support German and Korean.

### **6.3.8 Comfort**

The comfort of a device is also important. There is distinction between mental and physical comfort. For example, physically typing on a thumb keyboard and keypads can become uncomfortable after prolonged usage because the thumbs become numb. Typing on software keyboards is not as physically demanding as typing on thumb keyboards. On the other hand users perceive typing on software keyboards as "boring" [Zhai, Sue and Accot, 2002].

The general standard design of the mobile phone has also been reconsidered in light of the recent increase in text entry on these devices [Hirotaka, 2003].

It appears from a literature review that few text entry researchers considered comfort. The focus is generally on entry rate exclusively. This is also probably because comfort is highly individual and hard to measure. For example, it is plausible thumb

and nail sizes affect thumb keyboard comfort, especially for particularly small thumb keyboards.

Another aspect of comfort is the loose term “fluidity”. Fluidity is generally used as a term in sketching interfaces to describe user interface actions that are perceived by users as easy, straight-forward and quick, yet with enough expressiveness for the actions to perform something useful. For example, with marking menus [Kurtenbach et al., 1993] users can select deeply-nested menu choices with single fluid pen-gestures. Another example is continuous shape writing where users write entire words by single fluid pen-gestures.

### 6.3.9 User Engagement

A problem with mobile text entry methods is the novelty of the methods for users. Text entry is a highly skill-based task that demands users’ active participation. This is problematic since users want to write text, not learn a new text entry method. A text entry method that is perceived as plain, tedious or simply “boring” is not likely to be actively used by users unless it is forced upon the users as the only text entry method on the device.

A related task is the challenge in creating an instructive tutorial or training application that is engaging enough to keep users interested long enough to reach efficacy with the text entry method. There is a risk that users who have the least time and just want to get work done are the ones that simultaneously need training to be effective and do not have time or patience to start engaging in a training program. Ideally the training part is integrated with the general text entry method and lets users be both productive and able to go through the training. For example, a training program that can teach users the text entry method and let them work on open text the users intend to write (e.g. email) is more valuable. An example of such a system is the software keyboard highlighting method presented by Magnien et al. [2004], discussed in the Software Keyboards subsection above.

### 6.3.10 Visual Attention

A good text entry method should minimize visual attention required. Minimizing visual attention allow users to focus on the text composition task, for instance formulating an email to a colleague. Many researchers have previously emphasized the importance of “eyes-free” text entry (e.g. [Goldberg and Richardson, 1993]). Pavlovych and Stuerzlinger [2003] explicitly investigated how fast users could type with and without visual reference to the device. They compared two methods of multi-tap, where one variant used the traditional letter assignments of the keys, and the other had letters on each key re-arranged if it reduced average number of required key presses for words [Pavlovych and Stuerzlinger, 2003]. In the final session participants were asked not to look at the device or the button assignment cheat sheet. Overall average entry rate dropped slightly but error rates increased dramatically (see

Figures 6 and 7 in Pavlovych and Stuerzlinger [2003] for details). With visual reference error rates were less than 1%. Without visual reference, error rates increased to over 6% for multi-tap with standard letter arrangement, and 4% when the letters on each key were arranged.

Wobbrock et al. [2007] investigated how fast participants could write text using joystick-EdgeWrite while holding the input device occluded under table. Wobbrock et al. [2007] found that users had an average text entry rate of 3.09 wpm for multi-tap and 8.09 wpm with isometric joystick-EdgeWrite. The average error rates were 1.58% for multi-tap and 2.96% for joystick-EdgeWrite. As a reference point, with visual reference users reached around 10 wpm for either condition. With visual reference the grand average error rate for EdgeWrite was 1.34% and 0.52% for multi-tap.

In summary, experimental results reveal that even methods that are a highly plausible to be used eyes-free suffer decreased entry and error rate performance when participants are explicitly denied visual reference. It appears users prefer looking at least partially at the input device when entering text, even when using so-called eyes-free text input methods.

It is important to note that an eyes-free method does not necessarily translate into a fast text entry method and vice versa. For example Unistrokes [Goldberg and Richardson, 1993] are eyes-free but rather slow. In comparison, traditional software keyboards and Dasher [Ward, Blackwell and MacKay, 2002] are relative fast even though the visual attention demand is essentially maximized.

### **6.3.11 Cognitive Resources**

Another cost almost any text method uses significant cognitive resources. Shneiderman [2000] writes in an influence article about how increased cognitive demands confound the text writing process for speech recognition interfaces. A method such as Dasher [Ward et al. 2000] most likely require intense cognitive processing because the user must scan a dynamically changing scene and plan ahead where to steer the cursor to enter the desired text. Most text entry methods are in a continuum ranging from little to massive cognitive overhead depending on the skill of the user and the current word inputted. For example continuous shape writing demands plenty of cognitive resources from users writing unfamiliar words: the user must plan ahead at least part of the trajectory, internalize how to spell the word, etc. On the other hand, if the user completely learns the shape writing gesture for the word, articulation can be executed directly from muscle memory which has a low cognitive overhead. The same argument follows for artificial alphabets that require a conscious initial learning effort from users.

### 6.3.12 Privacy

Some situations, such as public transit or conference meetings require the ability to enter text privately. Speech recognition does not fulfill this criterion. Keyboard-based methods can also be disturbing, for instance if used by the audience at presentations.

Most text entry methods have a reasonable level of privacy in this regard. The notable exception is again speech recognition.

### 6.3.13 Single vs. Multi-Character Entry

Some text entry methods work on a character-by-character level. In contrast, other text entry methods operate on the word level, for example handwriting recognition, speech recognition and continuous shape writing. A method such as Dasher [Ward et al., 2000] where the user navigates through a structure to enter words is in this regard operating on the single-character level because each character in the process can be outputted as soon as it is specified.

There are reasons to believe single-character entry is better. One reason is easier integration with operating systems and applications that almost all are built to accept single-character input from a keyboard. Further, the results from single-character input can be corrected immediately by the user. With a single-character input method an error results in a single character that is wrong. Many such errors can be trapped with a spell checker. With a word-level entry method an error results in another legible word that might be harder to spot. The risk for misconceptions by a reader when reading entire words that are wrong is also greater.

On the other hand, word-level entry methods can result in fewer errors because information about an entire word is taken into account by the text entry method. For example, continuous shape writing uses information about the entire geometrical trace to match a likely word. This allows a certain degree of tolerance even if the user's input is severely deformed. A similar case is the discrete shape writing method which can correctly recognize the intended word even when all keys are missed in the most extreme case, provided no other candidate words are closer to input pattern (see Chapter 2 and Chapter 3 for in-depth information on discrete and continuous shape writing). A further hypothetical advantage of word-level text entry methods is that the input process is chunked into larger semantic entities (cf. chunking in gestural interfaces in [Buxton, 1986]). It is plausible that users think of words rather than characters when writing. In fact even with character level input methods such as type writing, skilled typists tend to memorize larger chunks of key sequences that are frequently used [Yamada, 1980]. People's typing performance tends to be much lower when typing a random sequence of letters [Fendrick, 1937]. Therefore it may be more effective or intuitive for users to write with an interface that operates on words directly.



### 6.3.14 Specification vs. Navigation

Specification vs. navigation concerns users' mental model of the text entry method. A specification-based method strives to internalize in muscle memory the patterns involved in articulating a character or word. Almost all text entry methods fall into this category, e.g. traditional software keyboards, thumb keyboards, Unistrokes, EdgeWrite and continuous shape writing. However, some text entry methods are inherently dynamic. Examples are Dasher and adaptive prediction. These text entry methods rely on a visual-feedback loop between the system and the user. The most extreme example is probably Dasher that provides an explicit "driving" model where the user drives the cursor to the desired destination. Because the driving path is always different, Dasher cannot rely on users internalizing articulations for specific characters or words in their muscle memory. Rather, Dasher must rely on expert users gaining increased hand-eye coordination and reading ability skills.

Interestingly, this analysis suggests that this design dimension splits text entry methods into two orthogonal categories.

The first category attempts to minimize information transfer from the human motor control system to the computer at any given point time. Methods such as Dasher and adaptive language modeling achieves this by recognizing that most motor actions achievable by users are in fact redundant and should be minimized. For example, in Dasher the cursor is merely steered toward a target, and depending on the target's likelihood the target's area is greater and can be reached faster with the cursor.

In contrast, the second category does not attempt to minimize information transfer from the human motor control system to the computer in a context dependent manner. Rather, the information transfer is minimized by the designers of the text entry system in the design stage. A straight-forward example is Unistrokes [Goldberg and Richardson, 1993] whose alphabet is designed with minimal redundancy to reduce users' required articulation times for single letters. This basic idea is implicitly or explicitly present in almost any text entry invention. As a result of only relying on this initial optimization, at any point in time the information transfer from the user to the system cannot be optimal because the user's text input signal is fixed and context-independent. In return, the user can during usage gradually and partially internalize the articulation for a characters or word into muscle memory and enable text entry that is primarily based on open-loop recall from memory.

The suitability of either approach depends on the user's and the device's abilities. For example, a motor-impaired user is more likely to benefit from a navigation-based text entry method. A healthy individual may also benefit more from a navigation-based method if the input mechanisms on the device are limited, for example if the input mechanism is a joystick. Outside these hypothetical situations a specification-based method is more likely to be faster because of eventual fast open-loop articulation.

### 6.3.15 One-Handed vs. Two-Handed

A mobile text entry method should be versatile. A text entry method that can be manipulated with one handed instead of two is more flexible to a user on the move. Therefore, *ceteris paribus* a one-handed text entry method is preferred.

### 6.3.16 Task Integration

The efficacy of a mobile text entry method is not only restricted to the method itself. To be practical the method needs to be tightly integrated with the operating system and device it is used on. For character-level text entry methods this poses less of a problem since almost all operating systems and devices that need text entry readily supports the concept of single-character input (e.g. a series of keyboard scan codes). More advanced text entry methods that rely on lexicons or language models may have more difficult issues with integration. For instance, if the text entry method relies on some form of pattern recognition, it is beneficial to let users choose from the next best candidates in case the user's intended word did not get properly recognized. However, most operating systems and devices do not have built-in facilities to directly support this behavior, and it can be difficult or even impossible to implement it without consulting the operating system or hardware vendor directly.

### 6.3.17 Robustness

Robustness is a measure of how much external noise the text entry method can handle while maintaining acceptable entry and error rates. Since mobile text entry methods are intended to be used on the go, it is desirable they withstand some external noise such as the user walking along a corridor while writing, or writing an email while using public transportation. To my knowledge little studies have been conducted where participants were in a truly mobile setting. One of the closest studies I am able to find is Crossan, Murray-Smith, Brewster, Kelly and Musizza [2005] who investigated how stylus pointing performance on a handheld computer was impacted when participants were walking. As expected Crossan et al. [2005] observed lower response times and higher error rates when participants were walking in comparison to when they were sitting down. For the smallest targets that had a diameter of 5 pixels Crossan et al. [2005] observed 43.5% error rate when participants were walking in comparison to 22% when they were sitting down. Not until targets' diameters were expanded to 25 pixels did the error rate converge between the two conditions. As a reference point the 3.8" touch-screen used in the experiment had a screen resolution of 240 x 320 pixels (these specifications are derived from Hewlett-Packard product information booklet about the test device used in Crossan et al. [2005]). No statistical significances were calculated from the collected data material. Never the less, their results appear to suggest that pointing tasks suffer a considerably increased error rate when participants are walking. The reported data suggests that software keyboards without correction algorithms, such as for example discrete shape writing, are not going to be robust to users typing text on the move.

Another study by Price, Lin, Feng, Goldman, Sears and Jacko [2006] examined how speech recognition performance is affected when users are on the move. They found a significant increase in error rates when participants were walking.

Empirical evidence of the effect of robustness on text entry performance suggests that as soon as the user starts moving, performance decreases. It is interesting that an interface such as speech recognition that should be relatively insulated from motor control interference in comparison to systems that require hand-eye or hand coordination, still suffers significantly in error rate when users are on the move [Price et al., 2006].

### **6.3.18 Device Independence**

In general, a text entry method is more useful if it can be used on many varying mobile devices: from joysticks, small mobile phones with the size of a match box, to larger portable multimedia devices. Isokoski [2004a] advocates the usefulness of device independent text entry systems. Unfortunately, device independent text entry methods proposed in Isokoski [2004a] are too slow to be practical. Therefore, tasks that demand a significant amount of text writing, such as writing emails, are likely to force users to use non-device independent text entry methods.

### **6.3.19 Computational Demands**

Mobile text entry methods are almost always targeted towards small handheld devices. These small computers have typically simultaneously less processing power and less memory than their desktop counterparts. They also often lack a floating-point unit (FPU), which causes problems when implementing text entry methods that involve many floating-point calculations. As a result, pattern-recognition based text entry methods like handwriting and speech recognition can be difficult to implement without impeding performance and user experience. A real-world example is the study by Price and Sears [2005] on speech recognition for mobile devices. Because of limited computational power speech recognition must be off-loaded to a server. Price and Sears [2005] therefore introduced network latency as a factor affecting entry rate.

### **6.3.20 Manufacturing Cost**

Some mobile text entry methods require significant manufacturing costs. For instance, physical key-based solutions require factory assembly. Touch-screen methods obviously require a touch-screen. Since mobile phones and other mobile devices usually already have a screen and physical buttons, text entry methods founded on those assumptions may incur little or no extra manufacturing cost, except possibly licensing. A method based on the assumption that a touch-screen is present, does not incur extra manufacturing cost if the device is multimedia device, such as a video phone or mobile movie player.

### 6.3.21 Support Cost

Support costs should not be neglected. Text entry methods that are not intuitive to new users do most likely increase support costs for the producer. Some methods can result in a useless device for the user if no appropriate fall-back text entry method is implemented. For example, a too small thumb keyboard may be unusable for a person with large thumbs. Other methods may increase the risk of damaged hardware. For example, as discussed when presenting software keyboards, repeated tapping on the screen with a pen can cause damage the touch-screen.

### 6.3.22 Market Acceptance

Belonging to an engineering science discipline, text entry method inventors should focus on real world impact. Is the proposed text entry method practical? Or in other words: does the proposed text entry method satisfy or tradeoff the above given design dimensions in such a way that it is believable users would want to use the method? As an example, a text entry method that offers users four wpm average entry rate is not practical, unless the device is severely limited in functionality. A new text entry method should significantly exceed the capabilities of an existing method in at least some design dimension, preferably many.

In a sense text entry research is an engineering science and driven by incremental improvements to existing techniques. Are incremental improvements to text entry enough of an incentive for users to switch?

David [1985] argue that path dependency and “one damn thing follows another” resulted in QWERTY not being adopted in favor of the Dvorak Simplified Keyboard. Essentially, the argument goes that because of the QWERTY’s strong market presence the cost of switching layout and re-training staff was too high, even though the Dvorak Simplified Keyboard in the long run might be more efficient. Liebowitz and Margolis [1990] questions this view because the scientific evidence that Dvorak provided when arguing that the Dvorak Simplified Keyboard offered a substantial improvement over QWERTY typing is unconvincing. Therefore Liebowitz and Margolis [1990] argues that the QWERTY layout has prevailed because it is good enough for most users and the enormous re-training cost is not worth it.

In the end, the only way to find out if a text entry method will succeed in the market is to put it out there. Obviously business skills, marketing, popular press coverage, timing and luck are significant factors.

## 6.4 Text Entry Methods in Design Space

---

The above mentioned design dimensions serve to give a wider perspective on the many and often conflicting dimensions a novel text entry method must endure. Some dimensions are inherently related, most notably entry rate and error rate. Other dimensions, such as localization and device independence are less critical since they

only affect how general the text entry method can be applied in other contexts. There is no reason to demand that a text entry method suitable for a particular language and device should be effective for all purposes and all languages.

#### **6.4.1 What Matters Most?**

The above analysis suggests not all design dimensions are equally important. Assuming the form factor of the text entry method is sufficiently small to qualify the text entry method as “mobile”, this section argues that, in general, five specific design dimensions are more critical than others.

First, a text entry method that is not perceived as efficient from the start, i.e. does not have immediate efficacy, does not gain traction from users. Therefore such a text entry method has to be exceptionally strong to gain wide acceptance among users. For example, the Dvorak Simplified Keyboard has not been widely adopted even though there are claims that it is both faster and more comfortable than QWERTY. However, to invest in Dvorak the user needs to invest a significant amount of time to re-learn touch typing on the new layout. Because the Dvorak layout is not commonly used few users apparently felt the required amount of invested work would be worth it. See David [1985] and Liebowitz and Margolis [1990] for an in-depth discussion on the controversy of Dvorak vs. QWERTY.

Two other important design dimensions are entry rate and error. Because of the inherent speed-accuracy tradeoff in human performance in general, and text writing in particular, these design dimensions cannot be separated out. Assuming errors are within a tolerable level among users, entry rate should naturally be as fast as possible. A higher entry rate allows user to focus on text composition rather than being undermined by user interface limitations.

Three other critical design dimensions are comfort, the learning curve, and how engaging the text entry method is perceived by users

An uncomfortable method is inadequate if there are better alternatives. The telephone keypad on mobile phones is not comfortable for extended text messaging use, yet text messaging is highly popular. Since no compelling alternatives compatible with the mobile phone form factor has attracted the interests of mobile phone manufacturers, users have to suffer.

The learning curve of a text entry method has some importance, especially if the text entry method is underperforming initially. The learning curve from floor to ceiling entry rate should preferably be as short as possible. However, provided users quickly reach an acceptable entry rate, the amount of learning required to reach ceiling entry rate may not matter that much.

Finally, a perhaps underestimated design dimension is to what level the user is engaged in using the method, at least initially. In Experiment 3.2 participants liked writing with continuous shape writing more than thumb keyboard. They also thought it was more fun to use continuous shape writing. Such subjective ratings should be interpreted with care because the low sample size may have influenced the outcome. Never the less, if a text entry method is perceived as more fun by users initially the chance is higher that users put up with an initial practice period with lower text entry performance than desirable.

Clearly, many of these dimensions and their relative ranking are not rigorous or scientific. There is no easy, or even possible, method to quantify, measure and rank design dimensions such as comfort and engagement except by subjective ratings because these design dimensions are likely composite design dimensions affected by many elements, some unknown. There is also no clear cut criterion held by every individual user on the level of immediate efficacy. The point in listing these design dimensions and listing them in a particular order is to set a context for discussion. What can be confidently said is the following. First, you cannot optimize for one design dimension alone. Second, a text entry method should be designed and evaluated with consideration of all design dimensions, but it is not necessary, or even desirable, to prioritize all design dimensions equally.

#### **6.4.2 Binary Decision Design Dimensions**

Two design dimensions stand out because they can be used to form a binary yes or no decision on whether the text entry method is applicable for particular requirements.

If preparation time is a significant component, for example if the device is a mobile phone, a text entry method with long preparation time is unacceptable. This decision would immediately rule out fold-up and projection keyboards.

If the text entry method must be able to be used one-handed, a two-handed text entry method can be immediately ruled out. In practice, while one-handed text entry is preferred in a mobile situation, if a text entry method can offer benefits that are beyond any one-handed technique (for example: thumb keyboard vs. keypad) the decision to go with a one-handed or two-handed text entry method must be guided by a design tradeoff.

#### **6.4.3 Guiding Design Dimensions**

The visual attention and cognitive resources design dimensions are important but partly subsumed into error and entry rate. The lack of empirical studies of text entry methods' visual and cognitive demands from users makes it impossible to perform a reliable comparison between existing text entry alternatives.

Also, visual attention and cognitive resources are design dimensions that are more useful as design dimensions that guide the design, than as comparative dimensions

between several text entry methods. Severe visual and cognitive overhead will manifest itself in quantitative measurements via reduced entry rate or increased error rate.

Another guiding design dimension is device independence, advocated by Isokoski [2004a]. No effective text entry method is completely device independent yet. Isokoski and Raisamo [2004] investigates Quikwriting as a basis for device-independent text entry but finds that it is too slow to be practical. To some extent the thumb keyboard and QWERTY software keyboard is device independent because of the ubiquity of the QWERTY layout. Although there are not exactly the same skills involved in touch typing on QWERTY and touch typing on thumb keyboard, it is plausible there is a high degree of skill transfer in between the systems. Further, MacKenzie and Zhang [1999] shows that even between the desktop QWERTY keyboard and the software QWERTY keyboard there is a skill transfer, because users initially typed faster on the software keyboard with the QWERTY layout than the unfamiliar optimized layout (OPTI).

Finally, a fundamental guiding design dimension is whether the text entry method is specification or navigation-based. As alluded earlier, navigation-based text entry may be better if the underlying input device mechanism is limited in expressiveness, or users' suffer from some level of motor impairment.

#### **6.4.4 Implementation and Commercially Related Design Dimensions**

Design dimensions such as task integration and character vs. word-level text entry concern the practical implementation efforts that need to be devoted to successfully transplant a text entry method onto a mobile device. These design dimensions can be viewed as tradeoffs – it is in general easier to integrate a single-character text entry into a device because devices normally have single-character keyboard support built-in.

Required localization efforts vary from method to method. Localization can be expensive but due to tight coupling between text entry and varying languages and cultures, localization overhead is almost impossible to completely avoid.

Computational demands, support cost and manufacturing cost are mostly a matter of how much investment that can be poured into the assembly and customer support of the mobile device. Such design dimensions are to some extent critical, but the virtues of these design dimensions must be determined on a case-by-case basis. Many text entry methods are software-based and designed to take advantage of certain common mobile device designs: keypads, touch-screens, etc. These text entry methods do not add any assembly cost (except possibly licensing) to a device. On the other hand, keypads and thumb keyboards can be assembled relatively cheaply in a highly optimized process (e.g. [Liu and Wu, 2006]).

Market acceptance is essentially impossible to predict. However, a text entry method that demands extensive training effort from users initially but only offers an incremental performance benefit will have a hard time to gain traction.

#### 6.4.5 Floor Entry Rate Comparison of Text Entry Methods

Is it possible to quantify the design dimensions and directly compare mobile text entry methods against each other? Some design dimensions do not lend themselves to direct comparison. For example comfort, although not a completely subjective measure is not studied enough to be reliably used to quantitatively compare different text entry methods against each other.

However, entry and error rates are frequently reported in the literature. Therefore it is possible to select a subset of text entry methods and perform a quantitative analysis. In practice, this process is not as simple as it sounds because there is no de-facto standard experimental procedure for text entry evaluation. Such an experimental procedure would also be complicated, perhaps impossible, to develop because of the creative nature of text entry inventions and the different views and motivations that guide the inventors and researchers.

Because there has been no consensus on participants' error handling and how error rate is reported, it is problematic to compare the error rates from different text entry experiments. Further, entry and error rates are related in a speed-accuracy tradeoff, so entry rate comparisons are also questionstionable when error rates were not the same.

Never the less, this section attempts to perform a baseline comparison of the floor entry rates achievable among a representative selection of text entry methods. Because the empirical material only allows a low resolution comparison, the floor entry rate measure is divided into three broad categories: low (effectively in the range of 0-15 wpm), medium (effectively in the range of 15-25 wpm) and high floor entry rate (effectively in the range of 25+ wpm). In the assignment of a text entry method to one of these categories a subjective judgment is made on to what extent error rate impacts the categorized floor entry rate.

Table 6.1. Floor entry rate estimation for text entry methods.

Text Entry Method	Floor Entry Rate
Graffiti	Low
Quikwriting	Low
Multi-tap	Low
T9	Low
EdgeWrite with Completions	Low
Half-QWERTY	Low
Twiddler	Low
Software keyboard (QWERTY)	Medium
Software keyboard (OPTI)	Low/Medium
Thumb keyboard	High



Continuous shape writing (QWERTY)	Medium
Continuous shape writing (ATOMIK)	Low
Dasher	Low

The Quikwriting measure is based of the initial 15-minutes of data reported in Isokoski and Raisamo [2004] where participants reached around 5 wpm with low error rate. T9 and EdgeWrite with completions measures are based on data from [Wobbrock et al., 2007]. Both methods had low error rate but also low entry rates during an initial 20-minute session (around 12.5 wpm for T9 and 10 wpm for EdgeWrite with completions). The half-QWERTY measure is from Matias et al. [1996] that reports of an entry rate around 13 wpm during the first 50 minutes of practice with an uncorrected error rate of 15.16%. The twiddler measure is from Lyons et al. [2006]. Lyons et al. [2006] reports of a low entry rate for the initial 20-minute session (around 6 wpm). The “total error rate” measure is problematic in Lyons et al. [2006] but the entry rate reported is so low that it doesn’t matter for this broad categorization. The software keyboard (QWERTY) and software keyboard (OPTI) measures are from [MacKenzie and Zhang, 1999]. Software keyboard (QWERTY) is classified as having a medium floor entry rate despite MacKenzie and Zhang [1995] reports that after 20 minutes of practice participants reaches an average entry rate > 25 wpm, which is in comparison to other mobile text entry methods is very high. This is because the uncorrected error rate is around 3% in [MacKenzie and Zhang, 1999], which is a three times higher error rate than what is found for T9, EdgeWrite with completions, continuous shape writing and thumb keyboard (all with uncorrected error rates around or less than 1%). The software keyboard (OPTI) is classified as having low/medium entry rate because average entry rate after 20 minutes of practice is > 20 wpm, while error rate is around 2%, which is twice as high error rate as the one found in Experiment 3.2. The thumb keyboard, continuous shape writing (QWERTY) and continuous shape writing (ATOMIK) measurements are from (Experiment 3.2). Thumb keyboard is classified as a having high floor entry rate because Experiment 3.2 showed that participants reaches > 25 wpm while the error rate was around 1%. Continuous shape writing (QWERTY) has medium floor entry rate because the average entry rate during the first 20 minutes is > 20 wpm, while the error rate is around 1%. The Dasher measure is estimated from Figure 14 in Ward et al. [2002]. Dasher is classified as having low floor entry rate because after 20 minutes participants on average only reached around 10 wpm.

The floor entry rates for the above selection of text entry methods are fairly comparable because most studies were conducted on open text. The exception is the multi-tap and Graffiti studies, however these have been independently investigated by many researchers and all studies have found that both Graffiti and multi-tap are relative slow. The reader is referred back to the survey sections on Graffiti and multi-tap for more details. All the other text entry method measurements stem from studies with approximately the same setup of participants writing open text phrases taken

from a corpus. The T9, EdgeWrite with completions, thumb keyboard, continuous shape writing (QWERTY) and continuous shape writing (ATOMIK) use the same corpus [MacKenzie and Soukoreff, 2003]. Half-QWERTY used phrases taken from an undisclosed novel. The software keyboard (QWERTY) and software keyboard (OPTI) used phrases from an undisclosed phrase set. Dasher used phrases taken from the novel *Emma*.

From the floor entry rate analysis the thumb keyboard and the software keyboard, including continuous shape writing (on QWERTY), emerge as the most promising text entry methods with medium or high floor entry rates. The high initial entry rate benefits immediate efficacy, which as discussed in the Immediate Efficacy design dimension section is important, perhaps more important than the learning curve. Graffiti, Quikwriting, multi-tap, T9 and EdgeWrite with completions, half-QWERTY, twiddler, continuous shape writing (ATOMIK) and Dasher are the lowest performing text entry methods.

#### 6.4.6 Ceiling Entry Rate Comparison of Text Entry Methods

Similar to the previous section, this section attempts to perform a baseline comparison of the ceiling entry rates achievable among a representative selection of text entry methods. Because the empirical material only allows a low resolution comparison, the ceiling entry rate measure is divided into three broad categories: low (effectively in the range of 0-20 wpm), medium (effectively in the range of 20-40 wpm) and high ceiling entry rate (effectively in the range of 40+ wpm). In the assignment of a text entry method one of these categories a subjective judgment is made on to what extent error rate impacts the categorized ceiling entry rate.

Table 6.2. Ceiling entry rate estimation for text entry methods.

Text Entry Method	Ceiling Entry Rate
Quikwriting	Low
Multi-tap	Low
T9	Low
EdgeWrite with Completions	Low
Half-QWERTY	Medium
Twiddler	Medium
Software keyboard (QWERTY)	Medium
Software keyboard (OPTI)	High
Thumb keyboard	High
Continuous shape writing (QWERTY)	High
Continuous shape writing (ATOMIK)	High

The Quikwriting measure is based of the 20 15-minutes sessions of data reported in Isokoski and Raisamo [2004] where participants reached around 15 wpm with low error rate. T9 and EdgeWrite with completions measures are based on data from [Wobbrock et al., 2007] where participants reached 15 wpm with both methods after

had low error rate an average entry rate of 15-16 wpm after 15 20-minute sessions (error rate < 1%). The half-QWERTY measure is from Matias et al. [1996] that reports of an entry rate around 34.7 wpm and an uncorrected error rate of 7.37% after 10 50-minute sessions. The half-QWERTY is assigned the medium category because of the high error rate combined with the fact that the entry rate is well below 40 wpm. The twiddler measure is from Lyons et al. [2006]. Lyons et al. [2006] reports of an entry rate of 37.3 wpm after 13 hours of practice. The “total error rate” measure of 6.2% is problematic in Lyons et al. [2006]. Taking both entry rate and the high error rate in consideration the twiddler is assigned the medium category. The software keyboard (QWERTY) and software keyboard (OPTI) measures are from [MacKenzie and Zhang, 1999]. After 20 20-minute sessions participants reached an average entry rate of 40 wpm with the software keyboard (QWERTY). However, the uncorrected error rate was 4.8%. In consideration of both these measures the software keyboard (QWERTY) is assigned the medium category but is borderline in the high category. After 20 20-minute sessions participants reached an average entry rate of 45 wpm and had an uncorrected error rate of 4.2% with the software keyboard (OPTI). Taking both entry and error rate into consideration the software keyboard (OPTI) is classified as having a high ceiling entry rate. The thumb keyboard, continuous shape writing (QWERTY) and continuous shape writing (ATOMIK) measurements are from (Experiment 3.3). Thumb keyboard, continuous shape writing (both QWERTY and ATOMIK) are classified as a having high ceiling entry rate because Experiment 3.3 showed that participants on average managed to write > 40 wpm with no errors.

#### **6.4.7 Comparison Redux – Non Quantitative Design Dimensions**

Not all design dimensions have a strong quantitative foundation. Never the less, as will soon be apparent they can still impact the choice of a suitable text entry method. In Table 6.3 five auxiliary design dimensions are listed: form factor, user engagement, one vs. two-handed, specification vs. navigation, and robustness. These design dimensions are chosen because they are non-quantitative design dimensions that directly impact the user experience. Because comfort is understudied and therefore any judgment of comfort would be arbitrary, comfort is left out of the following comparison. The design dimensions considered are to some extent highly subjective. For example user engagement cannot be reliably measured in a small sample controlled experiment and much less compared across different experiments. Software keyboards are assigned low user engagement because research has revealed that users find them tedious to use [Zhai, Sue and Accot, 2002]. The thumb keyboard is assigned medium user engagement and continuous shape writing assigned high, based on the results obtained in Experiment 3.2 and 3.3. EdgeWrite with completions is assigned high user engagement because users’ preferred it in favor of T9. T9 is assigned medium user engagement because there is no user study where T9 is found to be very tedious. Quikwriting is assigned medium user engagement based on my

own estimate. The reader is advised to re-consider and perhaps re-assign the pre-assigned design dimension values for the text entry methods in Table 6.3.

Table 6.3. Comparison of non-quantitative design dimensions.

<b>Text Entry Method</b>	<b>Form Factor</b>	<b>User Engage.</b>	<b>One/Two Handed</b>	<b>Spec. vs. Nav.</b>	<b>Robust.</b>
Quikwriting	Medium	Medium	Two	Spec.	Medium
T9	Medium	Medium	One	Spec.	High
EdgeWrite with Completions	Small	High	One	Spec.	High
Half-QWERTY	Large	Medium	Two	Spec.	Medium
Twiddler	Large	Medium	One	Spec.	Medium
Software keyboard (QWERTY)	Medium	Medium	Two	Spec.	Low
Software keyboard (OPTI)	Medium	Medium	Two	Spec.	Low
Thumb keyboard	Medium	Medium	Two	Spec.	Medium
Continuous shape writing (QWERTY)	Medium	High	Two	Spec.	Medium
Continuous shape writing (ATOMIK)	Medium	High	Two	Spec.	Medium
Dasher	Medium	High	Two	Nav.	Low

From Table 6.3 it is apparent that if very small form factor is required only EdgeWrite with Completions is applicable (within this subset of text entry methods). Also, EdgeWrite with completions is estimated to have the highest robustness because the EdgeWrite alphabet is explicitly designed for robustness [Wobbrock et al., 20003] and error rates with the isometric joystick-EdgeWrite were found to be low [Wobbrock et al., 2007]. However the actual robustness of the text entry method remains to be empirically verified. Price et al. [2006] results show that as soon as users are no longer stationary, error rates go up significantly for speech recognition, which suggests methods that require skilled motor behavior, such as joystick operation, may suffer even more in an actual empirical test situation.

If a navigation-based text entry method is desired, for example if the input mechanisms available on the device are severely limited, Dasher is the only alternative among selected subset of text entry methods

In relation to the quantitative comparison both EdgeWrite and Dasher have rather low performance (no ceiling entry rate is available for Dasher). However, from the preceding analysis it is evident that depending on the requirements, other text entry methods with higher entry rates may not be applicable.

Another interesting design dimension to consider is one-handed vs. two-handed interaction. All the quantitatively highest performing text entry methods require two hands. In contrast, the twiddler only requires one hand and has a competitive ceiling entry rate  $> 35$  wpm. On the other hand, twiddler has a low floor entry rate.

Clearly, different design dimensions can be traded off against others. In the end, the device and user experience requirements set the stage for a text entry method. From there, the text entry method should optimize quantifiable entry and error rates, or reconsider the basic assumptions if acceptable entry and error rates cannot be obtained.

## 6.5 Conclusions

---

This chapter introduces 22 design dimensions of mobile text entry. The analysis in this chapter shows that text entry development, evaluation and analysis is complex and multifaceted. Partly because of the diverse nature of mobile devices with a wide-array of auxiliary input mechanisms and displays, text entry methods tend to favor different design dimensions, which in turn create difficulties on how to fairly evaluate text entry methods against each other.

This chapter makes many implicit and hidden design dimensions in text entry research explicit. Some of these dimensions are highly subjective. However, this is not an excuse for arbitrary decisions, and subjectivity should not be a guiding principle if it can be at all avoided. By making hidden assumptions of all aspects of text entry research visible, these implicit and subjective design dimensions are forced up to the surface and require explicit motivation on how they impact the overall text entry method. It is easy in text entry to get blinded by entry rate alone.

In summary, text entry method design is an engineering science and should be guided by fundamental principles. In this chapter I have attempted to construct and apply as much principled analysis of existing and in-development text entry methods as the state of the art allows.

In a computer science essay celebrating ACM's 50-year anniversary Dijkstra [1997] warns of watching the waves, and completely missing the tide. His example is the concept of program correctness where computer programs are mathematically modeled and verified as behaving correctly, rather than constantly patched and debugged without any higher guiding principles.

Given the multitude of incomparable entry and error rates in the literature, can we learn anything from Dijkstra? A first step towards a principled text entry field should be an effort to standardized experimental procedure, or at least standardized entry and error rate reporting. Further, the analysis of the design dimensions in this chapter has highlighted the many, and often conflicting, tradeoffs that are necessary to take when devising a text entry method. New text entry methods that are proposed should ideally be justified with regards to a weighted combination of these design dimensions.

# Chapter 7

## Conclusions

---

This chapter is organized as follows. The first section summarizes the dissertation by answering the research questions posed in the introductory chapter. The second section presents limitations and future work. The last section marks the end of this dissertation with concluding remarks.

### 7.1 Summary

---

The introductory chapter states that the central hypothesis in this dissertation is that it is possible to combine three elements: software keyboard, language redundancy and pattern recognition, and create new effective interfaces for text entry and control.

In relation to the research questions posed in the introductory chapter, this section proceeds by addressing each of them with references to the respective chapters and experiments where the research questions are answered in full.

1. How are effective discrete and continuous shape writing systems engineered – from the user interface to the recognition algorithms?

Chapters 2 and 3 describe how the user interfaces for discrete and continuous shape writing are designed. Chapter 4 describes the mathematical principles for effective continuous shape writing recognition.

2. How effective are the discrete and continuous shape writing interfaces for text entry?

Experiment 2.1 and 2.2 showed that linear discrete shape writing did not significantly increase entry rate or reduce error rate. Informed by these experiments the elastic discrete shape writing algorithm was devised. Experiment 2.3 indicated some possible advantage of discrete shape writing with this new algorithm for experts. An analysis of the errors committed by participants in Experiment 2.1 show that many of these errors could also be corrected with the new elastic discrete shape writing algorithm. Taken together these results suggest that discrete shape writing may be more useful in reducing errors rates than gaining significant entry rate improvements.

Experiment 3.1 showed that, in an expanding rehearsal interval training paradigm, users learn on average 15 shapes for words in a 40 minute practice session, Experiment 3.2 showed that entry rate depended on the layout used for the software

keyboard. The QWERTY layout resulted in significantly faster entry rates than the optimized ATOMIK layout. With QWERTY, users had an average entry rate of 25 wpm and an error rate around 1% after 35 minutes of practice. With ATOMIK, users had an average entry rate of 16.6 wpm and an error rate around 1% after 35 minutes of practice. It is not surprising QWERTY is initially faster than ATOMIK, given that users are familiar with the desktop keyboard QWERTY layout. In contrast to QWERTY, when using continuous shape writing on ATOMIK users are involved in a dual learning task where both the keyboard layout and the text entry method are novel to the user.

Experiment 3.3 investigated the effect of accelerated novice performance. Experiment 3.3 showed that novice users reached an average entry rate of 46.5 wpm with shape writing on QWERTY and 45.1 wpm with shape writing on ATOMIK with no errors. It is important to be aware that accelerated novice performance is not the same as expert performance or accelerated expert performance. The potential entry rate may be significantly higher than what was measured in Experiment 3.3.

3. Can shape writing also be used as a control interface, and if so, how effective is it?

Chapter 5 systematically examines methods to extend continuous shape writing to become an effective control interface. Experiment 5.1 shows that as a control interface, continuous shape writing is 1.6 times faster than pull-down menus.

Experiments 5.2 and 5.3 investigate the effect of displaying a visual preview of the currently recognized command while the user is still articulating the shape for the command. Experiment 5.2 shows that visual preview does not slow users down and Experiment 5.3 shows that command preview leads to significantly lower error rates and shorter gestures when users enter new unpracticed commands.

## **7.2 Limitations and Future Work**

---

This dissertation has shown that it is possible to construct software systems for both discrete and continuous shape writing that are practical and effective. The user interface and recognition technology presented represents the first stage of the shape writing evolution and will no doubt continue to evolve when the software systems are further investigated or delivered to users as a product.

A limitation with the work is that Experiment 3.2 did not continue over several sessions (say 20 sessions). Therefore the extended learning curve of continuous shape writing is unknown. It is plausible the optimized keyboard layout (ATOMIK) would outperform the QWERTY layout with extended practice because MacKenzie and Zhang [1999] showed that users eventually typed faster with an optimized layout (OPTI) than QWERTY. However, because Experiment 3.2 was short, it cannot be concluded whether shape writing is faster on ATOMIK or QWERTY, and if so, where the turnover point is. It is also not possible to tell how fast users write with either discrete or continuous shape



writing after extended practice. Because of the power-law of practice [Crossman, 1959] it is likely that users write much faster than the entry rates measured after only 35 minutes of practice.

Further, after Experiment 3.2 was conducted my thinking about how to handle errors has changed. In Experiment 3.2 participants were not forced to correct errors because forcing participants to correct errors is psychologically demanding and participants can perceive the task as frustrating. On the other hand entry rates can only be practically compared across studies if no errors were allowed. Between the two evils, it may be necessary to force participants to correct their errors because it is not clear how to compare text entry methods if they do not have the same uncorrected error rate. For example, if one text entry method reaches 10 wpm with a 1% uncorrected error rate and another text entry method reaches 15 wpm with a 2% uncorrected error rate, which one of these two text entry methods is better? Since there is no known methodologically sound method for converting uncorrected errors into entry rate, these two methods cannot be reliably compared. Therefore, the only reasonable approach is to force participants to correct all errors.

The most important future work is to conduct a longitudinal learning curve study to find out how fast participants write after many sessions of practice. After Experiment 3.2 was conducted continuous shape writing has been ported to mobile phones. Running a longitudinal experiment on a mobile phone is more interesting than running on a tablet computer because mobile phones are more common, and portable.

### **7.3 Concluding Remarks**

---

In the 20<sup>th</sup> century the typewriter and the desktop keyboard out-competed the pen as the primary text entry device. Pens are versatile and fluent, but crippled by the slow speed of longhand writing. The long and fascinating history of shorthand reveals that devising an effective pen-based text entry method is very hard. Also, as is shown in Chapter 6, all pen-based text entry methods for computers, except the software keyboard, are slow. However, the standard software keyboard is a text entry method where expressive and fluid pen-gestures are replaced with tedious serial tapping, and thereby removes the point of having a pen-based interface in the first place.

The work in this dissertation is a renewal of the pen interface. This dissertation has three main messages. First, it is possible to construct a pen-based text entry and control interface that is simultaneously fluid, fast and easy to learn. Prior to this dissertation only shorthand systems that were not machine-recognizable could fulfill two of these criteria simultaneously (shorthand was never easy to learn). Second, the immediate efficacy of a method, that is, users' initial performance, is more important than the rate of increase of the learning curve. Continuous shape writing has immediate efficacy and thereby demonstrates that a text entry interface that is relatively complex and completely novel to users can still be effective from the start.

Third, text entry methods have complex and often conflicting requirements. Among 22 design dimensions of mobile text entry, continuous shape writing delicately trades off these design dimensions. Continuous shape writing is more challenging to integrate with mobile devices because of novel error correction strategies. It also requires more computational power than simpler methods. Further it demands visual attention from users. On the other hand, it has a high entry rate and low number of uncorrected errors. Further, users are writing effectively with shape writing after minimal amount of practice and find it comfortable and engaging.

# References

---

- BROOKS, C.P. AND NEWELL, A.F. 1984. Computer transcription of handwritten shorthand as an aid for the deaf – a feasibility study. *International Journal of Man-Machine Studies* **23**(1): 45-60.
- BURR, D.J. 1981. Elastic matching of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **3**(6): 708-713.
- BUTTER, A. AND POGUE, D. 2002. *Piloting Palm: The Inside Story of Palm, Handspring and the Birth of the Billion Dollar Handheld Industry*. New York: Wiley.
- BUTTS, L. AND COCKBURN, A. 2002. An evaluation of mobile phone text input methods. *Australian Computer Science Communications* **24**(4): 55-59.
- BUXTON, W. 1986. Chunking and phrasing and the design of human-computer dialogues. In *Proceedings of the 10<sup>th</sup> IFIP World Computer Congress*. IFIP: 475-480.
- CALLAHAN, J., HOPKINS, D., WEISER, M. AND SHNEIDERMAN, B. 1988. An empirical comparison of pie vs. linear menus. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '88)*. ACM Press: 95-100.
- CARAU, F.P. 2001. Method and apparatus for a virtual display/keyboard for a PDA. U.S. Patent 6,266,048.
- CARD, S.K., ENGLISH, W.K. AND BURR, B.J. 1978. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics* **21**(8): 601-613.
- CARD, S.K., MORAN, T. AND NEWELL, A. 1983. *The Psychology of Human-Computer Interaction*. Hillsdale: Lawrence Erlbaum.
- CHÁVEZ, E., NAVARRO, G., BAEZA-YATES, R. AND MARROQUÍN, J.L. 2001. Searching in metric spaces. *ACM Computing Surveys* **33**(3): 273-321.
- CLARKSON, E., CLAWSON, J., LYONS, K. AND STARNER, T. 2005. An empirical study of typing rates on mini-QWERTY keyboards. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. ACM Press: 1288-1291.

- CLARKSON, E., LYONS, K., CLAWSON, J. AND STARNER, T. 2007. Revisiting and validating a model of two-thumb text entry. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '07)*. ACM Press: 163-166.
- COCKBURN, A. AND SIRESENA, A. 2003. Evaluating mobile text entry with the Fastap keypad. In *People and Computers XVII: British Computer Society Conference on Human Computer Interaction (BCS HCI '03) 2*. British Computer Society: 77-80.
- COULON, K.E. AND MALHI, S.D.S. 1998. Compact foldable keyboard. United States Patent 5,712,760.
- CROSSAN, A., MURRAY-SMITH, R., BREWSTER, S., KELLY, J. AND MUSIZZA, B. 2005. Gait phase effects in mobile interaction. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. ACM Press: 1312-1315.
- CROSSMAN, E.R.F.W. 1959. A theory of the acquisition of speed skill. *Ergonomics* **2**: 153-166.
- CURRAN, K., WOODS, D. AND RIORDAN, B.O. 2006. Investigating text input methods for mobile phones. *Telematics and Informatics* **23**(1): 1-21.
- DARRAGH, J.J., WITTEN, I.H. AND JAMES, M.L. 1990. The reactive keyboard: a predictive typing aid. *IEEE Computer* **23**(11): 41-49.
- DAVID, P. 1985. Clio and the economics of QWERTY. *American Economic Review* **75**(2): 332-337.
- DETWELER, M.C., SCHUMACHER, R.M. AND GATTUSO, N.L. 1990. Alphabetic input on a telephone keypad. In *Proceedings of the Human Factors Society 34<sup>th</sup> Annual Meeting (HFES '90)*. Human Factors and Ergonomics Society: 212-216.
- DIAMOND, T.L. 1957. Devices for reading handwritten characters. In *Proceedings of the Eastern Joint Computer Conference (EJCC '57)*. American Federation of Information Processing Societies: 232-237.
- DIJKSTRA, E. 1997. The tide, not the waves. In Denning, P.J. and Metcalfe, R.M. (Eds.), 59-65, *Beyond Calculation: The Next Fifty Years of Computing*. New York: Springer-Verlag.
- DUDA, R.O. AND HART, P.E. 1973. *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons.
- DUDA, R.O., HART, P.E. AND STORK, D.G. 2001. *Pattern Classification, 2<sup>nd</sup> edition*. New York: John Wiley & Sons.

- FAGIN, R. AND STOCKMEYER, L. 1998. Relaxing the triangle inequality in pattern matching. *International Journal of Computer Vision* **28**(3): 219-231.
- FENDRICK, P. 1937. Hierarchical skills in typewriting. *Journal of Educational Psychology* **28**: 609-620.
- FITTS, P.M. 1954. The information capacity in the human motor system in controlling the amplitude in movement. *Journal of Experimental Psychology* **47**: 381-391.
- FITTS, P.M. 1964. Perceptual-motor skill learning. In Melton, A.W. (Ed.). *Categories of Human Learning*. 243-285. New York: Academic Press.
- FRANKISH, C., HULL, R. AND MORGAN, P. 1995. Recognition accuracy and user acceptance of pen interfaces. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '95)*. ACM Press: 503-510.
- FURNAS, G.W., LANDAUER, T.K., GOMEZ, L.M. AND DUMAIS, S.T. 1987. The vocabulary problem in human-system communication. *Communications of the ACM* **30**(11): 964-971.
- GETSCHOW, C.O, ROSEN, M.J. AND GOODENOUGH-TREPAGNIER, C. 1986. A systematic approach to design a minimum distance alphabetical keyboard. In *Proceedings of the 9<sup>th</sup> Annual Conference of the Rehabilitation Engineering Society of North America (RESNA '86)*. Rehabilitation Society of North America: 396-398.
- GOLDBERG, D. 1997. Unistrokes for computerized interpretation of handwriting. United States Patent 5,596,656. Continuation of United States Patent Application 08/132,401.
- GOLDBERG, D. AND RICHARDSON, D. 1993. Touch-typing with a stylus. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (INTERCHI '93)*. ACM Press: 80-87.
- GOLDSTEIN, M., BAEZ, O. AND DANIELSSON, P. 2000. Employing electrical field sensing for detecting static thumb position using the finger-joint gesture keypad input paradigm. In *Proceedings of the 4<sup>th</sup> IEEE International Symposium on Wearable Computers (ISWC '00)*. IEEE Press: 173-174.
- GOLDSTEIN, M., BOOK, R., ALSIÖ, G. AND TESSA, S. 1999. Non-keyboard QWERTY touch-typing: a portable input interface for the mobile user. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. ACM Press: 32-39.
- GONG, J., HAGGERTY, B. AND TARASEWICH, P. 2005. An enhanced multitap text entry method with predictive next-letter highlighting. In *Extended Abstracts of the ACM*

*Conference on Human Factors in Computing Systems (CHI '05)*. ACM Press: 1399-1402.

GONG, J. AND TARASEWICH, P. 2005. Alphabetically constrained keypad designs for text entry on mobile devices. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. ACM Press: 211-220.

GOODMAN, J., VENOLIA, G., STEURY, K. AND PARKER, C. 2002. Language modeling for soft keyboards. In *Proceedings of the 17<sup>th</sup> National Conference on Artificial Intelligence (AAAI '02)*. AAAI Press: 419-424.

GOPHER, D. AND RAIJ, D. 1988. Typing with a two-hand chord keyboard: will the QWERTY become obsolete? *IEEE Transactions on Systems, Man, and Cybernetics* **18**(4): 601-609.

GREEN, N., KRUGER, J., FALDU, C. AND ST. AMANT, R. 2004. A reduced QWERTY keyboard for mobile text entry. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. ACM Press: 1429-1432.

GUIMBRETIERE, F. AND WINOGRAD, T. 2000. FlowMenu: combining command, text and data entry. In *Proceedings of the 13<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM Press: 213-216.

GUNGL, K.P. 1989. Computer interface and touch sensitive screens. In *Proceedings of the IEEE Conference on VLSI and Computer Peripherals (EuroComp '89)*. IEEE Press: 2/98-2/100.

GUTOWITZ, H. 2001. Method and apparatus for improving multi-tap text input. U.S. Patent 6,219,731.

HAMMING, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal* **26**(2): 147-160.

HASHIMOTO, M. AND TOGASI, M. 1995. A virtual oval keyboard and a vector input method for pen-based character input. In *Conference Compendium of the ACM Conference Human Factors in Computing Systems (CHI '95)*. ACM Press: 254-255.

HIROTAKA, N. 2003. Reassessing current cell phone designs: using thumb input effectively. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 938-939.

HWTE, S.M., HIGGINS, C., LEEDHAM, G. AND YANG, M. 2005. Transliteration of online handwritten phonetic Pitman's shorthand with the use of a Bayesian network. In *Proceedings of the 8<sup>th</sup> IEEE International Conference on Document Analysis and Recognition 2005*. IEEE Press: 1090-1094.

- INGMARSSON, M., DINKA, D. AND ZHAI, S. 2004. TNT: a numeric keypad based text input method. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. ACM Press: 639-646.
- ISOKOSKI, P. 2004a. Manual text input: experiments, models, and systems. Doctoral dissertation, University of Tampere, Finland.
- ISOKOSKI, P. 2004b. Performance of menu-augmented soft keyboards. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. ACM Press: 423-430.
- ISOKOSKI, P. AND LINDEN, T. 2004. Effect of foreign language on text transcription performance: Finns writing English. In *Proceedings of the 4<sup>th</sup> Nordic Conference on Human-Computer Interaction (NordiCHI '04)*. ACM Press: 109-112.
- ISOKOSKI, P. AND MACKENZIE, I.S. 2003. Combined model for text entry rate development. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 752-753.
- ISOKOSKI, P. AND RAISAMO, R. 2004. Quikwriting as a multi-device text entry method. In *Proceedings of the 3<sup>rd</sup> Nordic Conference on Human-Computer Interaction (NordiCHI '04)*. ACM Press: 105-108.
- JAMES, C.L. AND REISCHEL, K.M. 2001. Text input for mobile devices: comparing model prediction to actual performance. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '01)*. ACM Press: 365-371.
- JANSSON, T. 2004. *Latin – kulturen, historien, språket*. Stockholm: Wahlström & Widstrand.
- JAGACINSKI, R.J. AND FLACH, J.M. 2003. *Control Theory for Humans: Quantitative Approaches to Modeling Performance*. Mahwah: Lawrence Erlbaum.
- KANO, A. 2005. Evaluating phrase sets for use with text entry evaluation with dyslexic participants. In *Workshop on Improving and Assessing Pen-Based Input Techniques at the British Computer Society Conference on Human Computer Interaction (BCS HCI '05)*. Unpublished.
- KARAT, C.M., HALVERSON, C., HORN, D. AND KARAT, J. 1999. Patterns of entry and correction in large vocabulary speech recognition systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. ACM Press: 568-575.
- KLIMT, B. AND YANG, Y. 2004. Introducing the Enron corpus. In *Proceedings of 1<sup>st</sup> Conference on Email and Anti-Spam*. Unpublished.

- KRISTENSSON, P.O. 2002. *Design and Evaluation of a Shorthand-Aided Soft Keyboard*. D-uppsats, Linköping University, Sweden.
- KRISTENSSON, P.O. 2004. *Large Vocabulary Shorthand Writing on Stylus Keyboard*. Licentiatavhandling, Linköping University, Sweden.
- KRISTENSSON, P.O. 2005. Breaking the laws of action in the user interface. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. ACM Press: 1120-1121.
- KRISTENSSON, P.O. AND ZHAI, S. 2004. SHARK<sup>2</sup>: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM Press: 43-52.
- KRISTENSSON, P.O. AND ZHAI, S. 2005. Relaxing stylus typing precision by geometric pattern matching. In *Proceedings of the 10<sup>th</sup> ACM International Conference on Intelligent User Interfaces (IUI '05)*. ACM Press: 151-158.
- KRISTENSSON, P.O. AND ZHAI, S. 2007a. Command strokes with and without preview: using pen gestures on keyboard for command selection. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '07)*. ACM Press: 1137-1146.
- KRISTENSSON, P.O. AND ZHAI, S. 2007b. Learning shape writing by game playing. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '07)*. ACM Press: 1971-1976.
- KOBAYASHI, M. AND IGARASHI, T. 2003. Considering the direction of cursor movement for efficient traversal of cascading menus. In *Proceedings of the 16<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM Press: 91-94.
- KOERICH, A.L., SABOURIN, R. AND SUEN, C.Y. 2003. Large vocabulary off-line handwriting recognition: a survey. *Pattern Analysis and Applications* **6**(2): 97-121.
- KOHL, R. AND SHEA, C.H. 1992. Pew (1966) revisited: Acquisition of hierarchical control as a function of observational practice. *Journal of Motor Behavior* **24**(3): 247-260.
- KOZBELT, A. 2001. Artists as experts in visual cognition. *Visual Cognition* **8**(6): 705-723.
- KUESTER, F., CHEN, M., PHAIR, M.E. AND MEHRING, C. 2005. Towards keyboard independent touch typing in VR. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '05)*. ACM Press: 86-95.



- KURTENBACH, G., FITZMAURICE, G., OWEN, R.N. AND BAUDEL, T. 1999. The Hotbox: efficient access to a large number of menu-items. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. ACM Press: 231-237.
- KURTENBACH, G., SELLEN, A.J. AND BUXTON, W.A.S. 1993. Some articulatory and cognitive aspects of “marking menus”: an empirical study. *Human-Computer Interaction* **8**(1): 1-23.
- KÖLTRINGER, T. AND GRECHENIG, T. 2004. Comparing the immediate usability of Graffiti 2 and virtual keyboard. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. ACM Press: 1175-1178.
- LALOMIA, M.J. 1994. User acceptance of handwritten recognition accuracy. In *Conference Compendium of the ACM Conference on Human Factors in Computing Systems (CHI '94)*. ACM Press: 107.
- LANDAUER, T.K. AND BJORK, R.A. 1978. Optimum rehearsal patterns and name learning. In Gruneberg, M.M., Morris, P.M. and Sykes, R.N. (Eds.), *Practical Aspects of Memory*, 625-632. London: Academic Press.
- LEVENSHTAIN, V.I. 1965. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR* **163**(4): 845-848.
- LEVY, D. 2002. The Fastap keypad and pervasive computing. In *Proceedings of Pervasive 2002, Lecture Notes in Computer Science* **2414**. Springer-Verlag: 58-68.
- LEWIS, J.R., KENNEDY, P.J. AND LALOMIA, M.J. 1992. Improved typing-key layouts for single finger or stylus input. IBM Technical Report 54.692.
- LEWIS, J.R., POTOSNAK, K.M. AND MAGYAR, R.L. 1997. Keys and keyboards. In Helander, M.G, Landauer, T.K. and Prabhu, P.V. (Eds.), *Handbook of Human-Computer Interaction*, 2<sup>nd</sup> ed., 1285-1315. Amsterdam: Elsevier.
- LIEBOWITZ, S.J. AND MARGOLIS, S. E. 1990. The fable of the keys. *Journal of Law & Economics* **33**(1): 1-25.
- LIPSCOMB, J.S. 1991. A trainable gesture recognizer. *Pattern Recognition* **24**(9): 895-907.
- LIU, D.K. AND WU, L.C. 2006. Keypad. United States Patent 7,119,296.
- LONG, A.C., LANDAY, J.A. AND ROWE, L.A. 1999. Implications for a gesture design tool. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. ACM Press: 40-47.

- LONG, A.C., LANDAY, J.A., ROWE, L.A. AND MICHIELS, J. 2000. Visual similarity of pen gestures. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '00)*. ACM Press: 360-367.
- LYONS, K., STARNER, T. AND GANE, B. 2006. Experimental evaluations of the twiddler one-handed chording mobile keyboard. *Human-Computer Interaction* **21**: 343-392.
- MACKENZIE, I.S. 1991. *Fitts' Law as a Performance Model in Human-Computer Interaction*. Doctoral dissertation, University of Toronto, Canada.
- MACKENZIE, I.S. 2002. Mobile text entry using three keys. In *Proceedings of the 2<sup>nd</sup> Nordic Conference on Human-Computer Interaction (NordiCHI '02)*. ACM Press: 27-34.
- MACKENZIE, I.S., CHEN J. AND ONISZCZAK, A. 2006. Unipad: single-stroke text entry with language-based acceleration. In *Proceedings of the 4<sup>th</sup> Nordic Conference on Human-Computer Interaction (NordiCHI '06)*. ACM Press: 78-85.
- MACKENZIE, I.S., KOBER, H., SMITH, D., JONES, T. AND SKEPNER, E. 2001. LetterWise: prefix-based disambiguation for mobile text input. In *Proceedings of the 14<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '01)*. ACM Press: 111-120.
- MACKENZIE, I.S. AND SOUKOREFF, R.W. 2002a. Text entry for mobile computing: models and methods, theory and practice. *Human-Computer Interaction* **17**: 147-198.
- MACKENZIE, I.S. AND SOUKOREFF, R.W. 2002b. A model of two-thumb text entry. In *Proceedings of Graphics Interface (GI '02)*. Canadian Information Processing Society: 117-124.
- MACKENZIE, I.S. AND SOUKOREFF, R.W. 2003. Phrase sets for evaluating text entry techniques. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 754-755.
- MACKENZIE, I.S. AND ZHANG, S. 1997. The immediate usability of graffiti. *Proc. Graphics Interface (GI '97)*. Canadian Computer Society: 129-137.
- MACKENZIE, I.S. AND ZHANG, S.X. 1999. The design and evaluation of a high-performance Soft Keyboard. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. ACM Press: 25-31.
- MACNEILL, D. 1998. Why did Apple kill the Newton? *Pen Computing*, June.
- MAGNIEN, L., BOURAOUI AND VIGOUROUX, N. 2004. Mobile text input with soft keyboards: optimization by means of visual clues. In *Proceedings of the 6<sup>th</sup> International Symposium on Mobile Human-Computer Interaction (MobileHCI '04)*, *Lecture Notes in Computer Science* **3160**. Springer-Verlag: 337-341.

- MANKOFF, J. AND ABOWD, G.D. 1998. Cirrin: a word-level unistroke keyboard for pen input. In *Proceedings of the 11<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '98)*. ACM Press: 213-214.
- MANKOFF, J., HUDSON, S.E. AND ABOWD, G.D. 2002. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Proceedings of the 15<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '02)*. ACM Press: 11-20.
- MARSHALL, D., FOSTER, J.C. AND JACK, M.A. 2001. User performance and attitude towards schemes for alphanumeric data entry using restricted input devices. *Behavior and Information Technology* **20**(3): 167-188.
- MARTIN, B. 2005. VirHKey: a VIRtual Hyperbolic KEYboard with gesture interaction and visual feedback for mobile devices. In *Proceedings of the 7<sup>th</sup> International Conference on Human-Computer Interaction with Mobile Services & Devices (MobileHCI '05)*. ACM Press: 99-106.
- MARTINVILLE, S. 1849. *Historie de la stenographie*. Paris.
- MARZAL, A. AND VIDAL, E. 1993. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15**(9): 926-932.
- MASUI, T. 1998. An efficient text input method for pen-based computers. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '98)*. ACM Press: 328-335.
- MATIAS, E., MACKENZIE, I.S. AND BUXTON, W. 1996. One-handed touch-typing on a QWERTY keyboard. *Human-Computer Interaction* **11**: 1-27.
- MEHRING, C. 2005. System and method for keyboard independent touch typing. U.S. Patent 6,885,316. Continuation of U.S. Patent 6,670,894.
- MELIN, O.W. 1927. *Stenografiens historia, första delen*. Stockholm: P.A. Norstedt & Söner.
- MELIN, O.W. 1929. *Stenografiens historia, andra delen*. Stockholm: P.A. Norstedt & Söner.
- MONTGOMERY, E.B. 1982. Bringing manual input into the 20<sup>th</sup> century: new keyboard concepts. *IEEE Computer* **15**(3): 11-18.
- MOORE, R.K. 2004. Modelling data entry rates for ASR and alternative input methods. In *Proc. 8<sup>th</sup> International Conference on Spoken Language Processing (INTERSPEECH '04)*. ISCA Archive: 2285-2288.

MORGAN, H.L. 1970. Spelling correction in systems programs. *Communication of the ACM* **13**(2): 90-94.

NAFIZ, A. AND YARMAN-VURAL, F.T. 2001. An overview of character recognition focused on off-line handwriting. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews* **31**(2): 216-233.

NESBAT, S. 2003. A system for fast, full-text entry for small electronic devices. In *Proceedings of the International Conference on Multimodal Interaction (ICMI '03)*. ACM Press: 4-11.

NEWMAN, M.W., LIN, J., HONG, J.I. AND LANDAY, J.A. 2003. DENIM: an informal web site design tool inspired by observations of practice. *Human-Computer Interaction* **18**(3): 259-324.

NEWMAN, W.M. AND SPROULL, R.F. 1979. *Principles of Interactive Computer Graphics*. New York: McGraw-Hill.

NIBLACK, W. AND YIN, J. 1995. A pseudo-distance measure for 2D shapes based on turning angle. In *Proceedings of the International Conference on Image Processing (ICIP '95)* **3**. IEEE: 352-355.

NORMAN, D.A. AND FISHER, D. 1982. Why alphabetic keyboards are not easy to use: keyboard layout doesn't much matter. *Human Factors* **24**(5): 509-519.

NOYES, J. 1983. The QWERTY keyboard: a review. *International Journal of Man-Machine Studies* **18**(3): 265-281.

ORR, L. 1987. The blind spot of history: logography. *Yale French Studies* **73**: 190-214.

PADMANABHAN, M. AND PICHENY, M. 2002. Large vocabulary speech-recognition algorithms. *IEEE Computer* **35**(3): 42-50.

PARTRIDGE, K, CHATTERJEE, S., SAZAWAL, V., BORRIELLO, G. AND WANT, R. 2002. TiltType: accelerometer-supported text entry for very small devices. In *Proceedings of the 15<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '02)*. ACM Press: 201-204.

PAVLOVYCH, A. AND STUERZLINGER, W. 2003. Less-tap: a fast and easy-to-learn text input technique for phones. In *Proceedings of Graphics Interface (GI '03)*. Canadian Information Processing Society: 97-104.

PAVLOVYCH, A. AND STUERZLINGER, W. 2004. Model for non-expert text entry speed on 12-button phone keypads. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. ACM Press: 351-358.

- PERLIN, K. 1998. Quikwriting: continuous stylus-based text entry. In *Proceedings of the 11<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '98)*. ACM Press: 215-216.
- PLAMONDON, R. AND SRIHARI, S.N. 2000. On-line and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(1): 63-84.
- POTTER, R.L., WELDON, L.J. AND SHNEIDERMAN, B. 1988. Improving the accuracy of touch screens: an experimental evaluation of three strategies. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '88)*. ACM Press: 27-32.
- POULTON, E.C. 1966. Unwanted asymmetrical transfer effects with balanced experimental designs. *Psychological Bulletin* **66**(1): 1-8.
- PRICE, K. AND SEARS, A. 2005. Speech-based text entry for mobile handheld devices: an analysis of efficacy and error correction techniques for server-based solutions. *International Journal of Human-Computer Interaction* **19**(3): 279-304.
- PRICE, K.J., LIN, M., FENG, J., GOLDMAN, R., SEARS, A. AND JACKO, J.A. 2006. Motion does matter: an examination of speech-based text entry on the move. *Universal Access in the Information Society* **4**(3): 246-257.
- PROSCHOWSKY, M., SCHULTZ, N. AND JACOBSEN, N.E. 2006. An intuitive text input method for touch wheels. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. ACM Press: 467-470.
- QUIGLEY, M., GOODRICH, M.A. AND BEARD, R.W. 2004. Semi-autonomous human-UAV interfaces for fixed-wing mini-UAVs. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '04)*. IEEE Press: 2457-2462.
- RABINER, L. AND JUANG, B.-H. 1993. *Fundamentals of Speech Recognition*. New Jersey: Prentice Hall.
- RASMUSSEN, J. 1983. Skills, rules and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics* **13**(3): 257-266.
- ROEBER, H., BACUS, J. AND TOMASI, C. 2003. Typing in thin air: the Canesta keyboard – a new method of interaction with electronic devices. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 712-713.
- ROSENBAUM, D.A. 1991. *Human Motor Control*. San Diego: Academic Press.

ROSENBERG, R. AND SLATER, M. 1999. The chording glove: a glove-based text input device. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews* **29**(2): 186-191.

RUBINE, D. 1991. Specifying gestures by example. In *Proceedings of the ACM International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. ACM Press: 329-337.

RUDNICKY, A.I., HAUPTMANN, A.G. AND LEE, K.-F. 1994. Survey of current speech technology. *Communications of the ACM* **37**(3): 52-57.

SACKS, D. 2003. *The Alphabet*. London: Hutchinson.

SAZAWAL, V., WANT, R. AND BORRIELLO, G. 2002. The unigesture approach: one-handed text entry for small devices. In *Proceedings of the 4<sup>th</sup> International Symposium on Mobile Human-Computer Interaction (MobileHCI '02), Lecture Notes in Computer Science* **2411**. Springer-Verlag: 256-270.

SEARS, A. AND ARORA, R. 2002. Data entry for mobile devices: an empirical comparison of novice performance with Jot and Graffiti. *Interacting with Computers* **14**(5): 413-433.

SHADMEHR, R. AND HOLCOMB, H.H. 1997. Neural correlates of motor memory consolidation. *Science* **277**(5327): 821-825.

SHADMEHR, R. AND WISE. 2005. *The computational neurobiology of reaching and pointing: a foundation for motor learning*. Cambridge: MIT Press.

SHANNON, C.E. 1948. A mathematical theory of communication. *Bell System Technical Journal* **27**: 379-423, 623-656.

SHIEBER, S.M. AND BAKER, E. 2003. Abbreviated text input. In *Proceedings of the 8<sup>th</sup> ACM International Conference on Intelligent User Interfaces (IUI '03)*. ACM Press: 293-296.

SHIEBER, S.M. AND NELKEN, R. 2007. Abbreviated text input using language modeling. *Natural Language Engineering* **13**(2): 165-183.

SHNEIDERMAN, B. 2000. The limits of speech recognition. *Communications of the ACM* **43**(9): 63-65.

SILFVERBERG, M., MACKENZIE, I.S. AND KORHONEN, P. 2000. Predicting text entry speed on mobile phones. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '00)*. ACM Press: 9-16.

SIMPSON, J. AND WEINER, E.C. (Eds.). 1989. *Oxford English Dictionary*. Oxford: Clarendon Press.

SMITH, S.L. AND GOODWIN, N.C. 1971. Alphabetic data entry via the touch-tone pad: a comment. *Human Factors* **13**(2): 189-190.

SMITH, B.A. AND ZHAI, S. 2001. Optimised virtual keyboards with and without alphabetical ordering – a novice user study. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT '01)*. IOS Press: 92-99.

SOUKOREFF, R.W. AND MACKENZIE, I.S. 2003a. Input-based language modelling in the design of high performance text input techniques. In *Proceedings of Graphics Interface (GI '03)*. Canadian Information Processing Society: 89-96.

SOUKOREFF, R.W. AND MACKENZIE, I.S. 2003b. Metrics for text entry research: an evaluation of MSD and KSPC, and a new unified error metric. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 113-120.

STUBBS, K. 2003. Kana no senshi (kana warrior): a new interface for learning Japanese characters. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 894-895.

TANAKA-ISHII, K. 2007. Word-based predictive text entry using adaptive language models. *Natural Language Engineering* **13**(1): 51-74.

TANAKA-ISHII, K., INUTSUKA, Y. AND TAKEICHI, M. 2002. Entering text with a four-button device. In *Proceedings of the 19<sup>th</sup> International Conference on Computational Linguistics (COLING '02)*. ACL: 1-7.

TAPPERT, C.C. 1982. Cursive script recognition by elastic matching. *IBM Journal of Research and Development* **26**(6): 765-771.

TAPPERT, C.C., SUEN, C.Y. AND WAKAHARA, T. 1990. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(8): 787-808.

TEITELMAN, W. 1964. Real time recognition of hand-drawn characters. In *Proceedings of the Fall Joint Computer Conference (FJCC '62)*. American Federation of Information Processing Societies: 559-575.

THE UNICODE CONSORTIUM. 2003. The Unicode standard 4.0. Boston: Addison Wesley.

THEODORIDIS, S. AND KONSTANTINOS, K. 1999. *Pattern Recognition*. London: Academic Press.

TOMASI, C., RAFIL, A. AND TORUNOGLU, I. 2003. Full-size projection keyboard for handheld devices. *Communications of the ACM* **46**(7): 70-75.

- VANDERHEIDEN, G.C. AND KELSO, D.P. 1987. Comparative analysis of fixed-vocabulary communication acceleration techniques. *Augmentative and Alternative Communication* **3**(4): 196-206.
- VENOLIA, D. AND NEIBERG, F. 1994. T-Cube: A fast self-disclosing pen-based alphabet. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94)*. ACM Press: 265-270.
- WAGNER, R.A. AND FISCHER, M.J. 1974. The string-to-string correction problem. *Journal of the Association of Computing Machinery* **21**(1): 168-173.
- WANG, Y.P. AND PAVLIDIS, T. 1990. Optimal correspondence of string subsequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(11): 1080-1087.
- WANG, J., ZHAI, S. AND SU, H. 2001. Chinese input with keyboard and eye-tracking – an anatomical study. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '01)*. ACM Press: 349-356.
- WARD, D.J., BLACKWELL, A.F. AND MACKAY, D.J.C. 2002. Dasher: a gesture-driven data entry interface for mobile computing. *Human-Computer Interaction* **17**: 199-228.
- WIGDOR, D. AND BALAKRISHNAN, R. 2003. TiltText: using tilt for text input to mobile phones. In *Proceedings of the 16<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM Press: 81-90.
- WIGDOR, D. AND BALAKRISHNAN, R. 2004. A comparison of consecutive and concurrent input text entry techniques for mobile phones. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. ACM Press: 81-88.
- WILLINGHAM, D.B. 1998. A neuropsychological theory of motor skill learning. *Psychological Review* **105**(3): 558-584.
- WOBBROCK, J.O. 2006. EdgeWrite: A versatile design for text entry and control. Doctoral dissertation, Carnegie-Mellon University, United States of America.
- WOBBROCK, J.O., MYERS, B.A. AND ROTHROCK, B. 2006a. Few-key text entry revisited: mnemonic gestures on four keys. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. ACM Press: 489-492.
- WOBBROCK, J.O., CHAU, D.H. AND MYERS, B.A. 2007. An alternative to push, press and tap-tap-tap: gesturing on an isometric joystick for mobile phone text entry. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '07)*. ACM Press: 667-676.
- WOBBROCK, J.O., MYERS, B.A. AND CHAU, D.H. 2006b. In-stroke word completion. In *Proceedings of the 19<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '06)*. ACM Press: 333-336.



WOBROCK, J.O, MYERS, B.A. AND KEMBEL, J.A. 2003. EdgeWrite: a stylus-based text entry method designed for high-accuracy and stability of motion. In *Proceedings of the 16<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '03)*. ACM Press: 61-70.

WOLF, C. AND MORREL-SAMUELS, P. 1987. The use of hand-drawn gestures for text editing. *International Journal of Man-Machine Studies* **27**(1): 91-102.

XU, L., KRZYZAK, A. AND SUEN, C.Y. 1992. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics* **22**(3): 418-435.

XUE, H. AND GOVINDARAJU, V. 2002. On the dependence of handwritten word recognizers on lexicons. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(12): 1553-1564.

YAMADA, H. 1980. A historical study of typewriters and typing methods: from the position of planning Japanese parallels. *Journal of Information Processing* **4**(2): 175-202.

ZHAI, S., HUNTER, M. AND SMITH, B.A. 2000. The Metropolis keyboard – an exploration of quantitative techniques for virtual keyboard design. In *Proceedings of the 13<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM Press: 119-128.

ZHAI, S., KONG, J. AND REN, X. 2004. Speed-accuracy tradeoff in Fitts' law tasks – on the equivalency of actual and nominal pointing precision. *International Journal of Human-Computer Studies* **61**(6): 823-856.

ZHAI, S. AND KRISTENSSON, P.O. 2003. Shorthand writing on stylus keyboard. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. ACM Press: 97-104.

ZHAI, S. AND KRISTENSSON, P.O. 2007. Introduction to shape writing. In MacKenzie, I.S. and Tanaka-Ishii, K. (Eds.), *Text Entry Systems*, 139-158. San Francisco: Morgan Kaufman.

ZHAI, S., KRISTENSSON, P.O. AND SMITH, B.A. 2005. In search of effective text input interfaces for off the desktop computing, *Interacting with Computers* **17**(3): 229-250.

ZHAI, S., SMITH, B.A. AND HUNTER, M. 2002. Performance optimization of virtual keyboards. *Human-Computer Interaction* **17**(2&3): 89-129.

ZHAI, S., SUE, A. AND ACCOT, J. 2002. Movement model, hits distribution and learning in virtual keyboarding. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '02)*. ACM Press: 17-24.

ZHAO, S. AND BALAKRISHNAN, R. 2004. Simple vs. compound marking mark hierarchical marking menus. In *Proceedings of the 17<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology (UIST '04)*. ACM Press: 33-42.

ZIPF, G.K. 1935. *Human Behavior and the Principle of Least-Effort*. Cambridge: Addison-Wesley.

Dissertations

Linköping Studies in Science and Technology

- No 14 **Anders Haraldsson:** A Program Manipulation System Based on Partial Evaluation, 1977, ISBN 91-7372-144-1.
- No 17 **Bengt Magnhagen:** Probability Based Verification of Time Margins in Digital Designs, 1977, ISBN 91-7372-157-3.
- No 18 **Mats Cedwall:** Semantisk analys av processbeskrivningar i naturligt språk, 1977, ISBN 91-7372-168-9.
- No 22 **Jaak Urmi:** A Machine Independent LISP Compiler and its Implications for Ideal Hardware, 1978, ISBN 91-7372-188-3.
- No 33 **Tore Risch:** Compilation of Multiple File Queries in a Meta-Database System 1978, ISBN 91-7372-232-4.
- No 51 **Erland Jungert:** Synthesizing Database Structures from a User Oriented Data Model, 1980, ISBN 91-7372-387-8.
- No 54 **Sture Hägglund:** Contributions to the Development of Methods and Tools for Interactive Design of Applications Software, 1980, ISBN 91-7372-404-1.
- No 55 **Pär Emanuelson:** Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation, 1980, ISBN 91-7372-403-3.
- No 58 **Bengt Johnsson, Bertil Andersson:** The Human-Computer Interface in Commercial Systems, 1981, ISBN 91-7372-414-9.
- No 69 **H. Jan Komorowski:** A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation, 1981, ISBN 91-7372-479-3.
- No 71 **René Reboh:** Knowledge Engineering Techniques and Tools for Expert Systems, 1981, ISBN 91-7372-489-0.
- No 77 **Östen Oskarsson:** Mechanisms of Modifiability in large Software Systems, 1982, ISBN 91-7372-527-7.
- No 94 **Hans Lunell:** Code Generator Writing Systems, 1983, ISBN 91-7372-652-4.
- No 97 **Andrzej Lingas:** Advances in Minimum Weight Triangulation, 1983, ISBN 91-7372-660-5.
- No 109 **Peter Fritzson:** Towards a Distributed Programming Environment based on Incremental Compilation, 1984, ISBN 91-7372-801-2.
- No 111 **Erik Tengvald:** The Design of Expert Planning Systems. An Experimental Operations Planning System for Turning, 1984, ISBN 91-7372-805-5.
- No 155 **Christos Levcopoulos:** Heuristics for Minimum Decompositions of Polygons, 1987, ISBN 91-7870-133-3.
- No 165 **James W. Goodwin:** A Theory and System for Non-Monotonic Reasoning, 1987, ISBN 91-7870-183-X.
- No 170 **Zebo Peng:** A Formal Methodology for Automated Synthesis of VLSI Systems, 1987, ISBN 91-7870-225-9.
- No 174 **Johan Fagerström:** A Paradigm and System for Design of Distributed Systems, 1988, ISBN 91-7870-301-8.
- No 192 **Dimiter Driankov:** Towards a Many Valued Logic of Quantified Belief, 1988, ISBN 91-7870-374-3.
- No 213 **Lin Padgham:** Non-Monotonic Inheritance for an Object Oriented Knowledge Base, 1989, ISBN 91-7870-485-5.
- No 214 **Tony Larsson:** A Formal Hardware Description and Verification Method, 1989, ISBN 91-7870-517-7.
- No 221 **Michael Reinfrank:** Fundamentals and Logical Foundations of Truth Maintenance, 1989, ISBN 91-7870-546-0.
- No 239 **Jonas Löwgren:** Knowledge-Based Design Support and Discourse Management in User Interface Management Systems, 1991, ISBN 91-7870-720-X.
- No 244 **Henrik Eriksson:** Meta-Tool Support for Knowledge Acquisition, 1991, ISBN 91-7870-746-3.
- No 252 **Peter Eklund:** An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies, 1991, ISBN 91-7870-784-6.
- No 258 **Patrick Doherty:** NML3 - A Non-Monotonic Formalism with Explicit Defaults, 1991, ISBN 91-7870-816-8.
- No 260 **Nahid Shahmehri:** Generalized Algorithmic Debugging, 1991, ISBN 91-7870-828-1.
- No 264 **Nils Dahlbäck:** Representation of Discourse-Cognitive and Computational Aspects, 1992, ISBN 91-7870-850-8.
- No 265 **Ulf Nilsson:** Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs, 1992, ISBN 91-7870-858-3.
- No 270 **Ralph Rönquist:** Theory and Practice of Tense-bound Object References, 1992, ISBN 91-7870-873-7.
- No 273 **Björn Fjellborg:** Pipeline Extraction for VLSI Data Path Synthesis, 1992, ISBN 91-7870-880-X.
- No 276 **Staffan Bonnier:** A Formal Basis for Horn Clause Logic with External Polymorphic Functions, 1992, ISBN 91-7870-896-6.
- No 277 **Kristian Sandahl:** Developing Knowledge Management Systems with an Active Expert Methodology, 1992, ISBN 91-7870-897-4.
- No 281 **Christer Bäckström:** Computational Complexity

- of Reasoning about Plans, 1992, ISBN 91-7870-979-2.
- No 292 **Mats Wirén:** Studies in Incremental Natural Language Analysis, 1992, ISBN 91-7871-027-8.
- No 297 **Mariam Kamkar:** Interprocedural Dynamic Slicing with Applications to Debugging and Testing, 1993, ISBN 91-7871-065-0.
- No 302 **Tingting Zhang:** A Study in Diagnosis Using Classification and Defaults, 1993, ISBN 91-7871-078-2.
- No 312 **Arne Jönsson:** Dialogue Management for Natural Language Interfaces - An Empirical Approach, 1993, ISBN 91-7871-110-X.
- No 338 **Simin Nadjm-Tehrani:** Reactive Systems in Physical Environments: Compositional Modelling and Framework for Verification, 1994, ISBN 91-7871-237-8.
- No 371 **Bengt Savén:** Business Models for Decision Support and Learning. A Study of Discrete-Event Manufacturing Simulation at Asea/ABB 1968-1993, 1995, ISBN 91-7871-494-X.
- No 375 **Ulf Söderman:** Conceptual Modelling of Mode Switching Physical Systems, 1995, ISBN 91-7871-516-4.
- No 383 **Andreas Kägedal:** Exploiting Groundness in Logic Programs, 1995, ISBN 91-7871-538-5.
- No 396 **George Fodor:** Ontological Control, Description, Identification and Recovery from Problematic Control Situations, 1995, ISBN 91-7871-603-9.
- No 413 **Mikael Pettersson:** Compiling Natural Semantics, 1995, ISBN 91-7871-641-1.
- No 414 **Xinli Gu:** RT Level Testability Improvement by Testability Analysis and Transformations, 1996, ISBN 91-7871-654-3.
- No 416 **Hua Shu:** Distributed Default Reasoning, 1996, ISBN 91-7871-665-9.
- No 429 **Jaime Villegas:** Simulation Supported Industrial Training from an Organisational Learning Perspective - Development and Evaluation of the SSIT Method, 1996, ISBN 91-7871-700-0.
- No 431 **Peter Jonsson:** Studies in Action Planning: Algorithms and Complexity, 1996, ISBN 91-7871-704-3.
- No 437 **Johan Boye:** Directional Types in Logic Programming, 1996, ISBN 91-7871-725-6.
- No 439 **Cecilia Sjöberg:** Activities, Voices and Arenas: Participatory Design in Practice, 1996, ISBN 91-7871-728-0.
- No 448 **Patrick Lambrix:** Part-Whole Reasoning in Description Logics, 1996, ISBN 91-7871-820-1.
- No 452 **Kjell Orsborn:** On Extensible and Object-Relational Database Technology for Finite Element Analysis Applications, 1996, ISBN 91-7871-827-9.
- No 459 **Olof Johansson:** Development Environments for Complex Product Models, 1996, ISBN 91-7871-855-4.
- No 461 **Lena Strömbäck:** User-Defined Constructions in Unification-Based Formalisms, 1997, ISBN 91-7871-857-0.
- No 462 **Lars Degerstedt:** Tabulation-based Logic Programming: A Multi-Level View of Query Answering, 1996, ISBN 91-7871-858-9.
- No 475 **Fredrik Nilsson:** Strategi och ekonomisk styrning - En studie av hur ekonomiska styrsystem utformas och används efter företagsförvärv, 1997, ISBN 91-7871-914-3.
- No 480 **Mikael Lindvall:** An Empirical Study of Requirements-Driven Impact Analysis in Object-Oriented Software Evolution, 1997, ISBN 91-7871-927-5.
- No 485 **Göran Forslund:** Opinion-Based Systems: The Cooperative Perspective on Knowledge-Based Decision Support, 1997, ISBN 91-7871-938-0.
- No 494 **Martin Sköld:** Active Database Management Systems for Monitoring and Control, 1997, ISBN 91-7219-002-7.
- No 495 **Hans Olsén:** Automatic Verification of Petri Nets in a CLP framework, 1997, ISBN 91-7219-011-6.
- No 498 **Thomas Drakengren:** Algorithms and Complexity for Temporal and Spatial Formalisms, 1997, ISBN 91-7219-019-1.
- No 502 **Jakob Axelsson:** Analysis and Synthesis of Heterogeneous Real-Time Systems, 1997, ISBN 91-7219-035-3.
- No 503 **Johan Ringström:** Compiler Generation for Data-Parallel Programming Languages from Two-Level Semantics Specifications, 1997, ISBN 91-7219-045-0.
- No 512 **Anna Moberg:** Närhet och distans - Studier av kommunikationsmönster i satellitkontor och flexibla kontor, 1997, ISBN 91-7219-119-8.
- No 520 **Mikael Ronström:** Design and Modelling of a Parallel Data Server for Telecom Applications, 1998, ISBN 91-7219-169-4.
- No 522 **Niclas Ohlsson:** Towards Effective Fault Prevention - An Empirical Study in Software Engineering, 1998, ISBN 91-7219-176-7.
- No 526 **Joachim Karlsson:** A Systematic Approach for Prioritizing Software Requirements, 1998, ISBN 91-7219-184-8.
- No 530 **Henrik Nilsson:** Declarative Debugging for Lazy Functional Languages, 1998, ISBN 91-7219-197-x.
- No 555 **Jonas Hallberg:** Timing Issues in High-Level Synthesis, 1998, ISBN 91-7219-369-7.
- No 561 **Ling Lin:** Management of 1-D Sequence Data - From Discrete to Continuous, 1999, ISBN 91-7219-402-2.
- No 563 **Eva L Ragnemalm:** Student Modelling based on Collaborative Dialogue with a Learning Companion, 1999, ISBN 91-7219-412-X.
- No 567 **Jörgen Lindström:** Does Distance matter? On geographical dispersion in organisations, 1999, ISBN 91-7219-439-1.
- No 582 **Vanja Josifovski:** Design, Implementation and

- Evaluation of a Distributed Mediator System for Data Integration, 1999, ISBN 91-7219-482-0.
- No 589 **Rita Kovordányi:** Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999, ISBN 91-7219-506-1.
- No 592 **Mikael Ericsson:** Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999, ISBN 91-7219-532-0.
- No 593 **Lars Karlsson:** Actions, Interactions and Narratives, 1999, ISBN 91-7219-534-7.
- No 594 **C. G. Mikael Johansson:** Social and Organizational Aspects of Requirements Engineering Methods - A practice-oriented approach, 1999, ISBN 91-7219-541-X.
- No 595 **Jörgen Hansson:** Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999, ISBN 91-7219-542-8.
- No 596 **Niklas Hallberg:** Incorporating User Values in the Design of Information Systems and Services in the Public Sector: A Methods Approach, 1999, ISBN 91-7219-543-6.
- No 597 **Vivian Vimarlund:** An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999, ISBN 91-7219-544-4.
- No 598 **Johan Jenvald:** Methods and Tools in Computer-Supported Taskforce Training, 1999, ISBN 91-7219-547-9.
- No 607 **Magnus Merkel:** Understanding and enhancing translation by parallel text processing, 1999, ISBN 91-7219-614-9.
- No 611 **Silvia Coradeschi:** Anchoring symbols to sensory data, 1999, ISBN 91-7219-623-8.
- No 613 **Man Lin:** Analysis and Synthesis of Reactive Systems: A Generic Layered Architecture Perspective, 1999, ISBN 91-7219-630-0.
- No 618 **Jimmy Tjäder:** Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999, ISBN 91-7219-657-2.
- No 627 **Vadim Engelson:** Tools for Design, Interactive Simulation, and Visualization of Object-Oriented Models in Scientific Computing, 2000, ISBN 91-7219-709-9.
- No 637 **Esa Falkenroth:** Database Technology for Control and Simulation, 2000, ISBN 91-7219-766-8.
- No 639 **Per-Arne Persson:** Bringing Power and Knowledge Together: Information Systems Design for Autonomy and Control in Command Work, 2000, ISBN 91-7219-796-X.
- No 660 **Erik Larsson:** An Integrated System-Level Design for Testability Methodology, 2000, ISBN 91-7219-890-7.
- No 688 **Marcus Bjärelund:** Model-based Execution Monitoring, 2001, ISBN 91-7373-016-5.
- No 689 **Joakim Gustafsson:** Extending Temporal Action Logic, 2001, ISBN 91-7373-017-3.
- No 720 **Carl-Johan Petri:** Organizational Information Provision - Managing Mandatory and Discretionary Use of Information Technology, 2001, ISBN-91-7373-126-9.
- No 724 **Paul Scerri:** Designing Agents for Systems with Adjustable Autonomy, 2001, ISBN 91 7373 207 9.
- No 725 **Tim Heyer:** Semantic Inspection of Software Artifacts: From Theory to Practice, 2001, ISBN 91 7373 208 7.
- No 726 **Pär Carlshamre:** A Usability Perspective on Requirements Engineering - From Methodology to Product Development, 2001, ISBN 91 7373 212 5.
- No 732 **Juha Takkinen:** From Information Management to Task Management in Electronic Mail, 2002, ISBN 91 7373 258 3.
- No 745 **Johan Åberg:** Live Help Systems: An Approach to Intelligent Help for Web Information Systems, 2002, ISBN 91-7373-311-3.
- No 746 **Rego Granlund:** Monitoring Distributed Teamwork Training, 2002, ISBN 91-7373-312-1.
- No 757 **Henrik André-Jönsson:** Indexing Strategies for Time Series Data, 2002, ISBN 917373-346-6.
- No 747 **Anneli Hagdahl:** Development of IT-supported Inter-organisational Collaboration - A Case Study in the Swedish Public Sector, 2002, ISBN 91-7373-314-8.
- No 749 **Sofie Pilemalm:** Information Technology for Non-Profit Organisations - Extended Participatory Design of an Information System for Trade Union Shop Stewards, 2002, ISBN 91-7373-318-0.
- No 765 **Stefan Holmlid:** Adapting users: Towards a theory of use quality, 2002, ISBN 91-7373-397-0.
- No 771 **Magnus Morin:** Multimedia Representations of Distributed Tactical Operations, 2002, ISBN 91-7373-421-7.
- No 772 **Pawel Pietrzak:** A Type-Based Framework for Locating Errors in Constraint Logic Programs, 2002, ISBN 91-7373-422-5.
- No 758 **Erik Berglund:** Library Communication Among Programmers Worldwide, 2002, ISBN 91-7373-349-0.
- No 774 **Choong-ho Yi:** Modelling Object-Oriented Dynamic Systems Using a Logic-Based Framework, 2002, ISBN 91-7373-424-1.
- No 779 **Mathias Broxvall:** A Study in the Computational Complexity of Temporal Reasoning, 2002, ISBN 91-7373-440-3.
- No 793 **Asmus Pandikow:** A Generic Principle for Enabling Interoperability of Structured and Object-Oriented Analysis and Design Tools, 2002, ISBN 91-7373-479-9.
- No 785 **Lars Hult:** Publika Informationstjänster. En studie av den Internetbaserade encyklopedins bruksegenskaper, 2003, ISBN 91-7373-461-6.
- No 800 **Lars Taxén:** A Framework for the Coordination of Complex Systems' Development, 2003, ISBN 91-7373-604-X.
- No 808 **Klas Gäre:** Tre perspektiv på förväntningar och förändringar i samband med införande av informa-

- tionsystem, 2003, ISBN 91-7373-618-X.
- No 821 **Mikael Kindborg:** Concurrent Comics - programming of social agents by children, 2003, ISBN 91-7373-651-1.
- No 823 **Christina Ölvingson:** On Development of Information Systems with GIS Functionality in Public Health Informatics: A Requirements Engineering Approach, 2003, ISBN 91-7373-656-2.
- No 828 **Tobias Ritzau:** Memory Efficient Hard Real-Time Garbage Collection, 2003, ISBN 91-7373-666-X.
- No 833 **Paul Pop:** Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems, 2003, ISBN 91-7373-683-X.
- No 852 **Johan Moe:** Observing the Dynamic Behaviour of Large Distributed Systems to Improve Development and Testing - An Empirical Study in Software Engineering, 2003, ISBN 91-7373-779-8.
- No 867 **Erik Herzog:** An Approach to Systems Engineering Tool Data Representation and Exchange, 2004, ISBN 91-7373-929-4.
- No 872 **Aseel Berglund:** Augmenting the Remote Control: Studies in Complex Information Navigation for Digital TV, 2004, ISBN 91-7373-940-5.
- No 869 **Jo Skåmedal:** Telecommuting's Implications on Travel and Travel Patterns, 2004, ISBN 91-7373-935-9.
- No 870 **Linda Askenäs:** The Roles of IT - Studies of Organising when Implementing and Using Enterprise Systems, 2004, ISBN 91-7373-936-7.
- No 874 **Annika Flycht-Eriksson:** Design and Use of Ontologies in Information-Providing Dialogue Systems, 2004, ISBN 91-7373-947-2.
- No 873 **Peter Bunus:** Debugging Techniques for Equation-Based Languages, 2004, ISBN 91-7373-941-3.
- No 876 **Jonas Mellin:** Resource-Predictable and Efficient Monitoring of Events, 2004, ISBN 91-7373-956-1.
- No 883 **Magnus Bång:** Computing at the Speed of Paper: Ubiquitous Computing Environments for Healthcare Professionals, 2004, ISBN 91-7373-971-5.
- No 882 **Robert Eklund:** Disfluency in Swedish human-human and human-machine travel booking dialogues, 2004, ISBN 91-7373-966-9.
- No 887 **Anders Lindström:** English and other Foreign Linguistic Elements in Spoken Swedish. Studies of Productive Processes and their Modelling using Finite-State Tools, 2004, ISBN 91-7373-981-2.
- No 889 **Zhiping Wang:** Capacity-Constrained Production-inventory systems - Modelling and Analysis in both a traditional and an e-business context, 2004, ISBN 91-85295-08-6.
- No 893 **Pernilla Qvarfordt:** Eyes on Multimodal Interaction, 2004, ISBN 91-85295-30-2.
- No 910 **Magnus Kald:** In the Borderland between Strategy and Management Control - Theoretical Framework and Empirical Evidence, 2004, ISBN 91-85295-82-5.
- No 918 **Jonas Lundberg:** Shaping Electronic News: Genre Perspectives on Interaction Design, 2004, ISBN 91-85297-14-3.
- No 900 **Mattias Arvola:** Shades of use: The dynamics of interaction design for sociable use, 2004, ISBN 91-85295-42-6.
- No 920 **Luis Alejandro Cortés:** Verification and Scheduling Techniques for Real-Time Embedded Systems, 2004, ISBN 91-85297-21-6.
- No 929 **Diana Szentivanyi:** Performance Studies of Fault-Tolerant Middleware, 2005, ISBN 91-85297-58-5.
- No 933 **Mikael Cäker:** Management Accounting as Constructing and Opposing Customer Focus: Three Case Studies on Management Accounting and Customer Relations, 2005, ISBN 91-85297-64-X.
- No 937 **Jonas Kvarnström:** TALplanner and Other Extensions to Temporal Action Logic, 2005, ISBN 91-85297-75-5.
- No 938 **Bourhane Kadmiry:** Fuzzy Gain-Scheduled Visual Servoing for Unmanned Helicopter, 2005, ISBN 91-85297-76-3.
- No 945 **Gert Jervan:** Hybrid Built-In Self-Test and Test Generation Techniques for Digital Systems, 2005, ISBN: 91-85297-97-6.
- No 946 **Anders Arpteg:** Intelligent Semi-Structured Information Extraction, 2005, ISBN 91-85297-98-4.
- No 947 **Ola Angelsmark:** Constructing Algorithms for Constraint Satisfaction and Related Problems - Methods and Applications, 2005, ISBN 91-85297-99-2.
- No 963 **Calin Curescu:** Utility-based Optimisation of Resource Allocation for Wireless Networks, 2005, ISBN 91-85457-07-8.
- No 972 **Björn Johansson:** Joint Control in Dynamic Situations, 2005, ISBN 91-85457-31-0.
- No 974 **Dan Lawesson:** An Approach to Diagnosability Analysis for Interacting Finite State Systems, 2005, ISBN 91-85457-39-6.
- No 979 **Claudiu Duma:** Security and Trust Mechanisms for Groups in Distributed Services, 2005, ISBN 91-85457-54-X.
- No 983 **Sorin Manolache:** Analysis and Optimisation of Real-Time Systems with Stochastic Behaviour, 2005, ISBN 91-85457-60-4.
- No 986 **Yuxiao Zhao:** Standards-Based Application Integration for Business-to-Business Communications, 2005, ISBN 91-85457-66-3.
- No 1004 **Patrik Haslum:** Admissible Heuristics for Automated Planning, 2006, ISBN 91-85497-28-2.
- No 1005 **Aleksandra Tešanovic:** Developing Reusable and Reconfigurable Real-Time Software using Aspects and Components, 2006, ISBN 91-85497-29-0.
- No 1008 **David Dinka:** Role, Identity and Work: Extending the design and development agenda, 2006, ISBN 91-85497-42-8.
- No 1009 **Iakov Nakhimovski:** Contributions to the Modeling and Simulation of Mechanical Systems with Detailed Contact Analysis, 2006, ISBN 91-85497-43-X.
- No 1013 **Wilhelm Dahllöf:** Exact Algorithms for Exact Satisfiability Problems, 2006, ISBN 91-85523-97-6.
- No 1016 **Levon Saldamli:** PDEModelica - A High-Level Language for Modeling with Partial Differential Equations, 2006, ISBN 91-85523-84-4.
- No 1017 **Daniel Karlsson:** Verification of Component-based Embedded System Designs, 2006, ISBN 91-85523-79-8.

- No 1018 **Ioan Chisalita:** Communication and Networking Techniques for Traffic Safety Systems, 2006, ISBN 91-85523-77-1.
- No 1019 **Tarja Susi:** The Puzzle of Social Activity - The Significance of Tools in Cognition and Cooperation, 2006, ISBN 91-85523-71-2.
- No 1021 **Andrzej Bednarski:** Integrated Optimal Code Generation for Digital Signal Processors, 2006, ISBN 91-85523-69-0.
- No 1022 **Peter Aronsson:** Automatic Parallelization of Equation-Based Simulation Programs, 2006, ISBN 91-85523-68-2.
- No 1023 **Sonia Sangari:** Some Visual Correlates to Focal Accent in Swedish, 2006, ISBN 91-85523-67-4.
- No 1030 **Robert Nilsson:** A Mutation-based Framework for Automated Testing of Timeliness, 2006, ISBN 91-85523-35-6.
- No 1034 **Jon Edvardsson:** Techniques for Automatic Generation of Tests from Programs and Specifications, 2006, ISBN 91-85523-31-3.
- No 1035 **Vaida Jakoniene:** Integration of Biological Data, 2006, ISBN 91-85523-28-3.
- No 1045 **Genevieve Gorrell:** Generalized Hebbian Algorithms for Dimensionality Reduction in Natural Language Processing, 2006, ISBN 91-85643-88-2.
- No 1051 **Yu-Hsing Huang:** Having a New Pair of Glasses - Applying Systemic Accident Models on Road Safety, 2006, ISBN 91-85643-64-5.
- No 1054 **Åsa Hedenskog:** Perceive those things which cannot be seen - A Cognitive Systems Engineering perspective on requirements management, 2006, ISBN 91-85643-57-2.
- No 1061 **Cécile Åberg:** An Evaluation Platform for Semantic Web Technology, 2007, ISBN 91-85643-31-9.
- No 1073 **Mats Grindal:** Handling Combinatorial Explosion in Software Testing, 2007, ISBN 978-91-85715-74-9.
- No 1075 **Almut Herzog:** Usable Security Policies for Runtime Environments, 2007, ISBN 978-91-85715-65-7.
- No 1079 **Magnus Wahlström:** Algorithms, measures, and upper bounds for satisfiability and related problems, 2007, ISBN 978-91-85715-55-8.
- No 1083 **Jesper Andersson:** Dynamic Software Architectures, 2007, ISBN 978-91-85715-46-6.
- No 1086 **Ulf Johansson:** Obtaining Accurate and Comprehensible Data Mining Models - An Evolutionary Approach, 2007, ISBN 978-91-85715-34-3.
- No 1089 **Traian Pop:** Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies, 2007, ISBN 978-91-85715-27-5.
- No 1091 **Gustav Nordh:** Complexity Dichotomies for CSP-related Problems, 2007, ISBN 978-91-85715-20-6.
- No 1106 **Per Ola Kristensson:** Discrete and Continuous Shape Writing for Text Entry and Control, 2007, ISBN 978-91-85831-77-7.
- skap samstämmighet mellan informationssystemarkitektur och verksamhet, 1998. ISBN-9172-19-296-8.
- No 2 **Stefan Cronholm:** Metodverktyg och användbarhet - en studie av datorstödd metodbaserad systemutveckling, 1998. ISBN-9172-19-299-2.
- No 3 **Anders Avdic:** Användare och utvecklare - om anveckling med kalkylprogram, 1999. ISBN-91-7219-606-8.
- No 4 **Owen Eriksson:** Kommunikationskvalitet hos informationssystem och affärsprocesser, 2000. ISBN 91-7219-811-7.
- No 5 **Mikael Lind:** Från system till process - kriterier för processbestämning vid verksamhetsanalys, 2001, ISBN 91-7373-067-X
- No 6 **Ulf Melin:** Koordination och informationssystem i företag och nätverk, 2002, ISBN 91-7373-278-8.
- No 7 **Pär J. Ågerfalk:** Information Systems Actability - Understanding Information Technology as a Tool for Business Action and Communication, 2003, ISBN 91-7373-628-7.
- No 8 **Ulf Seigerroth:** Att förstå och förändra systemutvecklingsverksamheter - en taxonomi för metautveckling, 2003, ISBN91-7373-736-4.
- No 9 **Karin Hedström:** Spår av datoriseringens värden - Effekter av IT i äldreomsorg, 2004, ISBN 91-7373-963-4.
- No 10 **Ewa Braf:** Knowledge Demanded for Action - Studies on Knowledge Mediation in Organisations, 2004, ISBN 91-85295-47-7.
- No 11 **Fredrik Karlsson:** Method Configuration - method and computerized tool support, 2005, ISBN 91-85297-48-8.
- No 12 **Malin Nordström:** Styrbar systemförvaltning - Att organisera systemförvaltningsverksamhet med hjälp av effektiva förvaltningsobjekt, 2005, ISBN 91-85297-60-7.
- No 13 **Stefan Holgersson:** Yrke: POLIS - Yrkeskunskap, motivation, IT-system och andra förutsättningar för polisarbete, 2005, ISBN 91-85299-43-X.
- No 14 **Benneth Christiansson, Marie-Therese Christiansson:** Mötet mellan process och komponent - mot ett ramverk för en verksamhetsnära kravspecifikation vid anskaffning av komponentbaserade informationssystem, 2006, ISBN 91-85643-22-X.

#### Linköping Studies in Information Science

- No 1 **Karin Axelsson:** Metodisk systemstrukturerings- att