EdgeWrite: A Versatile Design for Text Entry and Control

Jacob O. Wobbrock

July 2006

CMU-HCII-06-104

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Brad A. Myers, Chair
Scott E. Hudson
Jennifer Mankoff
Richard C. Simpson, University of Pittsburgh
Shumin Zhai, IBM Almaden Research Center

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

This work was supported in part by the National Science Foundation under grant UA-0308065. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect those of the National Science Foundation. Additional support was provided by General Motors, Microsoft, NEC Foundation of America, NISH, Synaptics, and A.T. Sciences.

Keywords: Text entry, text input, unistrokes, gestures, assistive technology, computer access, universal access, universal design, motor impairments, situational impairments, mobile device, mobile phone, handheld, PDA, stylus, game controller, displacement joystick, isometric joystick, power wheelchair joystick, touchpad, trackball, on-screen keyboard, capacitive sensors, word prediction, word completion, goal crossing, physical edges, EdgeWrite, Pebbles, Fitts' law, Hick-Hyman law, Steering law, Zipf's law.

Abstract

This dissertation presents a versatile design for text entry and control called *EdgeWrite*. EdgeWrite was designed to provide accessible text entry on a variety of platforms to people with motor impairments and to able-bodied users of small devices.

The EdgeWrite design includes a square input area, four corner sensors, corner-sequence recognition, physical edges, goal crossing, and unistroke segmentation. Advanced EdgeWrite features include continuous recognition feedback, non-recognition retry, slip detection, and word-level stroking—concepts not before realized in a text entry method. The EdgeWrite alphabet, which is the same for all EdgeWrite versions, was designed for maximum guessability and learnability through participatory design. A key result is that EdgeWrite is about as guessable as Palm OS Graffiti, a method lauded for its immediate usability.

Multiple versions of EdgeWrite were built and user-tested, including Stylus EdgeWrite for the Palm OS, Joystick EdgeWrite for game controllers and power wheelchairs, Touchpad EdgeWrite, Trackball EdgeWrite, Isometric Joystick EdgeWrite for mobile phones, and EdgeWrite on four keys or sensors. For each version, empirical results from formal user studies were obtained. Of particular importance for users with motor impairments are the results for Stylus and Trackball EdgeWrite, which show marked improvements over existing techniques. In addition, five versions of EdgeWrite built by other researchers further highlight EdgeWrite's versatility.

As part of EdgeWrite's ongoing evaluations, a new character-level error analysis for text entry input streams was developed. This error analysis and the algorithms that automate it constitute a methodological and theoretical contribution to the field of text entry evaluation and measurement.

The thesis is:

> *A versatile design for text entry and control called "EdgeWrite," which uses physical edges, goal crossing, and a minimized need for sensing, is effective on handhelds and desktops for people with motor and situational impairments.*

Acknowledgements

I would like to thank my advisor, Brad A. Myers, for his faithful support and sharp insight. Not one of my projects would have fared as well apart from his input. It has been a real pleasure working with him for five years. Along these lines, I also thank Scott E. Hudson for his helpful comments.

My other committee members deserve special thanks for their time and energy in improving this work. Jennifer Mankoff was a source of directional help. Richard C. Simpson provided valuable access to subjects and resources. Shuman Zhai inspired me by the rigor of his methods and brilliance of his innovations.

Along with Shumin Zhai, I owe an intellectual debt to I. Scott MacKenzie and R. William Soukoreff for escorting text entry into the 21st century. Without these scientists, my work would have been greatly impoverished.

This work has corporate and governmental sponsors and collaborators who deserve thanks: the National Science Foundation, General Motors, Microsoft, NEC Foundation of America, NISH, Synaptics, and A.T. Sciences.

A number of students, faculty, and others have influenced this work. In alphabetical order, they are: Gregory Abowd, Sonya Allin, Lisa Anthony, Chris Atkeson, Htet Htet Aung, Daniel Avrahami, Thi Truong Avrahami, Ryan S. Baker, Aaron Bauer, Gábor Blaskó, Matthew Blythe, Dan Boyarski, Connie Campbell, Duen Horng Chau, Ed Chi, Benjamin Cramer, Marian D'Amico, Kim D'Ewart, Victoria Danforth, Andrew Faulring, James Fogarty, Jodi Forlizzi, Susan Fussell, Darren Gergle, Iván E. González, Steve Hayashi, Mitch Hill, Bonnie John, Geoff Kembel, John Kembel, Silvia Kembel, Pedram Keyani, Sara Kiesler, Roberta Klatzky, Andrew Ko, Panu Korhonen, Robert Kraut, Queenie Kravitz, Johnny C. Lee, Edmund F. LoPresti, Tom Lorusso, Bob Milan, Jeffrey Nichols, Sue O'Connor, Dan Olsen, Roni Rosenfeld, Brandon Rothrock, Joe Rutledge, John SanGiovanni, A. Fleming Seay, Ted Selker, Howard Seltman, Michael Sinclair, John Springsteen, Jeffrey Stylos, Alan Tannenbaum, Janis Thoma-Negley, James A. Vroom, Elaine Wherry, Matthew Wick, Andrew Wilson, Alison N. P. Wobbrock, Nicholas J. Wobbrock, and John Zimmerman. Thank you, all.

My gratitude goes to my church friends in both California and in Pittsburgh. Whether we journeyed only briefly or over the long haul, it was indeed the journey that mattered; and not because of the road we walked, but because of who we walked it with. I admire your commitment to faith.

My family—*Góralska* and American—has earned my deep thanks for their encouragement, generosity, and devotion. My father and mother deserve great thanks for their lifelong love and support. Without them I would not have lasted long in this Ph.D. program. I count on my brother to keep me humble. I know he always will.

Finally, to my wife Ala, to whom this dissertation is dedicated: You have been my strength, my will, and my rock. There is no one for whom I would more gladly punch keys. If I can be half as good a man as you are woman to me, I will be twice as good a man as any man could ever be.

To Ala
Kocham cię

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

This dissertation is motivated by two primary concerns. The first is the need for better desktop computer access, particularly for people with motor impairments. The second is the need for better mobile device accessibility, which affects disabled users and able-bodied users alike. Improving text entry in these domains with a single versatile design is the primary goal of this research.

1.1.1 The Need for Better Computer Access

The value of access to computers and information technology has increased dramatically over the past decade. As the World Wide Web has grown, libraries, news sources, health and financial information, entertainment applications, and other resources have become increasingly available online. The Web's value has further increased with improved search. Other Internet applications like email and instant messaging have also grown in use and importance in keeping in touch with coworkers, friends, and relatives. In just over a decade, the Web has matured from a research curiosity to a factor contributing to one's very quality of life.

Accordingly, the importance of computer access for people with motor impairments has never been greater. People with motor impairments often use wheelchairs, and computer work such as programming, graphic design, or Web site creation may be a way in which impaired individuals can make a living (Keates *et al.* 1998, Myers *et al.* 2002). For people who are homebound, computer-based entertainment may be an engaging alternative to watching television. Access to computers may also allow people to sustain relationships with their families and friends who are located elsewhere, providing communities for those who may otherwise be isolated (Nelson 1994, Shropshire 2003). But all of these activities depend crucially on good computer access.

Although the importance of access has been recognized and numerous assistive technologies exist (Anson 1997, Cook and Hussey 2001), providing access still remains a formidable challenge. Studies show that less than 60% of people who indicate a need for computer access devices actually use them (Fichten *et al.* 2000). Furthermore, at least 35% of purchased solutions are never adopted (Dawe 2006). Complex technologies in particular, like voice recognition systems, are subject to high abandonment rates (Koester 2003). Reasons cited for these failures include the high cost of devices, device complexity, and the need for extensive customization (Dawe 2004). Additionally, the process of acquiring devices often proves prohibitively difficult and fails to incorporate enough input from the people for whom the devices are intended (Riemer-Reiss and Wacker 2000). The need for ongoing maintenance is also a noted problem (Fichten *et al.* 2000). Of prime importance, then, is *simplicity* in both the design of devices and in the process of adopting them (Dawe 2006). In being usable on readily available devices, EdgeWrite provides simpler access solutions.

1.1.2 The Need for Mobile Device Accessibility

Unfortunately, efforts to provide better computer access can no longer be focused on just the desktop. We also face a growing need for better mobile device accessibility. In this context, the term "mobile device" means any of a number of small portable devices, including personal digital assistants (PDAs), mobile phones, pocket jukeboxes (e.g. the Apple iPod), two-way pagers, mobile web browsers, and pocket game devices (Figure 1.1). Mobile devices are not only used by consumers, but are also used in the workplace for tracking inventories, acquiring pricing information, supporting field investigations, and communicating among employees. In fact, estimates from 2004 projected that by 2006, 10% of the American workforce would be entirely mobile (i.e. without an office) (York and Pendharkar 2004).

Figure 1.1. Numerous mobile devices exist on the market today, but many of these devices remain inaccessible to both motor- and situationally-impaired users.

Despite the growing prevalence of mobile devices in our society, there has been almost no work on making them more accessible (Bertini and Kimani 2003). (A rare exception is Blair's customizable hardware interface (Blair 2005), which allows large external buttons to access features on some mobile phones.) Small fonts and low contrast make reading difficult (Mustonen *et al.* 2004), small soft buttons and input areas make stylus actions prone to error, and low strength can make it difficult to press physical buttons or apply consistent pressure to the screen (Myers *et al.* 2002). Although it is a tired term, the "digital divide" is applicable here, and it threatens to grow if handheld devices are not made more accessible to people with disabilities.

The need for better mobile device accessibility is not only felt by people with motor impairments, but also by able-bodied users. Recently, manufacturers have been pressured to improve mobile device accessibility, motivated not just by the needs of disabled users but also by the needs of able-bodied users "on the go" (BBC News 2006). This points toward a concept that recent academic work has dubbed "situationally-induced impairments and disabilities" (Sears *et al.* 2003, Sears and Young 2003), or just "situational impairments" for short. This phrase refers to the temporary, ephemeral impairments that mobile device users incur due to the varied contexts in which they operate. For example, a person walking down a dim street on a cold night trying to dial his or her cell phone while wearing gloves incurs vibrational tremor, reduced visual information, divided attention, and impaired motor skills. A less extreme case is simply trying to read the screen of a mobile device while walking (Mustonen *et al.* 2004). Devices are currently inaccessible to situationally-impaired users for many of the same reasons that they are inaccessible to people with physical disabilities: small fonts, poor contrast, awkward input mechanisms, shrunken targets, tiny dials, miniature joysticks, the need for two hands, and cramped finger postures all compromise usability. Therefore, a design well-suited to meet the needs of motor-impaired users may also help the able-bodied mainstream. This is the spirit behind "universal design" (Mace *et al.* 1991, Steinfeld 1994), and a key motivator of the current work.

1.1.3 A Versatile Design for Text Entry and Control

The concerns above highlight the need for better accessibility on both desktop personal computers (PCs) and mobile devices. A successful solution for desktops must be simple, cheap, robust, readily available, and require little configuration and maintenance (Dawe 2004, Dawe 2006). A successful solution on mobile devices must provide support for overcoming situational impairments through properties like physical stability, high tactility, increased visibility, and comfort (Hirotaka 2003, Silfverberg 2003, Wobbrock and Myers 2005b). Not surprisingly, these properties will probably improve device accessibility for motor-impaired users as well.

Although the use of computers involves many different activities, *text entry* consumes a great deal of users' time and energy. In a single month, many desktop users strike hundreds of thousands or even millions of keys (MacKenzie and Soukoreff 2002b), and European mobile phone users send 15 billion text messages (GSM World 2004). Clearly, text entry is a considerable part of human-computer interaction (HCI), and therefore an important part of computer access.

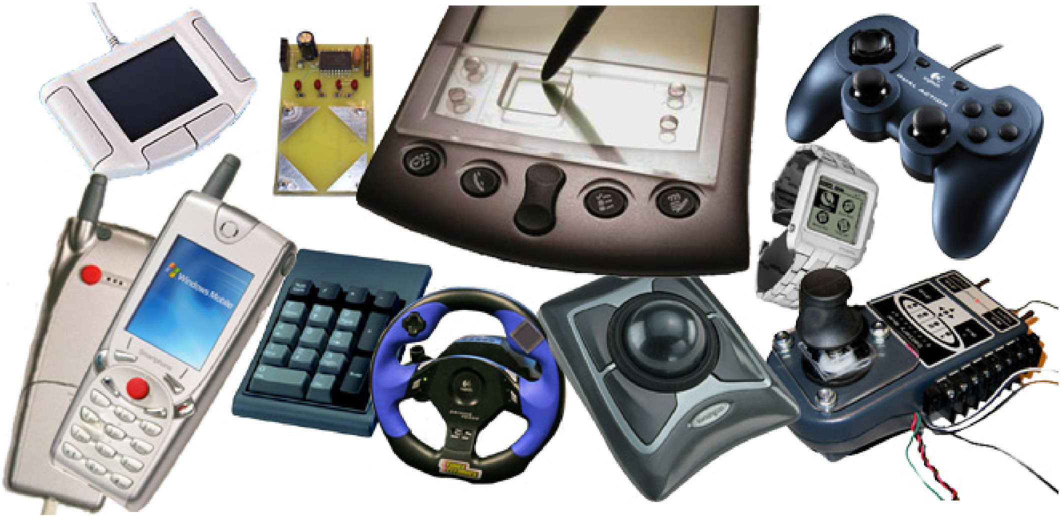


Figure 1.2. The devices on which EdgeWrite text entry has been implemented.

Furthermore, a significant part of the text entry process involves *control*. For example, research shows that the four cursor keys comprise upwards of 10% of all keystrokes made during general desktop PC use, rivaling even the use of the SPACE key (MacKenzie and Soukoreff 2002b). Other control operations include accessing application menus related to text entry (e.g. for font selection), tabbing among dialog elements, and confirming selections with ENTER.

Accordingly, a *versatile* design for text entry and control has been created that addresses these needs. The design is versatile in that it can adapt to various devices which are inexpensive and "off the shelf" (Figure 1.2), it is intended to be used in both office and mobile contexts, it has accessibility and mainstream audiences, and it is technologically simple, requiring little more than a few sensors for input, which makes it suitable for both high- and low-end computing platforms. Having a versatile design of this type is important because:

- Users with motor impairments often experience rapid fatigue (Sears and Young 2003). A versatile design would allow users to switch among devices to distribute strain across different muscles while using the same essential technique.

- Some diseases are degenerative. A versatile design would permit users to switch to new devices as their abilities change without having to learn new techniques.

- Adapting new techniques to users is a lengthy and expensive process involving therapists and assistive technology (AT) professionals (Tremaine 2001). A versatile design may shorten this process by providing alternative devices for the same already-known technique.

- A versatile design may function as part of an "integrated control system" where text and mousing are available on the same device, reducing device switching and the need for multiple control sites, which have been shown to be barriers to access (Guerette and Sumi 1994, Spaeth *et al.* 1998).

- New mobile devices appear often, and few people are eager to learn a new input technique for every new device they acquire. A versatile design may be easily implemented on new devices while placing minimal constraints on the design of devices themselves. This may be called *ubi-input*, which is "learn once, write anywhere" (Wobbrock and Myers 2005a).

- Formerly non-computerized devices are now being given computing power. For instance, wrist watches used to be mechanical devices but now can serve as full-fledged PDAs (Raghunath and Narayanaswami 2002, Blaskó and Feiner 2004). As more devices become computerized, they may need some rudimentary form of text entry. A versatile design may work for these new devices.

- As the storage capacity of mobile devices increases, the desire for text-driven features like *type-ahead* and *search* may also increase. These features require a versatile design for text entry capable of functioning on different types of mobile devices, such as pocket jukeboxes and digital cameras, both of which can store large amounts of data.

Although there has been a flurry of text entry research in the last decade (MacKenzie and Soukoreff 2002b, Zhai *et al.* 2005), few methods have been reusable across desktop PCs and mobile devices, and even fewer have sought to be accessible to special populations. While high-performance methods tailored to specific devices and applications will remain useful, a versatile design for text entry and control can help improve the accessibility of both desktops and mobile devices for motor-impaired and situationally-impaired users. The design that addresses these problems is called *EdgeWrite.*

The following thesis will be demonstrated:

> *A versatile design for text entry and control called "EdgeWrite," which uses physical edges, goal crossing, and a minimized need for sensing, is effective on handhelds and desktops for people with motor and situational impairments.*

1.2 The EdgeWrite Concept

At its most basic, EdgeWrite is a *unistroke* text entry method that relies on four corners arranged in a square (see section 3.2). The order in which the four corners are entered during an EdgeWrite stroke determines the character being made. For styli (§4), displacement joysticks (§5–6), and touchpads (§6), physical edges connect the four corners of a square and provide stability when writing.[1] For trackballs (§7) and isometric joysticks (§8), physical edges are impractical, so *goal crossing* is used instead (Accot and Zhai 1997). On versions with just four keys or sensors (§9), corners correspond directly to the input mechanisms, allowing gestures to be defined on minimal devices.

The EdgeWrite alphabet was designed in multiple stages. The final alphabet is largely a result of a participatory design process intended to maximize guessability and learnability (§3.3.3). Figure 1.3 shows the word "edgewrite" using depictions of EdgeWrite strokes. Note that the curvature of the line segments is for readability only, particularly in the event that a stroke passes more than once over the same edge. In EdgeWrite, all actual motion is in straight lines along edges or diagonals and into corners.

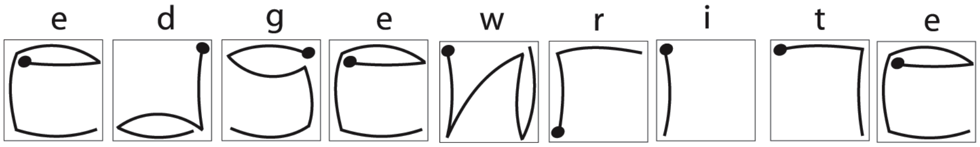


Figure 1.3. The word "edgewrite" written with depictions from the EdgeWrite character chart. A heavy dot indicates the start of each letter.

Some versions of EdgeWrite are targeted at people with motor impairments, such as the versions for styli, power wheelchair joysticks, touchpads, and trackballs. Other versions are targeted at able-bodied users in situations where conventional text entry is not feasible, such as with a displacement joystick on a video game console, or with an isometric joystick on a mobile phone. Thus, EdgeWrite is versatile across devices, users, and contexts of use.

1.3 Research Approach

This research has been based on a highly iterative "design, build, and user test" cycle where EdgeWrite versions are rapidly constructed and then evaluated in laboratory or field settings with real users. Extensive log files are continuously written as a part of this process to enable informed design decisions based on quantitative data. Refinements are then made before subsequent rounds of user testing.

[1] See the figure atop page 79 for an example.

The test protocols usually pit a version of EdgeWrite against a competitor method, often a *de facto* standard, either in a between- or within-subjects design, depending on carryover effects. Evaluations may be of novice users after minimal training, of expert users already familiar with the technique, or of novice-to-expert users over a longitudinal study. Longitudinal studies are the most informative because we can observe and model learning with the power law (Card *et al.* 1983), and find "crossover points" where one method overtakes another (MacKenzie and Zhang 1999). However, longitudinal studies are expensive in both time and money, so single-session studies were often used. To facilitate laboratory studies, a text entry evaluation suite and accompanying log file analyzer were built (§11).

The current work also yielded new procedures for improving guessability (§3.3.3) and for analyzing data from text entry experiments (§11.5). These procedures can be reused by other researchers.

Accompanying the empirical work is some theoretical modeling, particularly of Trackball EdgeWrite (§7.3.1). The theoretical models allow for upper-bound comparisons when exploring alternative designs. The theories utilized include Fitts' law (Fitts 1954, MacKenzie 1992), the Steering law (Accot and Zhai 1997), the Hick-Hyman law for visual search (Hick 1952, Hyman 1953), and Zipf's law for language (Zipf 1932).

1.4 Dissertation Organization

This chapter has illustrated the potential value of a versatile design for text entry. Here is a roadmap to subsequent chapters:

Chapter 2 describes prior work and its relation to EdgeWrite. Some of the work relates to EdgeWrite in general, while other work relates to specific EdgeWrite versions.

Chapter 3 introduces the core design of EdgeWrite, including fundamental concepts like corners, edges, recognition, segmentation, and "beating Fitts' law" through the use of physical edges and goal crossing. Section 3.3 describes the EdgeWrite alphabet and the guessability procedure used to create it. Section 3.5 describes *Fisch*, EdgeWrite's design for word-level stroking.

Chapter 4 introduces the original version of EdgeWrite, the stylus version for PDAs. Results show that both able-bodied users and motor-impaired users are more accurate while writing with EdgeWrite than with its competitor, Graffiti.

Chapter 5 discusses Joystick EdgeWrite for use with computer game controllers. EdgeWrite's adaptation to the joystick is described, along with results that show EdgeWrite is faster and produces more accurate text than two competitor methods, selection keyboard and date stamp.

Chapter 6 discusses two techniques for text entry from power wheelchairs, PW-Joystick EdgeWrite and Touchpad EdgeWrite. Both single session and longitudinal studies are presented. Results show that Touchpad EdgeWrite is faster and more pleasing to subjects than PW-Joystick EdgeWrite, but both are more error prone than using the same devices to control a commercial on-screen keyboard called WiViK.

Chapter 7 describes Trackball EdgeWrite, the first version without physical edges. Instead of physical edges, Trackball EdgeWrite uses the mouse cursor, and *goal crossing* provides for the underlying writing mechanism. Results for character-level and word-level versions are presented, showing Trackball EdgeWrite's superiority over on-screen keyboards in speed, accuracy, and subjective satisfaction.

Chapter 8 applies the design for Trackball EdgeWrite to isometric joysticks embedded in mobile phones. Both trackballs and isometric joysticks lack a notion of position, making them equally suited to the same software. Longitudinal results show that character-level EdgeWrite on an isometric joystick rivals Multitap in speed, and word-level EdgeWrite rivals T9.

Chapter 9 presents a feasibility study of EdgeWrite on four keys, where each key serves as one of the EdgeWrite corners. Results from a longitudinal study of able-bodied users show EdgeWrite's superiority to prior 3-key and 5-key methods. The 4-key EdgeWrite design is then adapted to four capacitive sensors, showing how EdgeWrite can be built with minimal sensing technology.

Chapter 10 briefly describes versions of EdgeWrite built by other researchers independent of the current work. These include versions for steering wheels, musical expression, watch faces, game devices, and joysticks.

Chapter 11 discusses a new character-level error analysis for text entry input streams. The algorithms presented are used to analyze data from an early EdgeWrite study. The *TextTest* and *StreamAnalyzer* tools are also presented.

Chapter 12 concludes with a “design space,” reflections and insights, major empirical results, a list of contributions, future work, and some closing remarks.

Chapter 2

Related Work

2.1 Universal Design

The philosophical foundations of the current work lie in universal design (Mace *et al.* 1991, Steinfeld 1994). The term "universal design" appeared first in the literature of architectural design in 1991 (Mace *et al.* 1991). Since most buildings may be used by anyone, the principle of making them accessible to everyone naturally followed. The movement that had begun in 1968 with the *Architectural Barriers Act* had been given new life in universal design. It was not long before the same barrier-reducing concepts were applied to products (Steinfeld 1994) and software (Bergman and Johnson 1995). In 1997, a group of architects, product designers, and engineers ratified the second version of the influential *Principles for Universal Design* (Connell *et al.* 1997), which has become widely used as a complement to mainstream usability principles (Nielsen 1993). The field of assistive technology, which began as a rehabilitation science and engineering discipline after Word War II, and newer approaches like "design for all" (Stary 1997) and "inclusive design" (Keates *et al.* 2000), have joined forces in the effort to meet the needs of an aging population with more handicapped and elderly than ever before. Although there are differences among these and other approaches, they share the common goal of removing or overcoming barriers to access (Law *et al.* 2005).

The central idea in universal design is that all products and environments should be designed so as to be usable by the greatest number of people (Mace *et al.* 1991).

Although *truly* universal designs are unrealizable in practice (Keates *et al.* 2000, Hull 2004), striving for this goal often enforces an ethic of simplicity and elegance. Proponents of universal design point to examples where those with special needs and the general population both benefit: curbs lowered for wheelchairs are valued by bicyclists, automatic doors open for people in wheelchairs and shoppers pushing grocery carts, advances in ergonomic seating help paraplegics and deskbound office workers, and trackballs aid computer users with carpel tunnel and travelers without enough desk space for a mouse. In these and many other cases, universal design has benefited both the physically-impaired and the situationally-impaired.

Universal design views “impairment” differently than assistive technology traditionally has. The eyes of universal design do not see two classes of users—the impaired and the unimpaired. Instead, they see all users on a continuum of impairment. Whereas one user may have a permanent physical disability, another may incur a temporary situational impairment while under certain constraints. Thus, impairment is not a static concept, but a fluid one that moves with context and constraint (Sears and Young 2003).

It is from this philosophical stance that EdgeWrite is offered as a versatile design for text entry, one which has demonstrated benefits for both motor-impaired and able-bodied users. For example, Stylus EdgeWrite (§4) provides higher accuracy to writers through the use of stabilizing physical edges. This higher accuracy has been demonstrated for both motor-impaired and able-bodied users (Wobbrock *et al.* 2003b). That both specialized and mainstream populations benefit from EdgeWrite is a case-in-point from the standpoint of universal design.

2.1.1 Situational Impairments

Although universal design has implicitly contained the notion of “situational impairments,” this term and its inclusion in disability discourse is fairly recent. In 2003, Sears *et al.* and Sears and Young coined the term *situationally-induced impairments and disabilities* (Sears *et al.* 2003, Sears and Young 2003). They point to various environmental factors (e.g. lighting, noise, vibration, and temperature) and various contextual factors (e.g. engagement in multiple tasks demanding cognitive, perceptual, and physical resources) as factors contributing to situational impairments. They offer two examples to illustrate this. In the first, a worker outdoors whose fingers are cold experiences a motor impairment not unlike a person with arthritis. In the second, a person trying to enter text with a stylus while riding down a bumpy road experiences tremor not unlike some people with Parkinson's.

Which physical impairments and corresponding situational impairments might be aided by similar designs for text entry? Certainly a blind user is not equivalent to an eyes-free sighted user, nor is a tremulous user equivalent to someone ambling down a bumpy sidewalk. But designs that aid the motor-impaired may nonetheless have properties that aid the situationally-impaired. For instance, EdgeWrite has shown that *physical stability* is one property that improves accuracy, even in the presence of tremor (Wobbrock *et al.* 2003b). Similarly, Silfverberg has shown that *high tactility* contributes to more usable mobile phone keypads, particularly with limited or no visual feedback, which would be the case for both blind and eyes-free users (Silfverberg 2003).

Recently, some researchers have studied the effects of walking on mobile device use. Price *et al.* found that speech recognition error rates are significantly higher, about 15.5%, when walking compared to when seated (Price *et al.* 2004). Lin *et al.* studied stylus tapping, finding that walking decreases throughput from 7.8 bits per second (BPS) to 6.6 BPS (Lin *et al.* 2005). Marentakis and Brewster had subjects point to spatial audio targets while walking through cones, finding that pointing speed and accuracy degraded 20% compared to standing still (Marentakis and Brewster 2006). Mustonen *et al.* found that walking reduces reading speed by about 18% (Mustonen *et al.* 2004). Oulasvirta *et al.* found that mobile attention spans are between 4–8 seconds, compared to 14–16 seconds in laboratory settings (Oulasvirta *et al.* 2005). Kjeldskov and Stage proposed six mobile usability evaluation scenarios based on walking different tracks, finding that none resulted in the same perceived workload as walking a real pedestrian street, but that user performance was not significantly different for a variety of tasks (Kjeldskov and Stage 2004). Although there are other situational impairments besides those caused by walking, this work represents an important start to this line of inquiry.

2.1.2 Mobile Device Accessibility

The value of mobile devices to people with disabilities has been observed (Abascal and Civit 2001), but surprisingly little work has assessed the accessibility of mobile devices. Exceptions include the study by Burzagli *et al.*, which found that blind users had trouble entering text and browsing on PDAs, largely due to the lack of screen-reading software

(Burzagli *et al.* 2005). Leonard *et al.* found that users with age-related macular degeneration could still use handheld computers, but that high contrast was essential for efficient use (Leonard *et al.* 2005). These findings have implications for device manufacturers, since small devices may lack the output capabilities of desktop computers.

A few prototyping efforts have sought to improve the accessibility of mobile devices. Manaris *et al.* developed *SUITEKeys*, a speech recognition "keyboard" originally developed for desktops but which the authors speculated could be ported to mobile devices (Manaris *et al.* 1999). Blair created a set of hardware buttons that interface with Symbian mobile phones, providing better access to some users with motor impairments (Blair 2005). Göransson redesigned a PDA to better support users with Parkinson's disease (Göransson 2004). Lee *et al.* created a pair of chording gloves that can accommodate both Korean and Braille as fingers are pressed into the thumb (Lee *et al.* 2003, Lee *et al.* 2005). Among these efforts, EdgeWrite is the most related to the problem of PDA text entry for people with motor impairments (Wobbrock *et al.* 2003b). But text entry is only one piece of the problem, and clearly there is much farther to go in achieving mobile accessibility for all.

2.2 Beating Fitts' Law

Fitts' law models the time required to access a target as a function of the target size *W* and its distance away *A* (Fitts 1954). Not surprisingly, it takes longer to successfully click or tap on small or distant targets. Figure 2.1 illustrates these parameters.

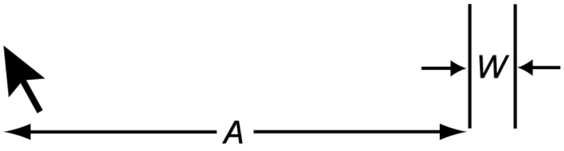


Figure 2.1. Fitts' law models the time required to access a target of size *W* at a distance *A* units away.

Equation 2.1 is a commonly used formulation of Fitts' law (MacKenzie 1992).

$$T = a + b \cdot \log_2\left(\frac{A}{W} + 1\right) \tag{2.1}$$

In Equation 2.1, *a* and *b* are empirical coefficients determined by regression. A powerful aspect of Fitts' law is that the units of *A* and *W* cancel, allowing us to compare results across different experimental apparatuses.

Accordingly, "beating Fitts' law" refers to improving target acquisition performance through interaction techniques that either decrease the effective target distance *A* or increase the effective target size *W* (Balakrishnan 2004). Perhaps the earliest examples of the latter were in Ivan Sutherland's Sketchpad system (Sutherland 1963), which included gravity fields, snapping behavior, and drawing constraints. In accessibility, Hwang *et al.* has investigated the effects of gravity fields on motor-impaired users (Hwang 2002, Hwang *et al.* 2003). Other means of increasing target size are area cursors and sticky icons (Worden *et al.* 1997), which are helpful for older adults who may have reduced motor skill. More recently, the Bubble Cursor was a dynamic extension of area cursors (Grossman and Balakrishnan 2005). Although a complete review of such techniques is beyond the current scope, it is important to note that most EdgeWrite versions "beat Fitts' law" through either physical edges or goal crossing, both of which can be regarded as increasing the effective target size.

2.2.1 Edges in the Interface

Physical edges have been used in user interfaces before EdgeWrite. A charming text entry example comes from Lewis Carroll, author of *Alice in Wonderland.* Carroll published the following description of his invention, the *Nyctograph*, in the October 29, 1891 issue of *The Lady*:

> Any one who has tried, as I have often done, the process of getting out of bed at 2 a.m. in a winter night, lighting a candle, and recording some happy thought which would probably be otherwise forgotten, will agree with me it entails much discomfort. All I have now to do, if I wake and think of something I wish to record, is to draw from under the pillow a small memorandum book containing my Nyctograph, write a few lines, or even a few pages, without even putting the hands outside the bed-clothes, replace the book, and go to sleep again. … I tried rows of square holes, each to hold one letter (quarter of an inch square I found a very convenient size), and this proved a much better plan than the former; but the letters were still apt to be illegible. Then I said to myself 'Why not invent a square alphabet, using only dots at the corners, and lines along the sides?' I soon found that, to make the writing easy to read, it was necessary to know where each square began. This I secured by the rule that *every* square-letter should contain a large black dot in the N.W. corner. … [I] succeeded in getting 23 of [the square-letters] to have a distinct resemblance to the letters they were to represent. Think of the number of lonely hours a blind man often spends doing nothing, when he would gladly record his thoughts, and

> you will realise what a blessing you can confer on him by giving him a small ‘indelible’ memorandum-book, with a piece of paste-board containing rows of square holes, and teaching him the square-alphabet (Carroll 1891).

The Nyctograph itself was probably quite simple, just a piece of paste-board with a series of square cut-outs arranged in a grid. Carroll, who apparently suffered from insomnia, would write his square-letters in the square holes “under the bed-clothes” by running his pen along the interior edges of the holes. In the morning, he would transcribe his notes in his regular hand.

Carroll’s description is full of concepts related to EdgeWrite. First, the invention is inspired by situational impairments, namely fatigue, cold, and darkness. Second, it is designed to be eyes-free through the use of physical edges. Third, in designing his alphabet, Carroll prizes similarity to Roman letters. Fourth, Carroll sees an opportunity to apply his invention to people with disabilities. A major difference from EdgeWrite, besides the obvious fact that computers were nowhere involved, is that many of Carroll’s square-letters were comprised of multiple disjoint strokes. He therefore solved the visual segmentation problem by placing a heavy black dot in the top-left corner of each letter. Figure 2.2 shows the Nyctograph letters (Tannenbaum 2005).

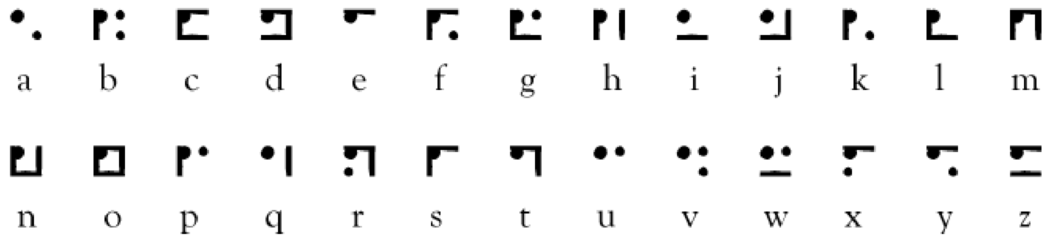
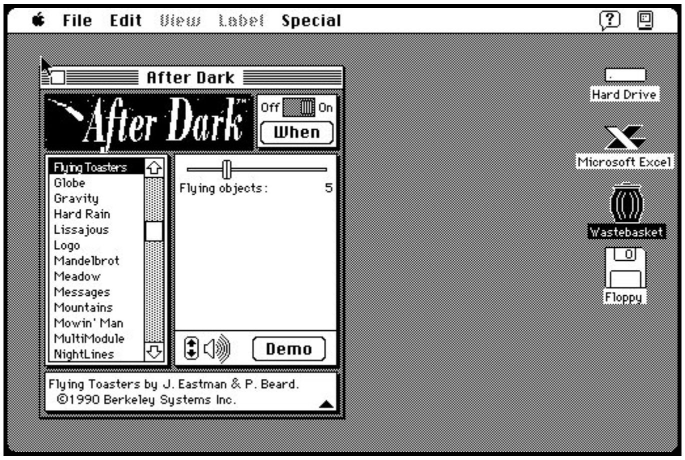


Figure 2.2. Lewis Carroll’s Nyctograph alphabet, written under the bed-clothes inside a series of square holes, one letter per hole. Courtesy of Alan Tannenbaum.

Nearly a century later, the Apple Lisa appeared in January 1983 (Perkins *et al.* 1997, Ludolph and Perkins 1998). Among many other novelties, the Lisa sported a *menu bar* atop its graphical desktop screen (Figure 2.3). The menu bar, which still exists on Apple Macintosh computers today, had an impenetrable border along its top edge, allowing mouse cursors to overshoot menu targets arbitrarily. This impenetrable edge has been shown to improve target acquisition times over Microsoft Windows-style “walking” or “floating” menus (Walker and Smelcer 1990, Accot and Zhai 2002). This type of result has been examined further by Farris *et al.* for different approach angles (Farris *et al.*

2002b), target distances (Farris *et al.* 2002a), and target sizes (Johnson *et al.* 2003) using impenetrable edges along browser windows (Farris *et al.* 2001).



Flying objects: 5

Flying Toasters by J. Eastman & P. Beard.
©1990 Berkeley Systems Inc.

Figure 2.3. The Macintosh menu bar and the After Dark screen saver, both of which used impenetrable screen edges to improve target acquisition.

The original *After Dark* screen saver also used edges. The program, also shown in Figure 2.3, allowed users to "throw" their mouse cursor into the corner of their desktop when they wanted to initiate their screen saver before leaving their computer. The corner of the screen "pockets" the cursor, much like the plastic template used in Stylus EdgeWrite traps a moving stylus (§4.2.2).

2.2.2 Goal Crossing

The trackball (§7) and isometric joystick (§8) versions of EdgeWrite do not use physical edges. Instead they use *goal crossing* to facilitate writing. Unlike pointing, which requires a cursor to enter an area target and remain stationary long enough to select it, crossing requires only that a goal line be penetrated, like a football player scoring a touchdown (Figure 2.4). Thus, one can think of the crossing goal as providing a similar benefit as an impenetrable edge. In both cases, the cursor is free to overshoot the target by an arbitrary amount.

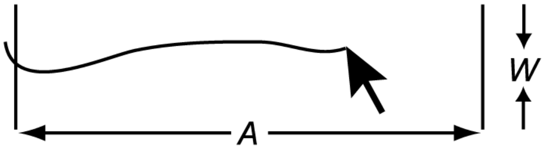


Figure 2.4. Crossing follows Fitts' law but with the *W* constraint orthogonal to the movement trajectory, rather than collinear.

Accot and Zhai have shown that crossing follows the Fitts' law formulation shown in Equation 2.1 (Accot and Zhai 1997), but that the regression coefficients *a* and *b* differ than for the traditional Fitts' reciprocal pointing task (Accot and Zhai 2002). In particular, for index of difficulties (ID) less than 4 bits, continuous orthogonal goal crossing is faster than pointing. On trackballs and isometric joysticks, EdgeWrite motion uses IDs ranging from 0.7–0.9 bits, where crossing performs very well.

In their development of *CrossY*, Apitz and Guimbretière found that crossing can indeed serve as the basis for a full-fledged application, replacing traditional point-and-click interactions for buttons, checkboxes, radio buttons, scroll bars, menus, and dialog boxes (Apitz and Guimbretière 2004).

Marking menus (Kurtenbach and Buxton 1994) are also related to EdgeWrite, as one could think of EdgeWrite letters as selections unfolding from hierarchical marking menus. Marking menus also "beat Fitts' law" in a way similar to crossing, since only the direction of a mark determines the selection, not the stroke length. Also, the stroke-based word prediction and completion system in Trackball EdgeWrite involves selecting words in a style similar to marking menus (§3.5.2).

2.3 Input Devices and Techniques

Text entry is an area of research dating back to the 1950s. Early influential works include Gentner *et al.*'s glossary of error terms (Gentner *et al.* 1984) and Grudin's analysis of transcription typing errors (Grudin 1984). Recently, mobile devices have spurred a renaissance of sorts for text entry research due to the constraints imposed and the challenges of small devices. Zhai has described the desirable properties of mobile text input methods (Zhai *et al.* 2005), and MacKenzie and Soukoreff have provided a thorough overview of mobile text entry methods (MacKenzie and Soukoreff 2002b). Readers are directed to these sources for more details.

This section discusses the related work in text entry for a variety of input devices. The first subsection highlights three "multi-device" methods of particular relevance to EdgeWrite. The following subsections are divided according to device type, roughly following this dissertation's chapter organization.

2.3.1 Multi-device Input

Multi-device adaptability is an important aspect of EdgeWrite's versatility. Although most text entry methods are not explicitly designed for multiple devices, three stand out as particularly relevant to EdgeWrite in this regard. The first is *Dasher*, a text entry method that can be used with any cursor control device (Ward *et al.* 2000). With Dasher (Figure 2.5), one moves through dynamic expanding letter regions, the sizes of which correspond to a letter's likelihood of entry. Although Dasher can achieve rapid entry rates (~30 WPM), a common sentiment, particularly among novices, is that it can be overwhelming because it is in constant visual flux. This also makes it difficult to use Dasher purely by feel, since one must constantly attend to the changing display. Dasher can be used on a PDA with a stylus, or with an eye-tracker for hands-free input (Ward and MacKay 2002).

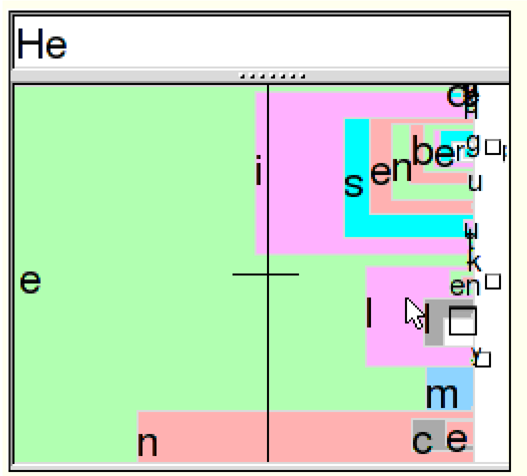


Figure 2.5. In Dasher, probabilistic letter regions expand rapidly from right-to-left toward the mouse cursor. Here the user is writing "Hello." Used with permission.

The second multi-device method is the Minimal Device Independent Text Input Method, or *MDITIM* (Isokoski and Raisamo 2000). MDITIM defines all letters in terms of *north*, *east*, *south*, and *west* primitives (Figure 2.6). As a result, it can be used with touchpads, trackballs, mice, joysticks, and keyboards. The downside of MDITIM is that letter shapes generally do not mimic their Roman counterparts in either look or feel. In contrast, the EdgeWrite alphabet was designed to maintain mnemonic correspondence with Roman letters (§3.3.3).

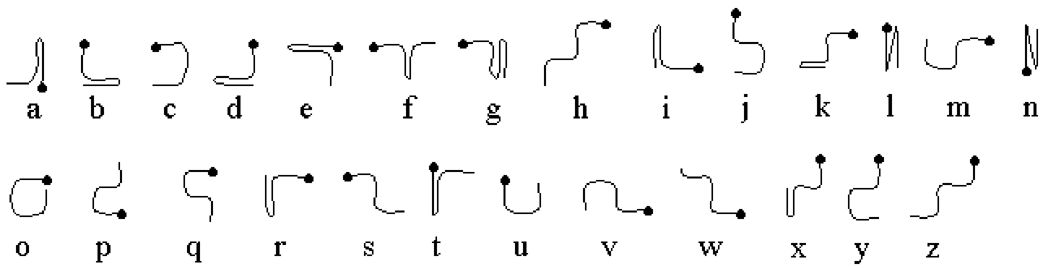


Figure 2.6. The MDITIM alphabet. Image adapted from (MacKenzie and Soukoreff 2002b). Used with permission.

In a study of MDITIM on different devices (Isokoski and Raisamo 2000), subjects entered text with a touchpad for nine 30-minute sessions followed by a single session in which they used a touchpad, trackball, mouse, joystick, and keyboard.[2] Respective speeds (WPM) and error rates were about 7.5 (6.2%), 6.5 (7.3%), 6.3 (4.8%), 5.6 (3.0%), and 4.9 (3.2%). These speeds are quite a bit slower than EdgeWrite on the same devices (Wobbrock and Myers 2005a). Like Dasher, MDITIM has been adapted for use with an eye-tracker, although performance results were not given (Isokoski 2000).

The third multi-device method is *Quikwriting* (Perlin 1998) and its evolution into Microsoft's *Xnav* (Kanellos 2006). Quikwriting defines letter regions about the perimeter of a square area (Figure 2.7). One enters a letter by moving from the center to one of the regions, possibly into a second region, and then back to the center. Since all letters begin and end in the center, words and phrases can be written fluidly as single continuous strokes. Isokoski studied the use of Quikwriting with styli, joysticks, and keyboards (Isokoski and Raisamo 2004). After 10 hours of practice, subjects reached 16 WPM with the stylus and 13 WPM with the joystick with uncorrected error rates of <1%. After this, subjects were tested with the keyboard version and reached 8 WPM. Isokoski concluded that Quikwriting is difficult to learn and not particularly fast, but feasible for writing on multiple devices.

[2] I calculated the keyboard method's *keystrokes per character* (KSPC) to be 3.06. See (MacKenzie 2002b) for more details on calculating characteristic KSPC.

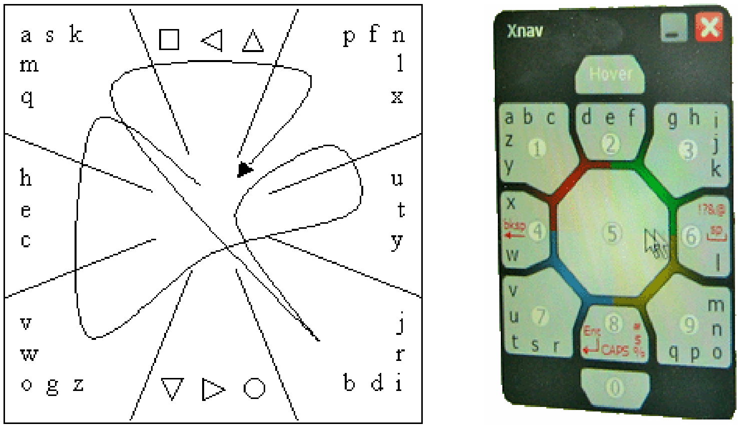


Figure 2.7. Quikwriting and its successor, Xnav. The stroke on the left produces the word “quik”. Images are from (MacKenzie and Soukoreff 2002b) and (Kanellos 2006). Used with permission.

Xnav has taken Quikwriting further by designing an alphabetical layout and creating versions for Xbox joysticks, capacitive sensing phone keypads, hover-tracking Tablet PCs, ultra-mobile PCs, and television remote controls. In addition, Xnav implements East Asian languages like traditional and simplified Chinese.

2.3.2 Stylus Input

Unistroke alphabets and “soft” stylus keyboards are two predominant forms of stylus-based text entry. As noted above (§1.2), EdgeWrite is an example of the former. This section therefore focuses on unistroke systems. Information on stylus keyboards can be found elsewhere (Soukoreff and MacKenzie 1995, Lewis *et al.* 1999a, Lewis *et al.* 1999b, MacKenzie and Zhang 1999, Zhai *et al.* 2000, MacKenzie and Soukoreff 2002b, Zhai *et al.* 2002).

The earliest example of a unistroke alphabet was *Notae Tironianae*, designed by a freed slave of Cicero in 63 B.C. (Buxton 2005). The earliest electronic character stroke recognition system was probably Dimond’s from 1957, which used electric plates and an energized stylus (Dimond 1957). In 1993, Xerox PARC’s *Unistrokes* (Goldberg and Richardson 1993) formalized the single-stroke concept and coined the term, but it wasn’t until Palm OS released *Graffiti* (Figure 2.8) that unistrokes became popularized (Blickenstorfer 1995). The success of Graffiti is often attributed, at least in part, to its similarity to hand-printed Roman letters (MacKenzie and Zhang 1997)—a property Unistrokes lacked (MacKenzie and Soukoreff 2002b). The importance of Roman similarity is preserved in EdgeWrite’s mnemonic strokes (§3.3.3).

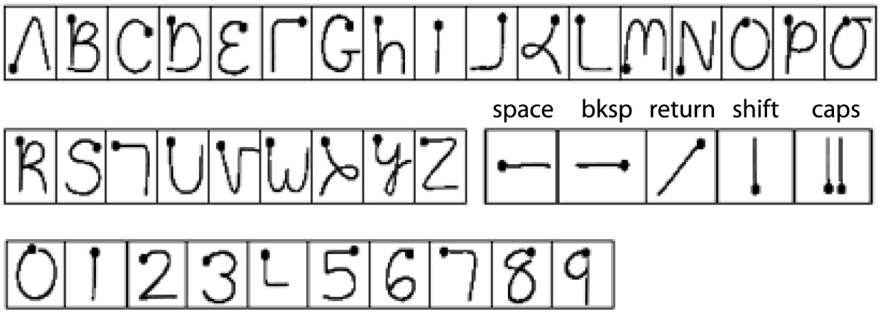


Figure 2.8. The Graffiti alphabet and numerals. Note the similarity to hand-printed Roman alphanumerics for most characters.

The character recognition approach in EdgeWrite is based on a sequence of corner regions. Other region-based recognizers include *LibStroke* (Willey 1997) and *Xstroke* (Worth 2003), which tokenize strokes based on underlying grids of nine regions.

A few studies have compared the performance of unistroke text entry systems. Fleetwood *et al.* compared Graffiti to the Palm OS stylus keyboard and found that novices were faster with the keyboard (7 *vs.* 16 WPM) but experts were faster with Graffiti (21 *vs.* 18 WPM) (Fleetwood *et al.* 2002). For both novices and experts, Graffiti error rates were higher than stylus keyboard error rates (9% *vs.* 2%). Költringer and Grechenig found similar results for a study of novices using Graffiti 2 (9 WPM, 19% errors) and the Palm OS stylus keyboard (13 WPM, 4% errors) (Költringer and Grechenig 2004). Sears and Arora compared novice performance with Jot and Graffiti, finding Jot to be significantly faster (7.37 *vs.* 4.85 WPM) but not detectably different in terms of uncorrected errors (10.1% *vs.* 14.8%) (Sears and Arora 2002). Lewis compared the entry rates of two keyboards—stylus QWERTY and a predictive dynamic layout—with "perfect" handwriting recognition (Lewis 1999). He found that natural handwriting was fastest (23 WPM), the standard QWERTY second (14 WPM), and a dynamic layout third (6 WPM). However, subjects rated handwriting the most difficult due to the device's small size.

Although many of these results favor stylus keyboards, there are significant drawbacks of soft keyboards not captured by these studies. These include the addition of a second focus-of-attention (FOA), inability to write "by feel," increased visual fatigue, the need for screen real estate, and the need for collocated input and output (i.e. a touch screen). Tapping on a stylus keyboard is akin to typing in a "hunt and peck" fashion. Gestural systems largely avoid these problems but require their strokes to be learned.

2.3.3 Word-level Stroking

EdgeWrite includes not only the ability to make character-level strokes, but also *word-level strokes* (§3.5). This subsection reviews other methods relevant to word-level strokes.

An innovative hybrid of unistrokes and stylus keyboards is *SHARK* (Zhai and Kristensson 2003) and *SHARK²* (Kristensson and Zhai 2004). These stylus keyboards[3] allow users to either tap words conventionally or stroke words as fluid unistrokes (Figure 2.9). Importantly, the stroking pattern for a word is the same as its tapping pattern, which reinforces learning as users transition from tapping to stroking. Thus, SHARK avoids many of the aforementioned drawbacks of stylus keyboards. Reported "burst" speeds with SHARK² for isolated repeated phrases range from 60–80 WPM.

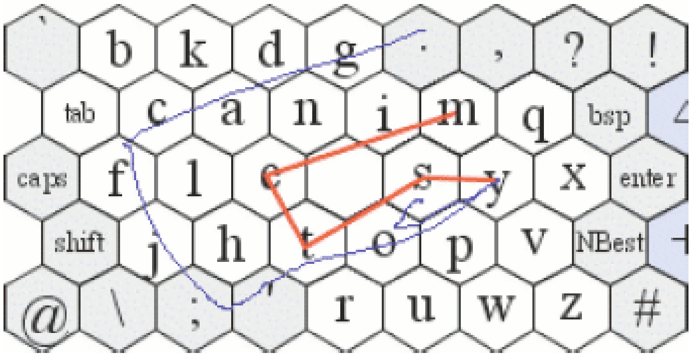


Figure 2.9. SHARK stylus keyboards allow both tapping and stroking of words. The user has stroked the pattern for the word "system," even though the stroke itself is not required to hit every letter in the word. The idealized pattern is also shown. Image taken from (Kristensson and Zhai 2004). Used with permission.

Besides SHARK, word-level unistrokes can be made in both *Cirrin* (Mankoff and Abowd 1998) and the aforementioned Quikwriting (§2.3.1). However, these methods require users to access each letter within the word being entered, so although they utilize a single stroke for each word, their strokes tend to be rather long and "swoopy," lacking any mnemonic correspondence to the Roman letters or words for which they stand.

Word prediction and completion systems are also relevant to word-level stroking. Stroke-based examples include *Ink Completion* (Denoue and Chiu 2005) and *Speech Pen* (Kurihara *et al.* 2005). However, unlike EdgeWrite's word-level stroking, these systems supply candidate words as part of a graphical list, and require visual verification and a

[3] For legal reasons, SHARK has recently been renamed *ShapeWriter*.

separate action for selection. Some studies even show that list-based word selection slows people down (Goodenough-Trepagnier *et al.* 1986, Soede and Foulds 1986). Also, having separate actions for selection breaks the fluid motor patterns that users may develop for making word-level strokes. In contrast, EdgeWrite allows words to be completed in a single non-lifting stroke (§3.5.2).

2.3.4 Displacement Joystick Input

A “displacement joystick” is one in which the joystick’s angle determines the magnitude and direction of the joystick’s output reading. Usually, the further the joystick is displaced in a given direction, the faster an object or view moves in that direction. Displacement joysticks differ from isometric joysticks, which sense force but do not move.

There have been a few other gestural text entry methods for joysticks. One is *Weegie*, a prototype method for use on X11, the popular UNIX window manager (Coleman 2001). With Weegie, a user moves a joystick to various positions (e.g. 12 o’clock) to access different characters. EdgeWrite differs from Weegie in that EdgeWrite’s strokes are similar to Roman letters, whereas Weegie’s arrangement of letters has no Roman similarity. Also, EdgeWrite requires only one stick, whereas Weegie requires two. A second method, *KeyStick*, is a displacement joystick method for use on some mobile phones (Bloor 2003, n-e-ware 2004). With KeyStick, a user moves the joystick left, right, up, or down to access menus of characters. Like Weegie, the placement of KeyStick’s characters within menus is not reminiscent of Roman forms. A third method, *myText*, is another joystick method for mobile phones (Co-operwrite Ltd. 1997). Unlike Weegie and KeyStick, myText is not menu-based but letter-like. myText recognizes gestures by looking at “unit vectors” of motion. No scientific performance results have been reported for any of these methods.

More common than gestural joystick text entry methods are various *selection-based* joystick methods. The earliest of these is probably *date stamp*, which is familiar from high-score screens from 1980s arcade games (Herz 1997). This method gets its name from a post office stamp that has rotating dials for each character. Users pull down on the joystick to “roll” the current letter “forward” from “a” to “z”, and push up on the joystick to roll it back from “z” to “a”. Overshooting “z” results in returning to “a”.

Selection keyboards are another form of selection-based joystick text entry. Selection keyboards are on-screen two-dimensional keyboards that show a selector highlight over the current letter. Users move the joystick in the four cardinal directions to move the highlight, and press a button to commit the currently selected letter. Modern video games such as *Halo* and *Brute Force* use selection keyboards for text entry during player configuration.

Wilson and Agrawala used the two thumb joysticks of an Xbox controller to manipulate two distinct selectors on separate halves of a split selection keyboard (Wilson and Agrawala 2006). They found that their dual-stick QWERTY design, at about 7.1 WPM, was significantly faster than a single-stick QWERTY layout (6.5 WPM) and a dual-stick alphabetical layout (6.4 WPM). Total error rates for dual-stick QWERTY were about 5.4%.

Displacement joysticks are also widely used on power wheelchairs. With the exception of EdgeWrite (§6.2.1), no gestural text entry systems exist for wheelchair joysticks, but there are commercial solutions for controlling a desktop mouse cursor from a power wheelchair joystick. An example is the *Mouse Driver* from Switch-It, Inc. (http://www.switchit-inc.com), a radio-controlled mouse cursor. However, text entry with this technology amounts to point-and-click or point-and-dwell over an on-screen keyboard.

Mouse cursor control using a power wheelchair joystick has been studied (Romich *et al.* 2002, LoPresti *et al.* 2004). Results show that joystick-based mousing is slow but feasible, and that proportional rate control is better than absolute mapping for text entry using an on-screen keyboard.

2.3.5 Touchpad Input

Besides EdgeWrite, few text entry techniques exist for touchpads. Two limited exceptions explored numeric text entry on touchpads. Enns and MacKenzie used a touchpad as a television remote control, demonstrating that Graffiti numbers could be used successfully to change television channels (Enns and MacKenzie 1998). Isokoski and Kaki compared two clock-face metaphors for entering numbers on touchpads, one that was essentially a single-tier marking menu and another that used two-tiers (Isokoski and Kaki 2002). Peak speeds were about the same (~18 WPM), but the two-tiered scheme was more accurate (14.6% *vs.* 10.2% total errors).

Most prior touchpad research has focused on new interaction techniques, not text entry methods. Innovative examples include using a touchpad in the non-preferred hand for manipulating maps (Hinckley *et al.* 1998), as an additional mode-switch for enabling keyboard-based mouse emulation (Rekimoto 2003), and with an underlying electronic relay to provide tactile feedback under pressure (MacKenzie and Oniszczak 1997). The latter, dubbed the *Tactile Touchpad*, was shown to be 20% faster for selection tasks than the lift-and-tap method, and 46% faster than using a touchpad button (MacKenzie and Oniszczak 1998).

Similar to EdgeWrite's plastic template, physical guides have been used on touch-sensitive surfaces to guide fingers (Buxton *et al.* 1985). Buxton and Myers used two touch-sensitive strips defined by physical edges to provide absolute positioning within a document and relative positing for scrolling (Buxton and Myers 1986).

A few researchers have placed touchpads on the *backs* of devices to enable new interaction techniques. Silfverberg *et al.* put two touchpads on the back of a portable computer for panning and zooming with one's fingertips (Silfverberg *et al.* 2003). This device is shown in Figure 2.10.

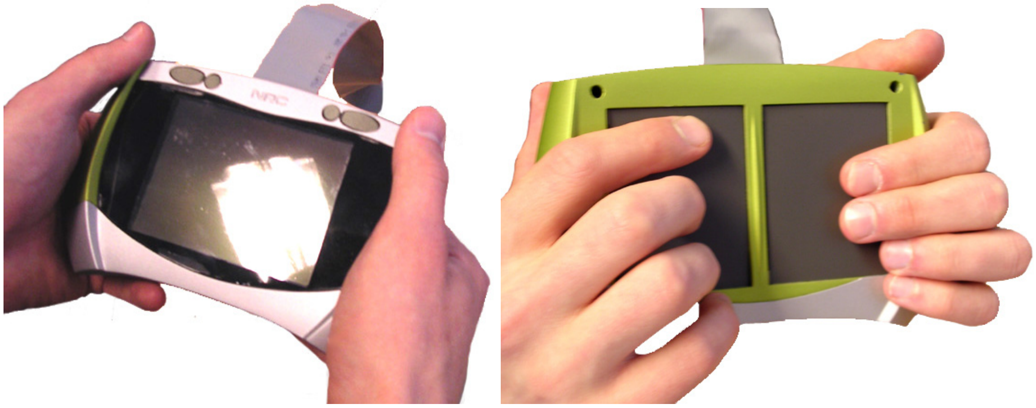


Figure 2.10. Silverberg *et al.*'s prototype. In the right image, the left touchpad is used to pan in four directions, and the right touchpad is used to zoom. Used with permission.

Schwesig *et al.* created a bendable computer that uses a touchpad on its back for 2D positioning (Schwesig *et al.* 2004). They also created a selection-based text entry method for use with the touchpad. The method, however, seemed to be only suitable for writing a few words. Hiraoka *et al.* built a touchpad into the back of a cell phone in the *Behind Touch* prototype (Hiraoka *et al.* 2003). The small pad had elevated bumps that could be used for entering text and numbers into the phone.

A new type of touch-surface is related to Four-sensor EdgeWrite (§9.4). Capacitive phone keypads are beginning to support touch-based interactions like gestures. The Motorola A668 phone has a capacitive sensing keypad (http://direct.motorola.com), as do Synaptics' *MobileTouch* technologies (http://www.synaptics.com). Rekimoto *et al.* explored a set of "preview" interaction techniques for touch-sensitive keypads in the *PreSense* prototype (Rekimoto *et al.* 2003).

2.3.6 Trackball Input

Trackballs have been used mostly as pointing devices. An early study by Epps *et al.* compared six pointing devices, including a 4 cm trackball, in target acquisition tasks (Epps *et al.* 1986). They found that the mouse and trackball were significantly faster than the other devices, but were not significantly different from each other. A follow-up study by Sperling and Tullis found that the mouse *was* faster for selection, dragging, and tracing, even among trackball users (Sperling and Tullis 1988). Accuracy differences were not significant for selection and dragging, but were significant for tracing, showing the trackball to be less accurate than the mouse. Therefore, Trackball EdgeWrite does not require precise pointing or smooth freeform gestures like those in handwriting or most unistroke methods.

An exercise related to "tracing" is "steering." Accot and Zhai studied five input devices, including trackballs, in various tunnel-steering tasks (Accot and Zhai 1999). They found that trackballs are comparable to touchpads—but both are worse than mice—and that trackballs perform best relative to other devices for short straight-line trajectories less than 250 pixels. They note that the performance of trackballs and touchpads suffers when the device must be "clutched" for travel over long distances. Accordingly, Trackball (§7.2) and Touchpad EdgeWrite (§6.2.2) both avoid the need for clutching.

Further comparisons of pointing devices by MacKenzie *et al.* show that trackballs are slower than mice and styli in pointing and dragging, and less accurate for dragging (MacKenzie *et al.* 1991). Dragging is particularly difficult with a trackball because of the confluence of the thumb and finger muscles. In a later study, MacKenzie *et al.* added that trackballs often move accidentally when clicking inside targets (MacKenzie *et al.* 2001a). Such slips can be particularly troubling for users with motor impairments (Trewin and Pain 1999, Paradise *et al.* 2005). Accordingly, Trackball EdgeWrite does not rely on buttons or require dragging or clicking. It can, in fact, be used button-free.

Hinckley and Sinclair developed the *TouchTrackball* that, when touched, caused a ToolGlass window to appear in what they called an "on-demand interface" (Hinckley and Sinclair 1999). When the trackball was released, the ToolGlass faded away.

With the exception of EdgeWrite, there have been no text entry methods developed explicitly for trackballs. This is unfortunate, since trackballs are popular alternatives for people with motor impairments and many of these users cannot type on physical keyboards. What trackball text entry *does* exist has been with an on-screen keyboard. Examples include *WiViK* (Shein *et al.* 1991) and *Visual Keyboard* (Bishop and Myers 1993). In Figure 2.11, WiViK is shown below Microsoft Notepad.

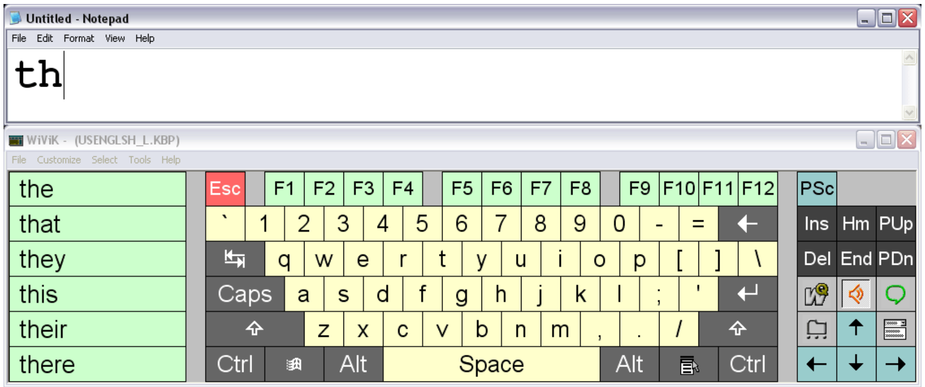


Figure 2.11. WiViK with word prediction and completion. Preferences allow letters and words to be triggered by either clicking or dwelling.

Although on-screen keyboards are easy to learn, they have many drawbacks. For instance, they exacerbate mouse travel to and from a document. They introduce a second focus-of-attention such that a user's eyes cannot remain on his or her document (MacKenzie and Soukoreff 2002b, Anson *et al.* 2005). They also require repeated target acquisitions for which trackballs are not well suited (MacKenzie *et al.* 1991, MacKenzie *et al.* 2001a). Furthermore, they are visually fatiguing, equivalent to typing in a tedious "hunt-and-peck" fashion. Finally, they consume screen real estate, considerably reducing one's visual workspace and exacerbating the need for window management. Although word prediction and completion systems may increase the speed of on-screen keyboards, they do not solve the above problems (Anson *et al.* 2005). A Trackball EdgeWrite test subject once remarked that he would prefer a gestural text entry method that he could do by feel over an on-screen keyboard, even if the on-screen keyboard were faster (§7.3.3). Fortunately, this subject was faster with Trackball EdgeWrite than with his on-screen keyboard, so he did not have to make this tradeoff.

2.3.7 Isometric Joystick Input

An "isometric joystick" is a joystick that senses force but does not move. Isometric joysticks were part of the early formal study of input devices by Card *et al.* that helped extend Fitts' law to pointing devices (Card *et al.* 1978). Isometric joysticks became popular with the advent of the *IBM TrackPoint*, the small isometric joystick embedded in ThinkPad laptops (Rutledge and Selker 1990, Barrett *et al.* 1995). The TrackPoint is slower than a conventional mouse for pointing, but faster for joint typing/pointing tasks due to reduced switching costs. It also takes up much less space than a conventional mouse, which has significant advantages for mobile computing.

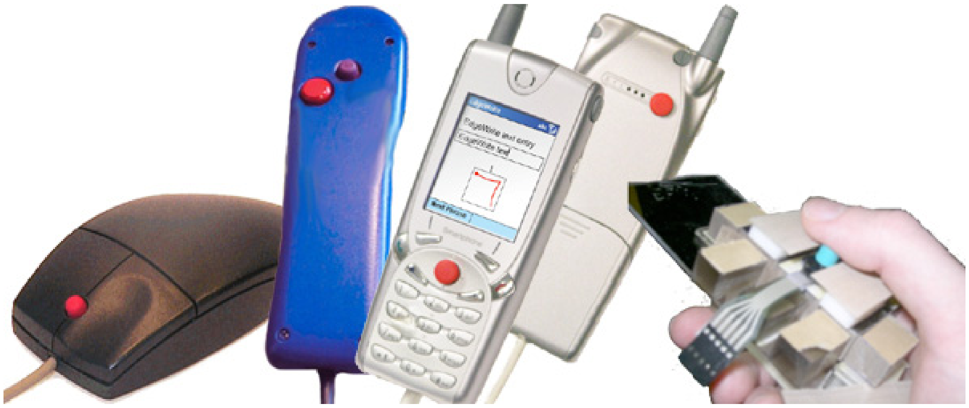


Figure 2.12. From left to right: Zhai et al.'s joystick mouse, Silfverberg et al.'s one-handed pointer, Chau et al.'s EdgeWrite phones (front and back), and Zaborowski's musical ThumbTec. Images are taken from their respective papers. Used with permission.

TrackPoint-style isometric joysticks have since been embedded in a variety of devices (Figure 2.12). Zhai *et al.* combined one with a desktop mouse to facilitate pointing with the mouse and scrolling with the finger-controlled joystick, finding this design faster and more liked than the IntelliMouse scroll wheel (Zhai *et al.* 1997). Silfverberg *et al.* embedded an isometric joystick in two handheld prototypes for pointing on information terminals (Silfverberg *et al.* 2001). They found that their two-handed prototype could achieve about 78% of a ThinkPad joystick's throughput—and their one-handed prototype about 68%—when using the thumb for control and a separate button for selection. Zaborowski invented *ThumbTec*, a thumb-controlled isometric joystick used in combination with three switches for playing music (Zaborowski 2004). Salem and Zhai put a TrackPoint in a mouthpiece for tongue-controlled pointing (Salem and Zhai 1997). Finally, Mithal and Douglas found that pointing with an isometric joystick involves many more sub-movements than with a mouse, making the former more difficult to control (Mithal and Douglas 1996).

In spite of these numerous designs utilizing isometric joysticks, there appear to be no text entry methods developed specifically for such devices. In this regard, EdgeWrite's mobile phone isometric joysticks are unique (§8.2).

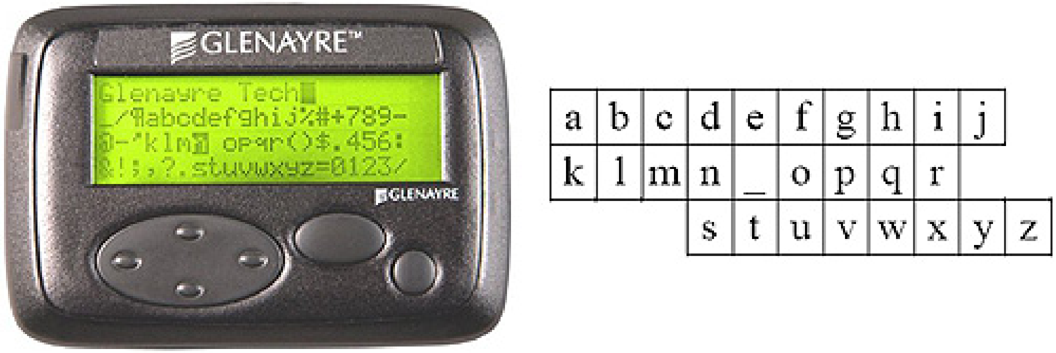
2.3.8 Mobile Keypad Input

Keypads for mobile devices may be thought to exist on a spectrum based on their number of keys (MacKenzie 2002c). There are text entry methods for 3 keys, 4 keys, 5 keys, 6 keys, 8–12 keys, and 26+ keys. Most research has been devoted to the 8–12 key group, since this is the size of most mobile phone keypads. EdgeWrite can be implemented as a gestural few-key method with four keys (§9).

Text entry with three keys has been studied in-depth (MacKenzie 2002b, MacKenzie 2002c). Three-key methods rely on two keys to move a selector left and right and a third key to select a letter. The 3-key design is a variant of the *date stamp* method used for displacement joysticks (§2.3.4). The KSPC values for most 3-key methods range from about 6.5 to 10.5 (MacKenzie 2002c). Typical entry rates are 9–10 WPM.

The gestural *UDLR* technique is intended for the four keyboard arrow keys (Evreinova *et al.* 2004). All letters consist of 3 presses of the arrow keys, so KSPC is 3.00. After a week of practice, speeds reached ~13.5 WPM, but users reportedly suffered from a great deal of confusion due to the alphabet's rather arbitrary key sequences.

Five-key text entry can be found on two-way pagers like the *Glenayre AccessLink II* shown in Figure 2.13 (MacKenzie 2002b). Five-key methods use four keys to move over a matrix of letters and a fifth key for selection (Bellman and MacKenzie 1998, MacKenzie 2002b). The KSPC for the Glenayre pager is 3.13 (MacKenzie 2002b).



a	b	c	d	e	f	g	h	i	j	
k	l	m	n	_	o	p	q	r		
			s	t	u	v	w	x	y	z

Figure 2.13. The Glenayre AccessLink II two-way pager and its alphabetic layout for use with five keys. The layout is adapted from (MacKenzie 2002b). Used with permission.

An interesting 6-key method is the GKOS keyboard (Tiainen 2000), a small commercial chording keyboard (http://gkos.com). Although no scientific results are given, the GKOS keyboard can reportedly reach 20 WPM after initial practice and 40 WPM for experts. Because it only uses six buttons, it can be integrated with mobile phones, ultra-mobile PCs, and even car steering wheels (§10.1).

Jannotti's design for *Iconic* text entry uses 10 keys of a numeric keypad, allowing users to trace letter-shapes that are somewhat reminiscent of Roman forms (Jannotti 2002). Iconic's KSPC is 2.43. It was reported to have a Fitts' law-derived peak theoretical speed of 19.8 WPM, but this was never empirically validated.

The most common text entry method in use on mobile phone keypads is *Multitap*. Karlson *et al.* report that 32.3% of surveyed mobile phone users enter text using this method, compared to just 18.3% using gestures, 17.0% using physical mini-QWERTY keyboards, and 10.5% using T9 (Karlson *et al.* 2006). With Multitap, users press each key the number of times (1–4) corresponding to the desired letter's position on the key. Thus, the letter "a" is one press of the "2" key, "b" is two presses, and "c" is three. The KSPC for Multitap is 2.03 (MacKenzie 2002b).

There have been a number of Multitap variants designed to lower KSPC and increase entry rate. Pavlovych and Stuerzlinger invented *Less-Tap*, a technique similar to Multitap that uses an optimized letter arrangement within each key, saving about a half a key press per character (Pavlovych and Stuerzlinger 2003). Its KSPC is about 1.53, which resulted in a 9.5% speed improvement over Multitap for novices. Ryu and Cruz did similarly in *LetterEase*, only instead of reassignment within keys, they reassigned letters *across* keys, finding that this resulted in an empirical KSPC of 1.32 (Ryu and Cruz 2005). However, LetterEase was slower than conventional Multitap for novices due to increased visual search. Wigdor and Balakrishnan combined Multitap with three extra chording buttons on the back of the phone (Wigdor and Balakrishnan 2004). This technique, named *ChordTap*, was about 32% faster over multiple sessions than Multitap with two-hands. Nesbat created *MessagEase* by re-lettering the mobile phone keypad and assigning each letter to 2 key presses (Nesbat 2003). A Fitts' law theoretical model showed an upper-bound entry rate of 25.9 WPM, but this speed was not empirically validated.

Predictive techniques lower KSPC by using a language model to disambiguate key sequences into the most probable word. After each key is pressed, the current key sequence is compared to a word dictionary for the most likely match. Incorrect matches can be corrected by the user with special keys that move through an *n*-best list. Commercial systems of this sort include *T9* (http://www.tegic.com), *iTap* (http://www.motorola.com/lexicus), and *eZiText* (http://www.zicorp.com). To avoid the drawbacks of using dictionaries, MacKenzie *et al.*'s *LetterWise* technique guesses the most probable letter for a given key based on a prefix of up to 3 letters already entered (MacKenzie *et al.* 2001b). Gong *et al.* added predictive next-letter highlighting to Multitap and showed that this kind of visual cueing marginally improves performance over LetterWise (Gong *et al.* 2005). Although faster than Multitap for in-vocabulary words, predictive techniques require users to constantly monitor their output to verify the results of disambiguation.

Mobile phone text entry has been modeled as well. Silfverberg *et al.* combined Fitts' law with letter digraph probabilities to create a model of expert performance for various mobile phone keypad techniques (Silfverberg *et al.* 2000). Model predictions were 25 WPM for one-thumb Multitap and 46 WPM for T9 using an index finger and two hands. However, James and Reischel empirically tested this model, finding that it overestimated expert performance in both Multitap and T9 (James and Reischel 2001). Instead of modeling experts, Pavlovych and Stuerzlinger modeled novices, including model terms for visual search, cognitive steps, and repeatedly pressing the same key (Pavlovych and Stuerzlinger 2004). Gong and Tarasewich developed optimal alphabetical and non-alphabetical letter assignments for phone keypads using genetic algorithms and exhaustive search, where reducing KSPC and disambiguation accuracy were the fitness criteria (Gong and Tarasewich 2005).

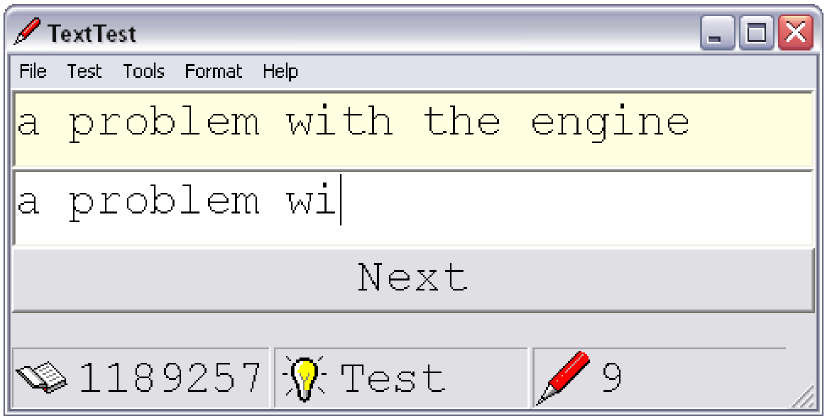
For wearable computing, *Twiddler* (http://www.handykey.com) uses a 12-button chording keypad operated with a single hand (Lyons *et al.* 2004). Because chords require substantial learning, Lyons *et al.* found that performance with Twiddler dramatically increases over sessions, from 4.3 WPM in session 1 to 26.2 WPM in session 20 (after ~6 hr. 40 min.). In this study, Twiddler caught Multitap in session 5 (~1 hr. 40 min.) at about 13 WPM. By session 20, Multitap reached about 19.8 WPM. It should be noted that Multitap was performed on a Twiddler keypad, and that subjects were allowed to use two thumbs concurrently.

Finally, a number of mobile devices now use miniature keyboards sporting 26 or more physical keys (MacKenzie and Soukoreff 2002b). The most famous of these are probably the Blackberry RIM devices (http://www.rim.com), which use mini-QWERTY keyboards. These types of keyboards were studied by Clarkson *et al.*, who compared two such keyboards in a longitudinal study (Clarkson *et al.* 2005). Results indicate that initial two-thumb typing speeds are 30–35 WPM and reach 60 WPM after 20 twenty-minute sessions. Although these speeds are impressive, drawbacks of such keyboards include the need to use two hands, larger physical footprints, and inaccessibility to people with poor motor function in their hands.

2.4 Text Entry Evaluation

Section 2.3 presented a number of relevant input devices. How input devices are evaluated is of prime importance to ensure reliable results. In chapters 4–9 of this dissertation, numerous text entry studies are conducted. Although different evaluation procedures exist (§11.1), the evaluations of EdgeWrite reported in this dissertation use the *unconstrained text entry experiments*. The unconstrained paradigm presents phrases to subjects for transcription (i.e. copying) as shown in Figure 2.14. Using the *TextTest* software, each phrase is presented after being chosen at random from a corpus of 500 phrases (MacKenzie and Soukoreff 2003). See §11.4.3 for more details.

During transcription, subjects are free to enter text as they would in any text box widget, with the exception that cursor control and mousing is not permitted. Backspace is the only means for error correction. Both correct and incorrect characters can be entered without the intrusion of “error beeps” or red-letter highlights. Subjects are encouraged to balance speed and accuracy, and to correct errors as they would when writing an email to an acquaintance—somewhere between an informal note and a formal correspondence. The benefits of the unconstrained paradigm lie mainly in its naturalness. Speed and error measurements, for example, are not thrown into question by the presence of intrusive error beeps or the halting of progress until a correct letter is entered. This section provides a brief overview of the metrics employed in these evaluations.



```
a problem with the engine
a problem wi
```

Figure 2.14. TextTest general-purpose text entry evaluation software.

2.4.1 Speed

The calculation of speed in the unconstrained paradigm is straightforward, with one caveat that is often overlooked: the numerator must be the length of the transcribed string *minus one* (MacKenzie 2002a). The reason is that timing begins with the entry of the first character and ends with the entry of the final character. Even in studies where subjects press ENTER to advance to the next phrase, the final time is the timestamp of the last entered letter.

Borrowing MacKenzie's example:

```
the quick brown fox jumps over the lazy dog (43 characters)
^                                         ^
t = 0 seconds                             t = 20 seconds
```

The proper WPM calculation is thus:

$$wpm = \frac{43-1\,\text{characters}}{20\,\text{seconds}} \times \frac{60\,\text{seconds}}{1\,\text{minute}} \times \frac{1\,\text{word}}{5\,\text{characters}} \qquad (2.2)$$

The result for this example is 25.2 WPM. Note that Equation 2.2 uses the standard definition of 5 characters per word.

2.4.2 MSD and KSPC

In the unconstrained testing paradigm, speed calculations are straightforward but error calculations are not. There are two types of errors: *uncorrected errors* that remain in the transcribed string, and *corrected errors* that are any characters backspaced during entry.

Soukoreff and MacKenzie wrote a paper on calculating these metrics (Soukoreff and MacKenzie 2001). Building on previous work (Damerau 1964, Levenshtein 1965, Wagner and Fischer 1974, Landraud *et al.* 1989), they proposed the *minimum string distance* (MSD) statistic for calculating uncorrected errors. Using the MSD statistic, we can compare the presented string to the transcribed string and arrive at the number of Morgan editing operations (Morgan 1970)—*insertions*, *omissions*, and *substitutions*—required to turn one string into the other. This gives us a count of uncorrected errors. Using the MSD algorithm (Soukoreff and MacKenzie 2001), our error rate formula is:

$$MSD\ error\ rate = \frac{MSD(P,T)}{\max(|P|,|T|)} \times 100\% \tag{2.3}$$

In Equation 2.3, P is the presented string and T is the transcribed string. The denominator is the greater length of either P or T. Note that this formula does not say *which* characters are erroneous, just the number of errors. To find out which characters are in error, we would need a character-level error analysis (§11).

For corrected errors, we can use the keystrokes per character (KSPC) dependent measure.[4] This measure is a simple ratio of the number of entered characters to the number of characters in the final transcribed string.

$$KSPC = \frac{Keystrokes}{|T|} \tag{2.4}$$

As an example, consider the following input:

```
Keystrokes:  tw←he qvi←←uick brx←owm←n (25 keystrokes)
Transcribed: the quick brown           (15 characters)
```

The KSPC error metric calculation is therefore 25 / 15 = 1.67.

[4] This is not the same as the KSPC characteristic measure (MacKenzie 2002b), although it is related. Characteristic KSPC is a theoretical model of keystrokes per character. In contrast, a dependent KSPC result is based on how accurately subjects perform during a text entry exercise.

2.4.3 Uncorrected, Corrected, and Total Errors

Soukoreff and MacKenzie unified their MSD and KSPC error metrics in a later paper (Soukoreff and MacKenzie 2003). They did this by classifying each character in a text input stream as belonging to one of four categories:

Correct (C)	All correct characters in the transcribed text.
Incorrect-Not-Fixed (INF)	All incorrect characters in the transcribed text.
Incorrect-Fixed (IF)	All characters backspaced during entry.
Fixes (F)	All backspaces.

The separation of *C* and *INF* can be accomplished using the MSD algorithm.[5] The *IF* and *F* classes are counted by scanning the keystroke input stream. We can now compute uncorrected and corrected errors and combine them into a unified *total error rate*.

$$uncorrected\ error\ rate = \frac{INF}{C + INF + IF} \times 100\% \tag{2.5}$$

$$corrected\ error\ rate = \frac{IF}{C + INF + IF} \times 100\% \tag{2.6}$$

$$total\ error\ rate = \frac{INF + IF}{C + INF + IF} \times 100\% \tag{2.7}$$

These are the formulae used to report error results throughout most of this dissertation. Readers are directed to prior work for more details (Soukoreff and MacKenzie 2003).

It should be noted that the total error rate is a somewhat arbitrary combination of uncorrected and corrected errors. The reason lies in the fundamental difference between uncorrected and corrected errors. Uncorrected errors are at odds with speed: the more errors one leaves, the faster one can enter text. Corrected errors, on the other hand, are subsumed in speed, since it takes time to correct errors. A text entry method that creates and corrects many errors during entry but ultimately produces error-free text in a short amount of time is a successful method, despite a high corrected (and therefore total) error rate. Furthermore, in most text entry studies, corrected errors greatly outnumber uncorrected errors, which makes the total error rate just a reflection of the corrected error rate. The total error rate *is* useful, however, when some subjects correct most of their

[5] As it turns out, $INF = MSD(P, T)$ and $C = \max(|P|, |T|) - MSD(P, T)$.

errors while others choose not to; that is, when the corrected and uncorrected error rates are roughly equal.

An additional drawback of these metrics is that they do not differentiate between backspaced letters that were *incorrect* and backspaced letters that were *correct* (Soukoreff and MacKenzie 2004). That's because the *IF* class is just a count of backspaced letters. In order to separate *IF* into corrected-and-wrong and corrected-but-right letters, a character-level error analysis of the keystroke input stream is needed. Although a character-level analysis existed prior to the current work (MacKenzie and Soukoreff 2002a), it only considered presented and transcribed strings, not the *IF* class and input stream. Thus, a contribution of this dissertation is providing new algorithms and analyses for erased characters (§11).

Chapter 3

The EdgeWrite Design

This chapter describes EdgeWrite's design independent of any specific version. It serves as a foundation for Chapters 4–9, which present specific EdgeWrite implementations.

3.1 Background

This section describes the genesis of the EdgeWrite concept and the initial user studies that informed its design.

3.1.1 Remote Commander

In 2002, we conducted field studies which found that although some people with Muscular Dystrophy or Cerebral Palsy could not use a conventional keyboard or mouse, they could effectively negotiate the small expanse of a PDA screen (Myers *et al.* 2002). Thus, some people with motor impairments could use *Remote Commander* (Myers 2001), a program for the Palm PDA that allows a handheld to control a desktop computer (Figure 3.1). With Remote Commander, a person can move his or her stylus or finger across the small PDA screen and thereby move the mouse cursor across the larger screen of a desktop PC. Similarly, by tapping anywhere on the PDA screen, a person can actuate a mouse-click on the desktop. Remote Commander proved quite useful for some people with motor impairments, and was even featured in an issue of *Quest*, the magazine of the Muscular Dystrophy Association, for enabling a home-schooled girl to complete her homework (Stack 2001).

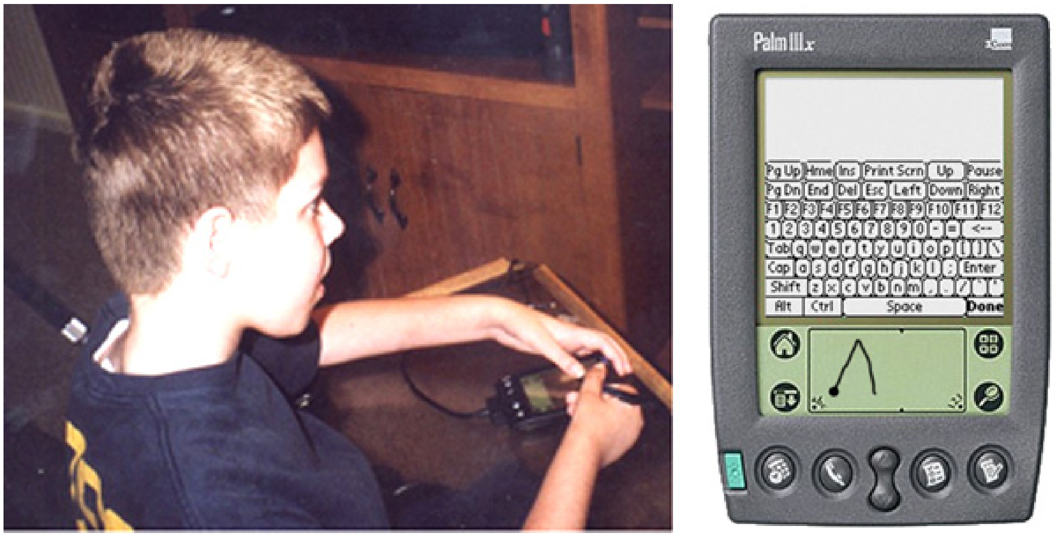

Palm IIIx
Pg Up Hme Ins Print Scrn Up Pause
Pg Dn End Del Esc Left Down Right
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12
Tab
Cap Enter
Shift
Alt Ctrl Space Done

Figure 3.1. This 12 year-old with Muscular Dystrophy is using Remote Commander attached to his desktop PC. His text entry options are a stylus keyboard or Graffiti (right), both of which are difficult. Images adapted from (Myers *et al.* 2002).

Although pointing was made easier with Remote Commander, text entry was still difficult for many users with motor impairments. The two built-in text entry methods on Palm PDAs—stylus keyboards and Graffiti—are inadequate for many people with tremor, low strength, poor coordination, or rapid fatigue. The small keys in stylus keyboards can be difficult for anyone to tap, and unconstrained stroke alphabets require substantial motor control to execute. Deviations outside the allowable range of stroke size and shape often result in misrecognitions.

3.1.2 Edge Keyboards

As a result, we began to explore methods for improving PDA text entry for people with motor impairments. Our initial design was *Edge Keyboards* as an attempt to improve stylus keyboards for people with poor motor function (Wobbrock *et al.* 2003a). The idea was to place the soft keys of the stylus keyboard around the perimeter of the PDA screen, where raised physical edges provide a backstop against which the stylus can hit. As with the Macintosh menu bar, the physical edges should make keys easier to acquire (Walker and Smelcer 1990). Furthermore, Edge Keyboards would allow users to keep their styli on the screen at all times, removing the need to hold their arms suspended.

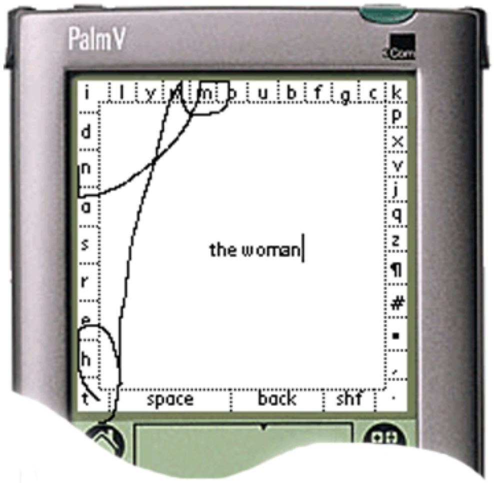


Figure 3.2. An Edge Keyboard for the Palm OS. The phrase "the woman" has been entered. Letters were arranged to minimize weighted digraph distance using a simulated annealing algorithm. Image adapted from (Wobbrock *et al.* 2003a).

Figure 3.2 shows one version of an Edge Keyboard. With this keyboard, 10 able-bodied novices could enter text at 10.2 WPM after 5 minutes of practice. This compared to 12.4 WPM for Graffiti and 22.2 WPM for stylus QWERTY. Although motor-impaired users might perform somewhat better with Edge Keyboards relative to Graffiti or stylus QWERTY, the lackluster results for able-bodied users did not warrant the further investigation of Edge Keyboards. The main problems appeared to be an increase in visual search time, too much unconstrained stylus movement, and long stylus travel distances, which negated any target acquisition benefits from physical edges. These three problems would eventually be remedied in the design of EdgeWrite.

3.1.3 Line Tracing Study

Before proceeding further, a study was conducted to discover the effects of physical edges on stylus movement. The goal was to analyze movement at a fine-grained level in order to discover what types of movement would be most successful for people with motor impairments. MacKenzie *et al.* developed path analysis techniques to compare pointing devices on the desktop (MacKenzie *et al.* 2001a). Keates *et al.* extended these measures for motor-impaired users on the desktop (Keates *et al.* 2002). The current study was the first to use these techniques to analyze stylus movement on PDAs (Wobbrock 2003).

The different line placements shown in Figure 3.3 were compared to learn how physical edges and corners affect the movements of motor-impaired users tracing lines.

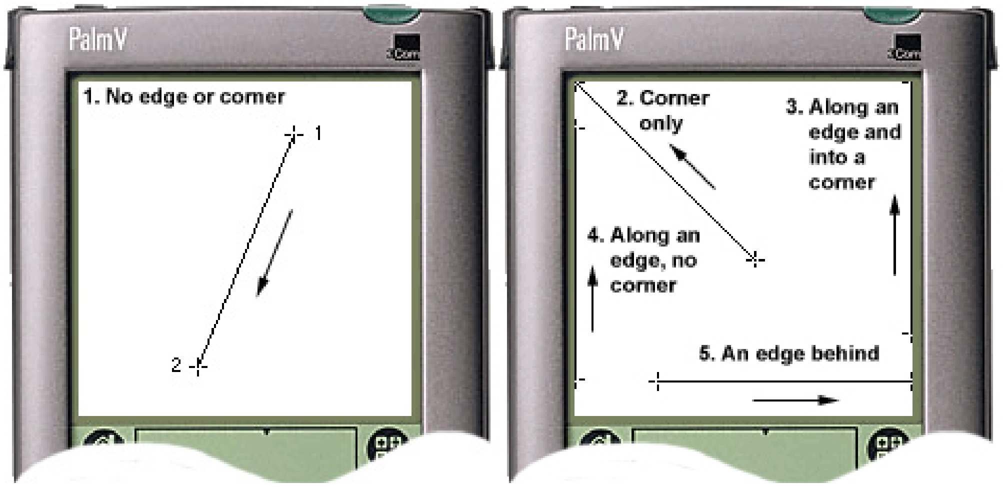


Figure 3.3. The line tracing patterns varied according to the presence or absence of edges and corners. The lines in condition 1 are presented at various angles according to the ISO 9241-9 standard. Images adapted from (Wobbrock 2003).

Three subjects, two with Cerebral Palsy and one with Muscular Dystrophy, were presented with a series of line-tracing tasks on a standard Palm V*x* screen measuring 160×160 pixels. Five line-types varied according to placement with respect to edges and corners: (1) circular in the ISO 9241-9 standard pattern (Douglas *et al.* 1999); (2) at three different angles into each of the upper corners; (3) along an edge and terminating in a corner; (4) along an edge but terminating prior to a corner; (5) not along an edge but orthogonal to and terminating on an edge. These numbers correspond to the numbered lines in Figure 3.3.

MacKenzie *et al.*'s path analyses provide a detailed account of what happens during movement (MacKenzie *et al.* 2001a). A Java program performed these analyses on the movement data obtained in the study. Two new measures were also added to the analyses. These were *Start Error* (SE) and *End Error* (EE), which were the pen-down and pen-up distances from the start and end of the line segment, respectively (Wobbrock 2003).

Figure 3.4 shows the results of the study. Movement along an edge (line types 3 and 4) result in better performance at the p<.05 level than the other three types for speed (MT), directional changes (MDC, ODC), and deviation from the task line (TAC, ME, MO, MV). In addition, type 3 is significantly more accurate to the end point than all other line types (p<.05). The only thing for which type 3 was *not* clearly better was for start error (SE), probably because there was no corner involved at type 3's starting point, only at type 3's ending point. Thus, it was clear that edge-guided lines can be traced more quickly, stably, consistently, and accurately than other line types.

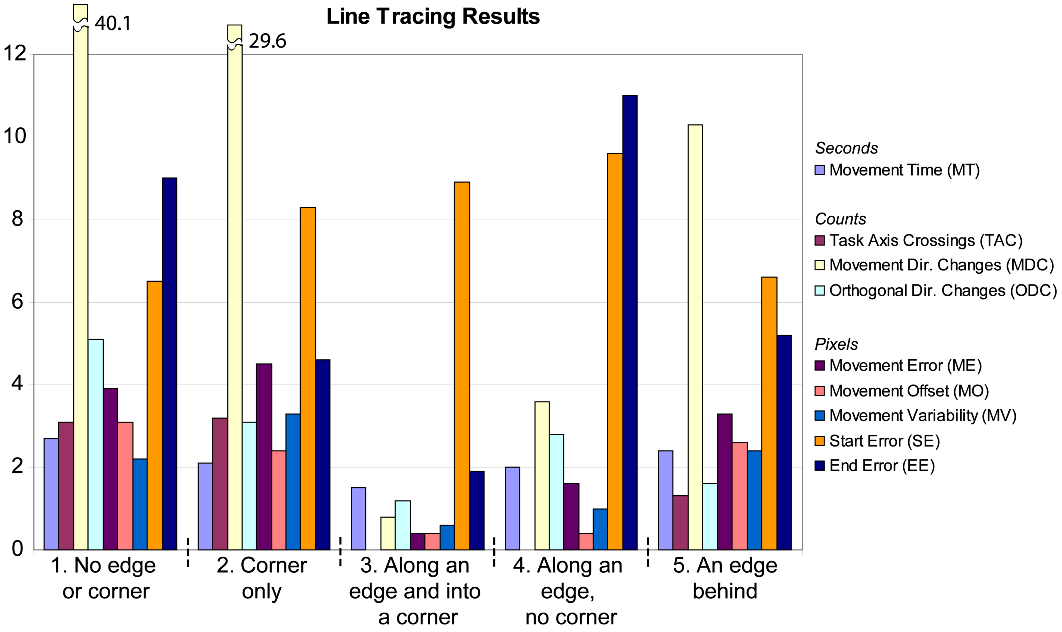


Figure 3.4. Results from the line tracing study. Lower is better for all measures. Moving from left-to-right within each line type moves correspondingly down the key at right.

The results of this study confirmed that edges are highly beneficial for stability, accuracy, and speed in straight-line gestures. The study also verifies that some motor-impaired subjects can exert enough pressure against an edge to enjoy these benefits. Motion along an edge and into a corner (line type 3) exhibited the best performance. The challenge now was to leverage these findings in the creation of a new text entry method.

3.1.4 Genesis of EdgeWrite

From the exploration of Edge Keyboards (§3.1.2), it was clear that visual search, unconstrained stylus motion, and long stylus travel distances should be avoided. From the line tracing study (§3.1.3), it was clear that motion along an edge and into a corner was the fastest, most accurate, and most stable (i.e. least "wiggly"). These findings culminated in the design of the first EdgeWrite prototype, a version for use with a stylus on a Palm PDA. Over time, this design would evolve into the technology described in the rest of this chapter.

3.2 Core Concepts

More than any other aspect, *four corners* define the EdgeWrite technique. All versions of EdgeWrite utilize four distinct corners arranged in a square (Figure 3.5). Whether the square is instantiated physically, like with a plastic template for Stylus EdgeWrite (§4), or virtually, like with an on-screen window for Trackball EdgeWrite (§7), the square defines the space in which all EdgeWrite strokes occur.

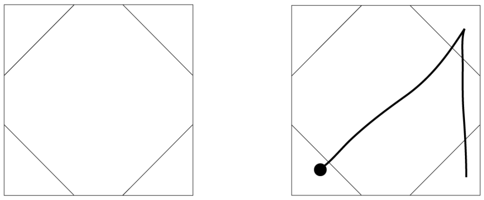


Figure 3.5. A conceptual EdgeWrite square with four corner regions. All EdgeWrite input takes place within its square. The letter "a" is shown as it might be written using a stylus. The heavy dot marks the beginning of the stroke.

EdgeWrite strokes are made along physical edges and into corners, which provide stability during the stroking process. In "virtual" versions where physical edges are not present, goal crossing is used and corners are indicated by vectors. In both physical and virtual versions of EdgeWrite, it is the corners that determine the strokes being made, not the overall path of motion—that is, not the full point trace.

Since strokes are defined by their sequence of corners, path-based input devices like styli, touchpads, and joysticks gain tolerance to wiggle due to tremor or instability. For the versions that use trackballs and isometric joysticks, the use of corners allows "messy" underlying mouse movement to be conveyed as idealized line segments that run cleanly between corners. This idealization is particularly easy because no EdgeWrite characters require that the same corner be entered twice in a row. Both cases are shown in Figure 3.6.

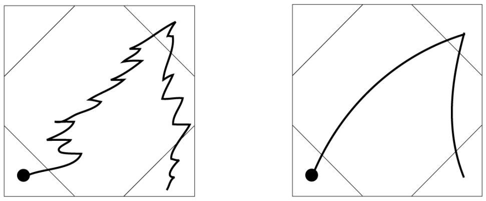


Figure 3.6. Defining strokes by their corner sequences allows wiggle to be tolerated (left) and underlying mouse movement to be portrayed as clean arcs between corners (right).

As an EdgeWrite stroke unfolds, it is effectively "tokenized" into its corner sequence in real-time. This tokenization provides a number of benefits not present in former unistroke alphabets. For instance, EdgeWrite can incrementally recognize a stroke as it is made (§3.4.1), only having to submit the current corner sequence to the recognizer when a new corner is entered. This allows EdgeWrite to display recognition results *before* they occur. Tokenization also allows EdgeWrite to provide certain advanced features like non-recognition retry and slip detection (§3.4).

In essence, EdgeWrite's corners provide intra-stroke segmentation points, allowing its gestures to be easily decomposed into component parts. Because of this, EdgeWrite only needs to receive those component parts as input. This enables EdgeWrite to be instantiated on devices that do not have continuous sensing surfaces. In fact, four "binary sensors" are all that is required for EdgeWrite input, since each sensor can correspond directly to a corner (§9).

The use of four corners also provides absolute referents that can differentiate similar strokes. For example, a vertical down-stroke on the left side of the square enters an "i", but on the right side, a numeral "1" (Figure 3.7). These "implicit spatial modes" help reduce the number of *actual* modes required for different types of input.

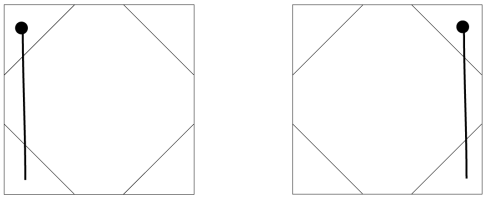


Figure 3.7. Using four corners means that the same stroke made in different parts of the square can be differentiated. The left stroke produces an "i"; the right, a numeral "1".

Using corners allows EdgeWrite characters to be efficiently defined, stored, and recognized. Each character is encoded as a 64-bit hexadecimal integer in which each corner is allocated 4 bits (Figure 3.8). This simple encoding scheme means that no training examples, glyphs, gesture prototypes, or pattern matchers are necessary for character recognition. It also means that there are no recognition collisions, as each stroke is unambiguous. This solves a common problem with gesture recognition systems where gestures are not sufficiently distinct from one another (Long *et al.* 1999).

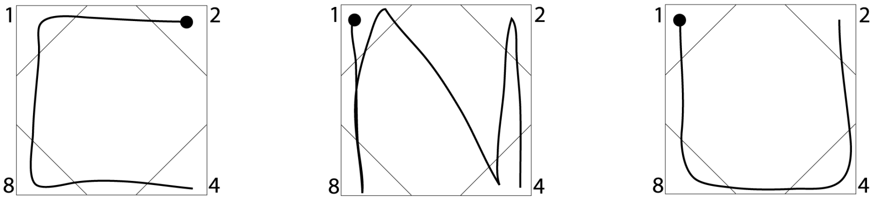


Figure 3.8. These strokes for "c", "m", and "u" can be defined with hexadecimal integers 0x2184, 0x181424, and 0x1842, respectively. Corners are encoded in 4 bits.

Another virtue of using corners is that end-users could define their own gestures using exactly *one* "training" example. This would allow for easy defining of special strokes for launching applications, entering boilerplate text, performing sequences of actions, and so forth. Assuming the example stroke is performed as the user intends, the system would have all the information it needs to recognize the stroke in the future. If the gesture were already assigned, the user could be prompted to resolve conflicts. Although this feature is not implemented in EdgeWrite, it is a straightforward extension based on EdgeWrite's reliance on corner sequences and an option for future work.

Corners also provide *targets* within the stroke-input space. This allows gestures and selections to occur together in interesting ways. For example, words can be assigned to corners as strokes are being made to provide word predictions and completions (§3.5). These words can be selected by selecting their respective corners.

The choice of a square input area is not arbitrary. The use of a square guarantees that edge-following motions will be in the four cardinal directions: up, down, left, and right. These have been shown to be the easiest directions to move in a gestural alphabet, even without an edge (Isokoski and Raisamo 2000). A more complicated polygon would result in more vertices and more possibilities for the next corner. This could be cognitively overwhelming in the case of many proximate vertices. Higher-order polygons would also result in more oblique angles at the vertices, which may cause the stylus to "slip out" during movement. While an acute angle pockets a stylus best, a square's 90° corners are adequate to catch a fast-moving stylus. A lower-order polygon, a triangle, would too severely limit the number of possible character forms and reduce the extent to which the forms resemble their handwritten counterparts. Section 3.3.2 analyzes some theoretical properties of using four corners in a square.

When a character-stroke is finished, the user *segments* the stroke in a manner dictated by the input device. With a stylus, the segmentation occurs when the pen is lifted. With a isotonic joystick, it occurs when the joystick snaps-to-center. With a trackball, it occurs when force (i.e. motion) ceases on the trackball. After segmentation, the stroke is processed and the recognized character, if any, is produced.

3.3 The EdgeWrite Alphabet

The EdgeWrite alphabet is consistent across all versions of the technique. This section describes the alphabet, how it was designed, and how its immediate usability compares to Graffiti (MacKenzie and Zhang 1997).

3.3.1 Alphabet Design

The design of an alphabet will depend on the properties most championed by its designers. For Unistrokes, this property was efficiency (Goldberg and Richardson 1993). For MDITIM, it was maintaining unambiguous prefix codes while using only the four cardinal directions (Isokoski and Raisamo 2000). For Graffiti, it was stroke differentiability and similarity to Roman letters (Blickenstorfer 1995, MacKenzie and Zhang 1997, Fleetwood *et al.* 2002).

Users with motor impairments do not have the luxury of enduring long practice sessions in which they can accomplish many trials. In addition, the use of mobile devices is intermittent. It is therefore crucial for EdgeWrite to be highly guessable and quickly learnable. The properties championed in its design are *Roman similarity* and *variation accommodation*.

Roman similarity was initially secured by observing how people write on paper and mimicking that as much as possible. Numerous user studies helped to refine these initial characters. Later, a formal guessability and immediate usability study improved the alphabet further (§3.3.3), after which a few minor improvements resulted in the character set shown in Figure 3.9.

Variation accommodation refers to the fact that people write letters differently, and the EdgeWrite alphabet should encompass that variation rather than force users to change. For example, some people commonly make their “n”’s from the top-left going down, while others make them from the bottom-left going up. EdgeWrite defines both strokes as “n”, as well as many others. In general, there are multiple forms of each letter, including many without diagonals, since these are the segments not supported by an edge.

Figure 3.9 shows the primary character chart for EdgeWrite. It is called "primary" because only one stroke is shown for each character. For a full chart, see Appendix A.

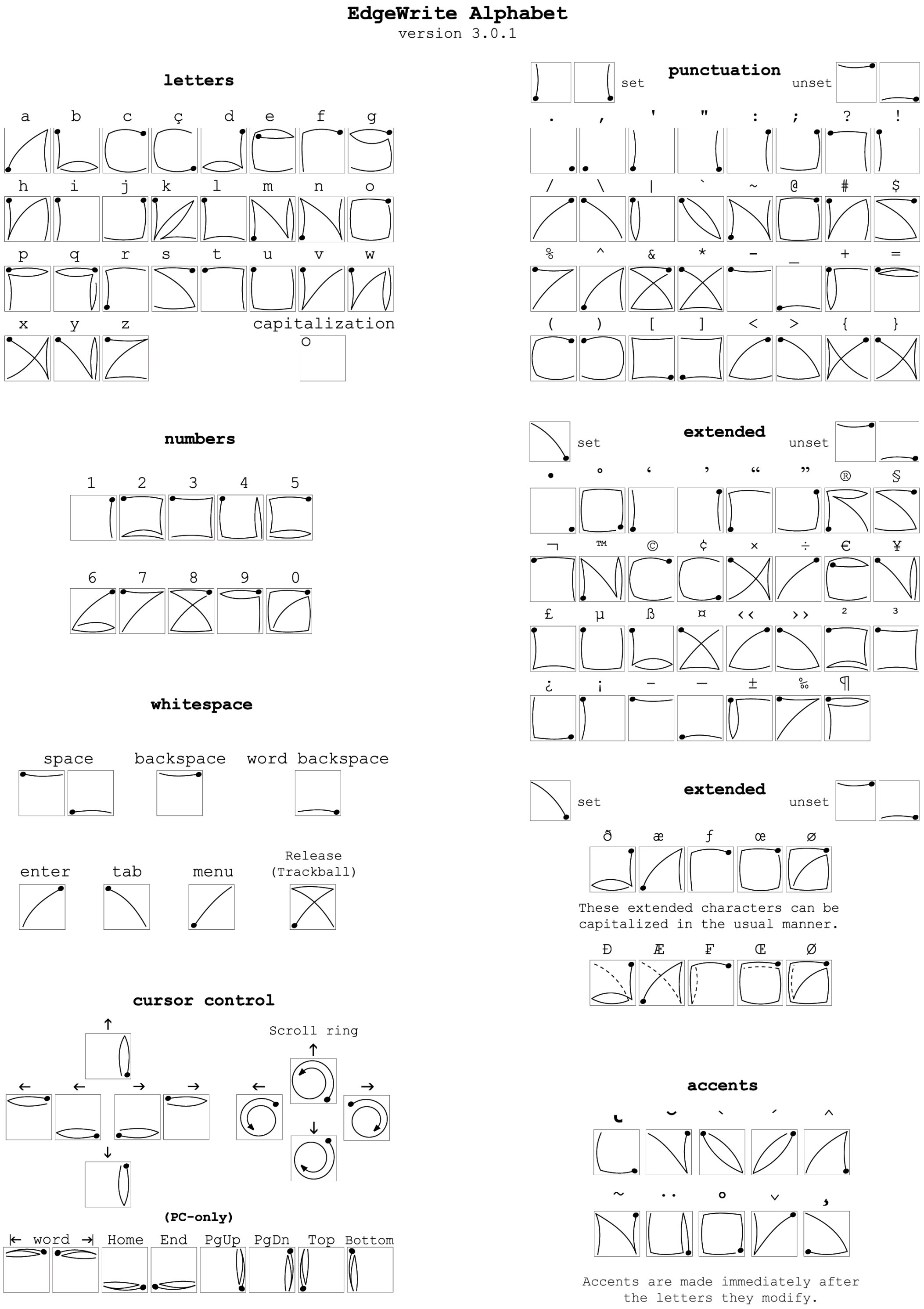


Figure 3.9. The EdgeWrite character chart. This version shows only primary forms.

Although many of the EdgeWrite characters look vaguely like their hand-printed counterparts, their mnemonic power comes from the way they feel when being written. One subject noted this when, after entering 20 phrases with Stylus EdgeWrite, he said, “I don’t remember any of the pictures in my mind, but I still feel them in my hand” (Wobbrock *et al.* 2003b).

EdgeWrite was designed to avoid some problematic aspects of Graffiti. For example, EdgeWrite tolerates the presence or absence of initial down strokes on “b,” “d,” “m,” “n,” “p,” and “r.” EdgeWrite also includes different forms of “k” to avoid the “k”-“x” confusion familiar to Graffiti users (MacKenzie and Zhang 1997).

One feature of EdgeWrite letters is that would-be loops—e.g. the bases of lowercase letters “b”, “d”, and “g”—are often “collapsed” along an edge, making for a double-pass over the same edge (Figure 3.10). Similarly, for wide letters like “m”, “w”, and “y”, collapsing along an edge is useful when the gesture has already moved across the square but is not yet complete (Figure 3.11).

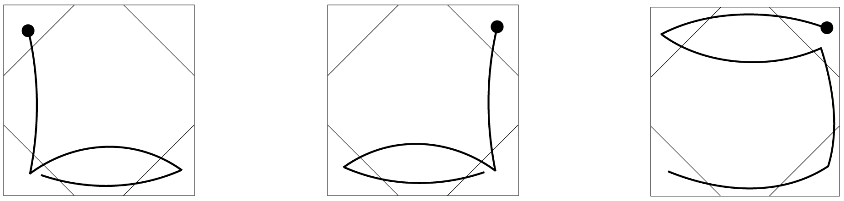


Figure 3.10. EdgeWrite letters “b”, “d”, and “g” contain loops that become “collapsed” along one edge.

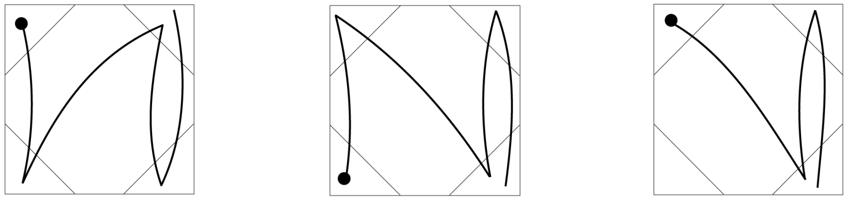


Figure 3.11. EdgeWrite letters “w”, “m”, and “y” all contain a double-pass.

Although double-passes “feel right” when writing, a challenge is portraying these strokes on paper. If double-passes are drawn literally, then the result is merely a single line. Figure 3.12 shows two early attempts at portraying characters, neither of which did a very good job at showing double-passes.

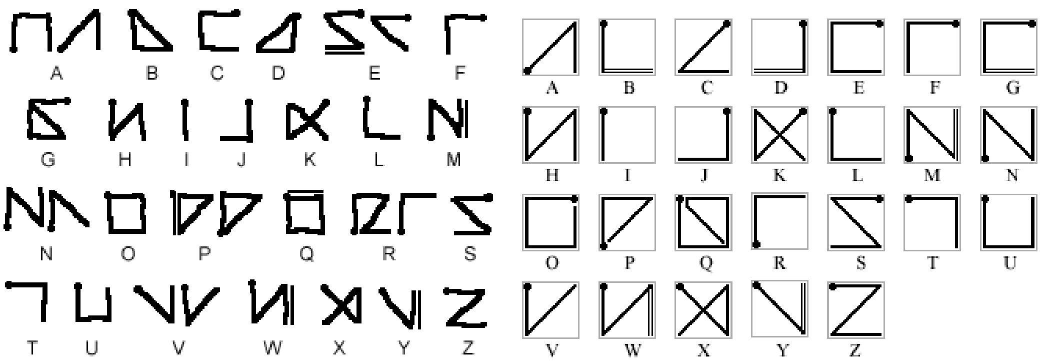


Figure 3.12. Two early attempts at portraying EdgeWrite characters. Neither image does a good job at indicating the order of corners. Adapted from (Wobbrock *et al.* 2003a).

A useful suggestion by John Zimmerman, a professor of visual design, was to *arc* the stroke paths to their intended corners. The arcs make it possible to clearly depict a double-pass over the same edge. They also often "point toward" the corners they intend. Thus, the EdgeWrite character depictions shown in Figure 3.9 are *representational*, not literal. In EdgeWrite, all motion is ideally in straight lines between corners.

As in Graffiti, some EdgeWrite gestures resemble lowercase letters, while others resemble uppercase letters. All gestures produce lowercase letters unless the *capitalization suffix stroke* is appended to the gesture. The suffix stroke is simply a motion to the top-left corner—think "up" to "make it big"—after the regular stroke is made but before segmentation. This modeless design was possible because no EdgeWrite letters finish in the top-left corner. In user studies, subjects had no trouble with this method of capitalization. Figure 3.13 shows three examples of capitalization.

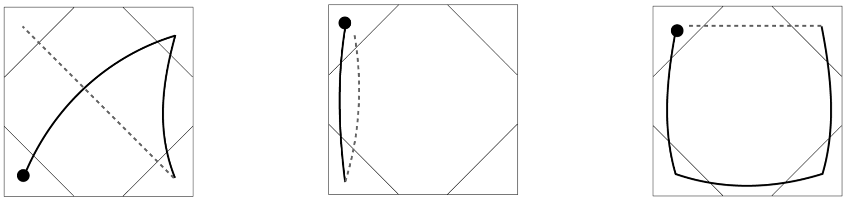


Figure 3.13. A capitalized "a", "i", and "u". The dashed lines represent a suffix stroke made to the top-left before segmenting.

EdgeWrite defines numerals in the same *alphanumeric* mode as letters. Thus, letters and numbers co-exist without the need for separate modes. However, EdgeWrite does provide explicit modes for *punctuation* and *extended characters*. These modes are entered with a single efficient stroke after which alternate characters become available

(Figure 3.14). Modes are not “sticky” like CAPS LOCK on a QWERTY keyboard, so after a character is made, the mode turns off. Backspace also will turn off a mode after the mode has been set.

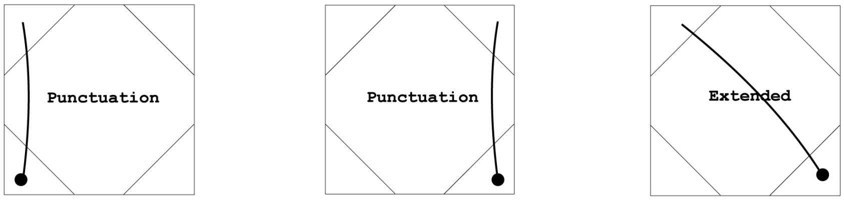


Figure 3.14. EdgeWrite has two alternative strokes for setting punctuation mode and one stroke for setting extended mode.

Accented letters can be entered in EdgeWrite using two successive strokes. The first stroke determines the letter to be accented, and the second stroke applies the accent. Available marks are *grave* (à), *acute* (á), *circumflex* (â), *tilde* (ã), *diaeresis* (ä), *ring* (å), *dot* (ż), *caron* (ǎ), *breve* (ă), *cedilla* (ç), and *ogonek* (ą). Of course, not all accents can be applied to all letters.

There are two types of backspace in EdgeWrite. A stroke from right-to-left along the *top* of the square erases the most recent character. The same stroke along the *bottom* of the square erases the most recent word. It also undoes a selected word completion when this feature is being used (§3.5). Either stroke can also be used to unset an active mode.

Finally, text cursor control is also available in EdgeWrite. There are both discrete and continuous ways to move the text cursor. Discrete ways include strokes to move the cursor one character or word forward or backward, one line up or down, PAGE UP, PAGE DOWN, to the HOME or END of the current line, or to the TOP or BOTTOM of the entire document. Continuous cursor control is available by using the EdgeWrite square as a *cursor scroll ring*. The initial direction in which one starts moving relative to the right or top sides of the EdgeWrite square determines the scrolling direction. Scrolling begins when the stroke completes its second full circle. Thereafter, each corner entered is another character or line in the scroll. The user can reverse the direction of the text entry cursor at any time without restarting or segmenting by simply changing his or her direction around the scroll ring. Figure 3.15 shows the strokes available for cursor control.

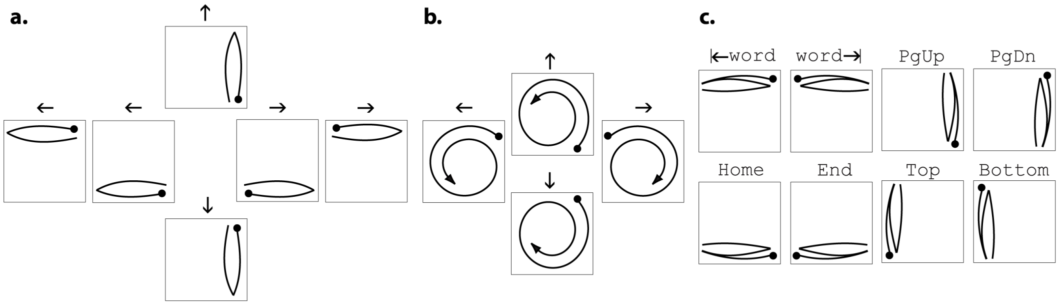


Figure 3.15. The various cursor control strokes include (a) discrete arrow keys, (b) a continuous scroll ring, and (c) discrete word, line, and document jumps.

3.3.2 Theoretical Properties

If we define a "segment" to be a straight line between two vertices (corners), then for gestures made inside a closed polygon with *v* vertices, the number of possible character forms *N* using *s* segments is given by Equation 3.1:

$$N = \sum_{i=0}^{s} v \cdot (v-1)^{i} \tag{3.1}$$

This formula treats a tap at a vertex as a legal stroke, and assumes that the same corner is never used twice in a row. For a square, $v = 4$. If $s = 0$, meaning we use no segments, we have $N = 4$ available forms: a tap in each of the corners. With 1 segment, we get $N = 16$ possible forms ($4 + 4\times3^1$); with 2 segments, we get $N = 52$ ($4 + 4\times3^1 + 4\times3^2$); with 3 segments, we get $N = 160$; with 4 segments, we get $N = 484$; and with 5 segments, we get $N = 1456$. Thus, there are many forms to choose from with relatively few segments.

There are 124 unique characters (excluding accents and capitals) in the primary EdgeWrite character set pictured in Figure 3.9. The full character set contains 314 character strokes (Appendix A). The number of segments between corners for different character subsets is shown in Table 3.1. Note that "letters" includes SPACE, which represents about 18.7% of written English (Soukoreff and MacKenzie 1995).

Mode	Characters	Mean No. of Segments	Weighted Mean
Primary letters	27	2.85	2.52
Primary alphanumeric	37	2.97	2.53
Primary punctuation	33	2.12	
Primary extended	36	2.53	
Primary characters	106	2.56	
All letters	117	3.97	2.78
All alphanumeric	145	3.94	2.79
All punctuation	49	2.59	
All extended	100	3.62	
All characters	294	3.62	

Table 3.1. Segment counts in different EdgeWrite modes. Accents, mode setters, capitalizations, cursor controls, backspaces, tab, enter, menu, and "ç" are excluded. For alphanumeric characters, a weighted mean based on character frequency is shown.

From Table 3.1, we can see that the average primary EdgeWrite letter weighted by letter frequency has 2.52 segments in it, excluding capitalization. The unweighted average number of segments per character for the entire EdgeWrite character set is 3.62.

Isokoski defined *unistroke complexity* as the minimal number of straight-line segments required to adequately specify character forms (Isokoski 2001). Since EdgeWrite characters are already composed of straight lines, this measure can be used without abstraction. The complexities for different alphabets are shown in Table 3.2. Note that this complexity measure uses only letters and SPACE and is weighted by letter frequency. The complexity of "e," for example, has a much greater effect on an alphabet's overall complexity than the complexity of "q". One can see that although EdgeWrite letters are designed to be similar to Roman letters, they rank second in efficiency. This comparison is only suggestive, however, because Isokoski's measure only considers the *number* of segments per letter, not segment length. Also, for curved letters in alphabets other than EdgeWrite, the measure can be rather arbitrary.

Alphabet	Unistroke Complexity
Unistrokes	1.40
EdgeWrite	2.52, 2.31
Graffiti	2.54
Roman hand-printing	2.76
MDITIM	3.06

Table 3.2. The line-segment calculations are similar to Isokoski's unistroke complexity measure, shown here for various alphabets (Isokoski 2001).

The lesser of the two EdgeWrite values (2.31) in Table 3.2 comes as a result of using either one of two shortcut forms of "e" that EdgeWrite users often prefer. These shortcut forms' corner sequences are 0x214 and 0x284—both have only two segments in them (Figure 3.16). Since "e" is so prevalent (10.8%), this change from the more verbose but Roman-like "e" in Figure 3.9 reduces the overall complexity of EdgeWrite by 0.21 segments per stroke. Both numbers are included in Table 3.2 to illustrate the effect a single common letter can have on overall complexity.

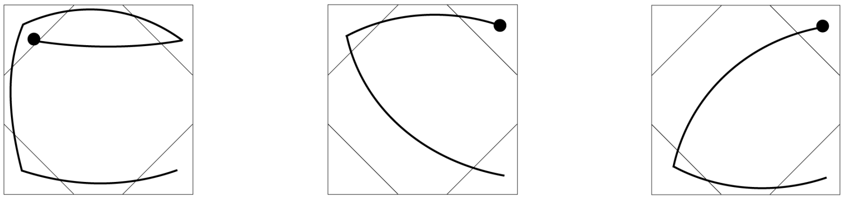


Figure 3.16. Three forms of "e", the first of which is the primary form. The other two are shortcut forms preferred by some users for their efficiency.

3.3.3 Maximizing Guessability[*]

As previously stated, the character charts shown in Figure 3.9 and Appendix A are the final versions of the EdgeWrite character set. However, the EdgeWrite alphabet underwent many fine adjustments and one thorough revision to get to this point. The fine adjustments, which usually involved changing or adding a few alternate strokes, were often done after completing various EdgeWrite user studies. The studies of Stylus EdgeWrite (§4), Joystick EdgeWrite (§5–6), and Touchpad EdgeWrite (§6) are examples. After these fine adjustments, however, a thorough revision was carried out by having *participants* design the alphabet in a procedure intended to maximize guessability. This resulted in a similar but improved alphabet that was used in studies of Trackball EdgeWrite (§7), Isometric Joystick EdgeWrite (§8), and Four-key EdgeWrite (§9).

[*] Parts of this section are adapted from (Wobbrock *et al.* 2005b).

This subsection describes the guessability maximization procedure, which is applicable beyond unistroke letters, and shows how it improved the guessability and immediate usability of EdgeWrite. In the end, the participant-designed alphabet was adjusted only a little to create the final alphabets shown in Figure 3.9 and Appendix A.

3.3.3.1 Motivation

The guessability of a system determines a great deal about its initial user experience. It is unrealistic to expect that users will have the time or desire to undergo extensive training with systems, whether by tutorial, on-line help, printed manual, or human instruction. Thus, a user's initial attempts at performing gestures, typing commands, or using buttons or menu items must be met with success despite the user's lack of knowledge of the relevant symbols. This requires high guessability.

Guessability is particularly important in symbolic input, where users enter or access *symbols* to indicate associated *referents*. Examples of symbols and referents are stylus strokes that enter ASCII characters, command-line names that execute programs, or graphical buttons that access features. In these cases, users often know what referent they desire (e.g. the letter, program, or feature they want), but they do not necessarily know what symbol to use (e.g. the corresponding stroke, command name, or graphical button).

In the past, guessability has been crucial to command-line interfaces. Prior studies show that designers often supply only one command-line term per referent (Good *et al.* 1984, Furnas *et al.* 1987). But one term, no matter how "natural," results in guessability failures of 80–90% (Furnas *et al.* 1984). A previously proposed solution is "unlimited aliasing" (Furnas *et al.* 1987), where the system makes the best guess at the intended referent in the event of an unrecognized symbol. Having multiple synonyms has also been recognized as a key to achieving high guessability in command-line interfaces (Good *et al.* 1984, Furnas *et al.* 1987).

The guessability of text labels and graphical icons has also been studied (Wiedenbeck 1999). Guessable labels and icons are important for the usability of buttons, toolbars, and menus. This dissertation's method for maximizing guessability can be applied to studies where participants devise text labels or sketch graphical icons for described features. Procedures for such studies have been delineated elsewhere (Brinck *et al.* 2001, p. 316).

High guessability is especially important when using small devices for off-desktop computing. Small devices mean contrived input schemes, limited screen real estate for help screens, and "on the go" mobile use without access to unwieldy manuals. Also, the typical intermittent use of handheld devices means that users have less time for in-use learning. Modern users expect success right from the start.

Good guessability is also crucial for assistive technologies, because people with physical disabilities often cannot endure lengthy training or practice sessions with many trials per session. Also, therapists must match an assistive technology to a user in a matter of hours, so systems must be usable from the start.

Guessability is not just limited to novices, either. Experts also need systems with high guessability. When experts must perform an uncommon action, like entering an obscure character in a unistroke alphabet (e.g. "q"), their otherwise high performance may be impeded unless the obscure symbol is guessable.

Guessability can be contrasted with immediate usability (MacKenzie and Zhang 1997). Immediate usability is the holistic evaluation of a system *after a brief period of exposure*. In contrast, guessability evaluates only the input symbols without any prior learning.

3.3.3.2 Guessability Procedure

It is possible to design a highly guessable symbol set by acquiring guesses from participants. With the same participant data, we can also evaluate the guessability of an existing symbol set. Participants are first recruited to propose symbols for specified referents within a given domain. The more participants, the more likely the resulting symbol set will be guessable to external users. The goal is to obtain a rich set of symbols from which to create the resultant symbol set.

Participants should be informed *only* of the details essential to proposing intelligent symbols. For example, if unistroke symbols are required, participants must be told what unistrokes are so that they refrain from making multi-stroke symbols. Participants should not be shown any example symbols or symbols from preexisting symbol sets. Of course, they must know the referents to which their symbols refer. Example referents are the ASCII letters to which unistrokes refer, the functions to which commands refer, or the features to which icons or text labels refer.

Participants propose a symbol for each referent in turn. Symbols are captured and coupled with their intended referents. It is important not to bias the forms of the symbols by displaying the referents. For example, if participants are proposing unistroke gestures for ASCII letters, they should not see typeset letters as prompts. Similarly, if command names are being proposed, prompts containing plausible keywords should be avoided.

It is essential for conflict resolution (below) that the captured symbols be testable for equality. Testing equality may be trivial, as in the case of keyword symbols, or more complex, as in the case of (x, y) point traces for some types of unistrokes. For more complex symbols, designers may already have software to interpret them. Human judgment can also determine equality among, for example, sketches of icons. In the case of EdgeWrite gestures, since they are wholly defined by integer corner sequences, testing for equality among gestures is trivial.

3.3.3.3 Resolving Conflicts

One might imagine that we could simply lump together all participants' proposed symbols as our resultant symbol set and trivially achieve 100% guessability for the participants used. In practice, however, this is not usually possible due to conflicts—i.e. the same symbol will have been used to indicate different referents. An example from the literature is the email command "To Dennis" (symbol) being proposed to mean "send a message to Dennis" (referent 1) and also "list messages sent to Dennis" (referent 2) (Good *et al.* 1984). Similarly, the same unistroke may be proposed for "h" and "n" (MacKenzie and Zhang 1997). Although many symbols can indicate one referent, only one referent can be indicated by any given symbol. How do we decide which referent "wins" the symbol?

Symbols are tested for equality and grouped so that identical symbols form a "conflict group." After grouping, the different referents within each group are identified and the number of referring symbols counted. Then a scoring function determines which referent within each group is assigned that group's symbol. To maximize guessability, the referent that wins the symbol is simply the one with the most proposed symbols. Equation 3.2 expresses this as a function.

$$score = \left|symbols\right| \tag{3.2}$$

For example, in 20 participants, if the same unistroke were proposed for “n”, “h”, and “a” with counts of 14, 5, and 1, respectively, the gesture would be assigned to “n”.

In general, the more conflicted the set of proposed symbols, the lower the maximized guessability of the resultant symbol set. Intuitively, high conflict means participants are using identical symbols for different referents. Designers may improve this situation by making referents more distinct, by relaxing constraints on symbolic forms, or by asking participants to resolve all conflicts within their own sets of proposed symbols before they are finished proposing.

Domain-specific considerations may be accommodated by using alternate scoring functions, although guessability may not be maximized. For example, in alphabetic entry we may wish to favor common letters over uncommon ones. Equation 3.3 is an example of an alternate scoring function that balances both letter frequency (0..1) and the number of proposed symbols.

$$score = frequency^{\frac{1}{\left|symbols\right|}} \tag{3.3}$$

3.3.3.4 Calculating Guessability

Guessability has not been previously formalized in the literature. We therefore introduce a measure of guessability for symbolic input. The guessability *G* of the resultant symbol set *S* for the captured set of proposed symbols *P* is:

$$G = \frac{\sum_{s \in S} \left|P_s\right|}{\left|P\right|} \times 100\% \tag{3.4}$$

In Equation 3.4, *P* is the set of proposed symbols for all referents, and P_s is the set of proposed symbols using symbol *s*, which is a member of the resultant symbol set *S*. For our example of “n”, “h”, and “a” above, *S* = □“n”□ and *G* = 14/20 × 100% = 70%. This means our resultant symbol set *S* was able to accommodate 70% of the symbols proposed by the participants.

3.3.3.5 Calculating Agreement

We may wish to know the agreement among symbols proposed by the participants. We therefore introduce a formalization of agreement A among symbols from our captured set P. Intuitively, agreement should be 100% when proposed symbols are identical, and ~0% when they are unique. For example, in 20 proposals for referent r, if 15/20 are of one form and 5/20 are of another, there should be higher agreement than if 15/20 are of one form, 3/20 are of another, and 2/20 are of a third. Equation 3.5 captures this:

$$A = \frac{\sum_{r \in R} \sum_{P_i \subseteq P_r} \left(\frac{|P_i|}{|P_r|} \right)^2}{|R|} \times 100\% \tag{3.5}$$

In Equation 3.5, r is a referent in the set of all referents R, P_r is the set of proposals for referent r, and P_i is a subset of identical symbols from P_r. The range of Equation 3.5 is $1/|P_r| \times 100\% \leq A \leq 100\%$. The lower bound is non-zero because even when all proposals disagree, each one trivially agrees with itself. For our example of r, $[(15/20)^2 + (5/20)^2] / 1 \times 100\% = 62.5\%$ and $[(15/20)^2 + (3/20)^2 + (2/20)^2] / 1 \times 100\% = 59.5\%$.

3.3.3.6 Existing Symbol Sets

One may also use the same participant data to evaluate the guessability of an existing symbol set S. Where a proposed symbol $p \in P$ used an existing symbol $s \in S$ that was correctly intended for s's referent r, the proposal p is assigned to s. These proposed symbols (i.e. guesses) accumulate to form P_s, the set of proposals using symbol s. Then Equation 3.4 is applied, giving the percentage coverage of the proposed symbols P by the symbols in the existing symbol set S. If all proposed symbols P are covered by S, the guessability G is 100%.

3.3.3.7 The Guessability of EdgeWrite

To put this procedure and its measures through their paces, the guessability of an earlier version of the EdgeWrite alphabet was measured. We had iterated extensively on this version of the alphabet after running multiple EdgeWrite studies using it, but we had not designed it with the direct involvement of any participants. Using this guessability procedure gave us the opportunity to formally assess its guessability and improve it. Intuitively, one might not expect EdgeWrite to have high guessability since its letters are made along the edges and into the corners of an area bounded by a square.

Twenty participants, mostly staff and students from Carnegie Mellon University, served as paid volunteers. None had prior experience with EdgeWrite or Graffiti. Participants were told they would be making unistroke gestures on a touchpad to indicate letters for a new alphabet. The unistroke concept was explained to prevent multi-stroke symbols. The importance of the four corners of the square input area was also explained. No other constraints were in place and no examples were shown.

Participants were verbally prompted to enter each letter of the alphabet (*a*–*z*) and each number (0–9) by a Visual C# program that also recorded their gestures (Figure 3.17). An audio prompt was used instead of a visual prompt to avoid biasing participants by showing typeset letters. Participants were free to remake their symbols as often as they liked, but once a symbol was committed for a character, it could not be changed. To increase the variety of proposed symbols, participants were required to resolve conflicts among their *own* symbols before they were finished. Thus, each participant contributed 36 unique symbols, for $|P| = 20 \times 36 = 720$ proposed symbols in all.

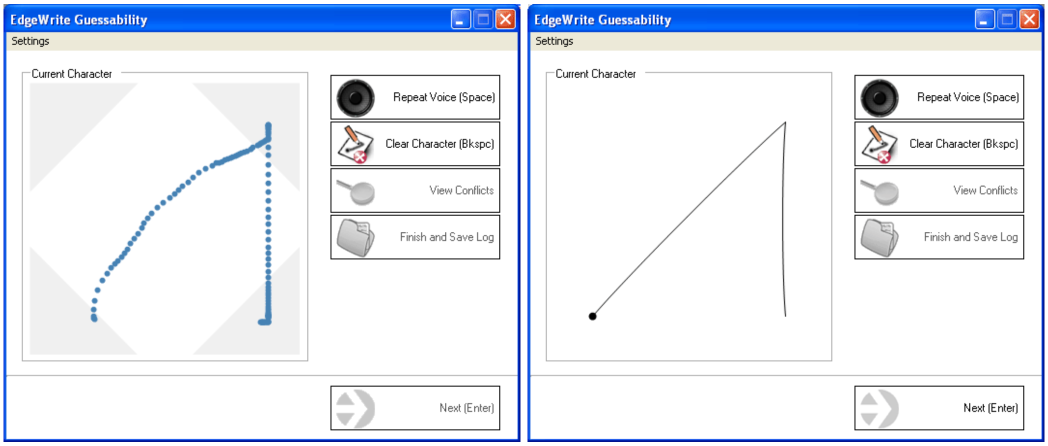


Figure 3.17. The guessability test software captured traces (left) and then displayed them in their idealized forms (right). It prompted users using text-to-speech only.

Since corner sequences fully define an EdgeWrite gesture, the 720 symbols could be grouped easily by identical corner sequences in preparation for conflict resolution. The agreement of P was A = 34.9%, meaning about a third of the proposed symbols for a given referent agreed. After conflict resolution using the maximization scoring function (Equation 3.2), the new symbol set S' accommodated 577 of 720 proposed symbols, for G' = 80.1%. The earlier EdgeWrite symbol set S was then evaluated for the proposed symbols P. It accommodated only 367 of 720, for G = 51.0%.

3.3.3.8 The Immediate Usability of EdgeWrite

In order to validate the improvement from *S* to *S′*, we replicated a prior study of the immediate usability of Graffiti (MacKenzie and Zhang 1997). Recall that *S′* was assembled *only* from the proposed symbols of participants without designer intervention, and *S* was created by designers over many prior studies. Indeed, we were skeptical that an amalgam of uninformed participant symbols could actually be more usable than the product of many hours' design work. We also were dubious that either alphabet would approach the immediate usability of Graffiti, since Graffiti had been shown to be "very respectable" in this manner (MacKenzie and Zhang 1997).

Twenty new participants served as paid volunteers. None of them had prior experience with Graffiti or EdgeWrite. The unistroke concept and importance of corner sequences were described to them. The same computer apparatus and touchpad were used as before.

As in the study of Graffiti (MacKenzie and Zhang 1997), participants entered the alphabet (*a*-*z*) five times. This occurred twice in two separate phases of testing: first, after 1 minute of studying a 26-letter EdgeWrite character chart; then, after 5 minutes of freeform practice with the same chart. Entered letters appeared in a Notepad document in Times 36pt font. Participants were not allowed to correct erroneous entries but could proceed as slowly as they wanted. Ten of the 20 participants used the earlier EdgeWrite alphabet *S*, and 10 used the new user-designed alphabet *S′*. Thus, for each alphabet, there were 26 × 5 × 2 × 10 = 2600 letters entered.

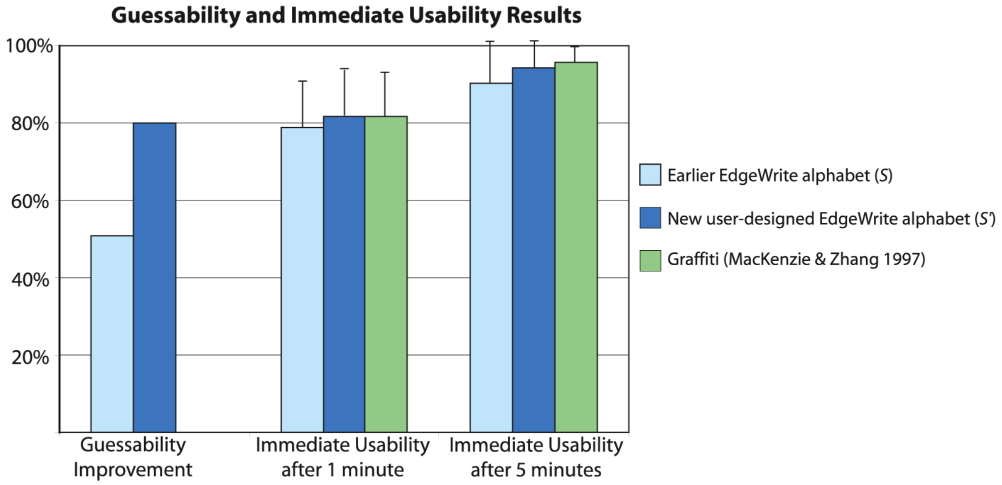


Figure 3.18. The results of the guessability and immediate usability study shows that EdgeWrite improved and that EdgeWrite is competitive with Graffiti.

As in the prior study, we measured the "accuracy attainable after minimal exposure" (MacKenzie and Zhang 1997). Figure 3.18 shows our results and those for Graffiti. After 1 minute of chart study, participants were 78.8% (σ=12.6) accurate with the earlier EdgeWrite alphabet *S*. This improved to 81.6% (12.8) for the new user-designed alphabet *S′*. This was very near the prior average for Graffiti of 81.8% (12.1). A one-way analysis of variance shows no statistical differences for the three percentages ($F_{2,42}$=0.23, p=.80).

After 5 minutes of freeform practice, participants entered *S* with 90.2% (11.0) accuracy. *S′* improved this to 94.2% (7.2). The latter was competitive with the prior result for Graffiti of 95.8% (4.0). A one-way analysis of variance is nearly significant for the three percentages ($F_{2,42}$=2.43, p=.10). A paired contrast shows Graffiti was significantly more accurate than *S* ($F_{1,42}$=4.85, p<.05), but not significantly more accurate than *S′* ($F_{1,42}$=0.40, p=.53). Unfortunately, *S′* was not significantly more accurate than *S* ($F_{1,42}$=1.73, p=.19), but the trend was in this direction.

3.3.3.9 Discussion

It was surprising that strict adherence to the guessability maximization procedure resulted in an alphabet (*S′*) with higher average immediate usability than a highly iterated designer-made alphabet (*S*). Although this improvement was not quite significant after 5 minutes, that the average immediate usability increased at all shows the power of using participants to improve even refined symbol sets (Good *et al.* 1984). Furthermore, after examining the immediate usability data, we saw that *S′* could be improved further by changing a few problematic symbols. For example, the "q" from *S′* was only 57% accurate, while the "q" from *S* was 75% accurate. Other letters that were better in *S* than *S′* were "t" (61% *vs.* 80%) and "y" (78% *vs.* 90%). Making these changes resulted in the EdgeWrite character set shown in Figure 3.9 and Appendix A.

It was also interesting that the standard deviations were similar for the three alphabets after 1 minute (~12), but shrank considerably after 5 minutes for *S′* (7.2) and Graffiti (4.0), but not for *S* (11.0), indicating more consistent performance for the more refined symbol sets. It was pleasing that EdgeWrite could be made competitive with Graffiti's laudable immediate usability.

3.4 Advanced Features

Besides the core features described above (§3.2), EdgeWrite implements advanced features not found in most unistroke systems. Three of them are described in this section: continuous recognition feedback, non-recognition retry, and slip detection. Word-level stroking is a fourth advanced feature, and is described in §3.5.

3.4.1 Continuous Recognition Feedback

In most unistroke text entry methods, a stroke is processed only after it has been segmented (e.g. after the stylus has been lifted). In contrast, EdgeWrite recognizes strokes as they unfold. Because strokes are fully defined by their corner sequences, there is no need to continuously recognize strokes after each new input point, or to determine some arbitrary criteria for knowing when to recognize strokes. Instead, a stroke-in-process can be recognized when each new corner is entered. This process is called *continuous recognition feedback*.

A few previous non-text systems have similar notions, although they have been more computationally expensive. Rubine's statistical feature-based recognizer has an extension called *eager recognition* where the system continuously recognizes a gesture after each additional input point (Rubine 1991). As soon as the recognition result is unambiguous, the gesture is given a bounding box and further movement manipulates the completed gesture. Zhao used *incremental recognition* of multi-stroke shapes to combine them into single shapes in a diagram editor (Zhao 1993). Arvo and Novins developed *Fluid Sketches*, a system in which continuous recognition after each input point gradually morphs a shape trace into an idealized shape (Arvo and Novins 2000). Their system uses differential equations and is computationally intensive, so it is only suitable for simple shapes. To improve upon this, Tandler and Prante recomputed only those features that may change with each new input point (Tandler and Prante 2001). Unfortunately, it is unclear whether their approach works for more than lines and squares. Finally, Li *et al.* tried to relax the computational demand of incremental recognition for arbitrary shapes by using *intention extraction* for previous "lag windows" (Li *et al.* 2005). Intentions are then combined to determine the incremental result of the unfolding shape.

EdgeWrite uses continuous recognition feedback to display the recognition result at the current point in the stroke. Figure 3.19 shows these results when making a "w".

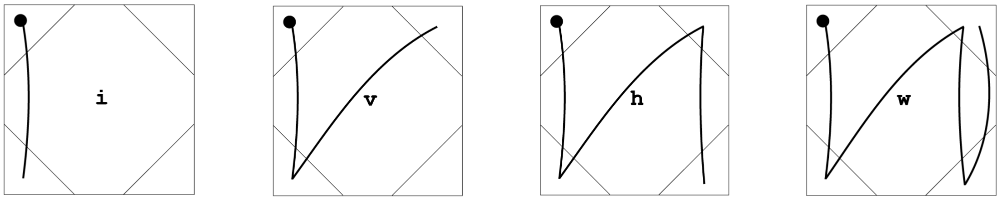


Figure 3.19. As a "w" is made, three other letters are recognized along the way.

Continuous recognition feedback is useful for novices trying to learn letters for the first time. It also is useful when combined with word-level stroking (§3.5), since incremental results may also be whole words.

3.4.2 Non-recognition Retry

What happens if a user sees an incremental result and that result is not what he or she wants? In most unistroke systems, the user would be forced to segment the stroke, incur the unwanted letter, backspace, and try again. In contrast, EdgeWrite permits users to fluidly restart the stroke *without segmentation*. This feature is called *non-recognition retry*, and it is useful for both novices and experts. Whereas novices may *see* an unwanted result, experts can often *feel* a mistake when writing. In either case, the user can simply restart his or her stroke and usually obtain the correct result. Figure 3.20 shows two strokes, both of which correctly produce the letter "w" because of non-recognition retry.

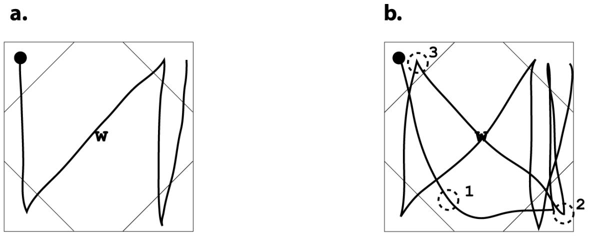


Figure 3.20. Non-recognition retry enables the garbled stroke at right to produce a "w".

For the garbled stroke in Figure 3.20b, the user began down the left side of the square but missed the bottom-left corner (1), not realizing this until after going to the bottom-right, top-right, and bottom-right again. After realizing the mistake (2), the user returned to the top-left and remade the stroke starting at (3). In practice, this process happens very quickly when the user "feels" he or she has missed a corner.

When given a gesture to recognize, if EdgeWrite finds its corner sequence unrecognizable, it iteratively trims corners from the start of the stroke's corner sequence until it finds a sequence it recognizes. This trimming is done one corner at a time from the head of the sequence. For instance, the corner sequence of the clean "w" in Figure 3.20a is 0x18242. The corner sequence of the garbled gesture in Figure 3.20b is 0x1424**18242**. Non-recognition retry tries every successively shorter sequence until it recognizes the 0x18242 at the end.

What happens if the desired letter is a subset of another? For example, 0x8242 happens to be an alternate form of "n" and is also a subset of "w". In this case, non-recognition retry will indeed find the "w" first and not produce an "n". Fortunately, in practice this subset issue is rarely a problem, since the shorter strokes are not as likely to be made erroneously. When subjects make mistakes, they tend to make them on the long gestures like "k", "m", "q", "w", etc., which are not subsets of anything else.

3.4.3 Slip Detection

Slip detection is an advanced feature for those versions of EdgeWrite based on the underlying movement of the mouse cursor—the so-called "virtual" versions with the trackball (§7) and isometric joystick (§8). In principle, however, slip detection could be applied to any of the "physical" versions as well, although this was not implemented.

When writing with Trackball or Isometric Joystick EdgeWrite, one may occasionally slip through an unwanted corner when trying to make a diagonal. Note that this is the opposite problem as with non-recognition retry, where a user accidentally *missed* a corner (Figure 3.20b). Here, a user may have accidentally *hit* a corner. To mitigate this problem, EdgeWrite includes a slip detector that adapts to the speed of the writer.

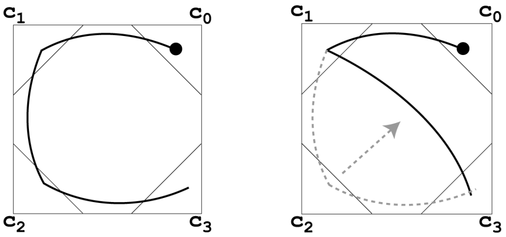


Figure 3.21. An idealized stroke is trying to move diagonally from c_1 to c_3 in the process of making an "s", but slips through c_2 on the way. The slip detector corrects this.

The slip detector tracks the average inter-corner time ψ_n for the last n corners. If after being in a corner c_1, a stroke is made abruptly at p percent of ψ_n from a corner c_2 to c_3, and c_1 and c_3 lie on a diagonal, then the stroke from c_1 to c_2 is removed and a connection is made directly from c_1 to c_3 (Figure 3.21). That is, c_2 is deemed to have been "slipped through" on the way from c_1 to c_3. Testing showed that $n = 16$ and $p = 37.5\%$ work well.

However, the slip detector is "humble." It knows it may not always be right. Therefore, upon detecting a possible slip, it processes multiple corner sequences: some with the slip-corner removed, and some with the slip-corner kept. Each recognition result is then checked for its likelihood given the preceding letter using Bellman and MacKenzie's digraph lists (Bellman and MacKenzie 1998). The result with the greatest digraph probability is retained, and the idealized stroke is updated. For example, "pu" is more common in English than "pv". Thus, after entering a "p", if the system detects a possible slip while making a "u" that would otherwise be corrected to a "v", the "u" will be entered instead. If there is no previous letter, then single-letter probabilities are used.

This process of recognizing "slip hypotheses" can be thought of as building a binary tree (Figure 3.22), where each level in the tree is created in response to a possible slip. At each split in the tree, one child removes the slip-corner from its parent's sequence, while the other child retains it. After a perceived slip, the current leaves are submitted for recognition. The final result is the leaf that contains the letter with the highest digraph probability given the previously entered letter.

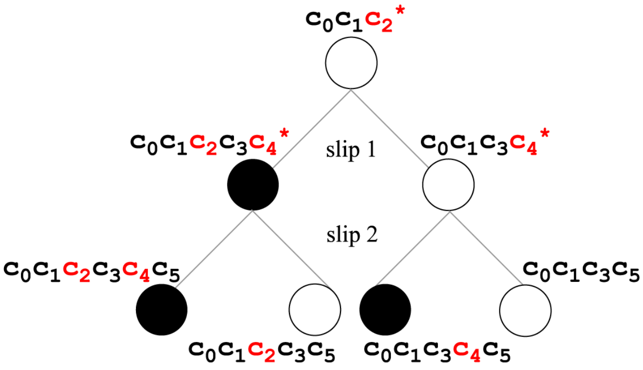


Figure 3.22. The slip detector builds a binary tree of possible corner sequences. In this tree, c_2 and c_4 are possible slip-corners. Black circles are children that *keep* the slip-corner, while white children are those that *remove* it.

Data for a motor-impaired user of Trackball EdgeWrite show that the system detects slips on about 5% of his letter strokes. Of these, about 93% are not backspaced following the detector's output. However, there is no easy way to tell from log files when the detector is failing to detect slips, since corrections are made for a variety of reasons.

3.5 Fisch Word-level Stroking*

This section describes the fourth advanced feature of EdgeWrite: *word-level stroking*. This concept, dubbed *Fisch*, is described here in general terms applicable to any unistroke text entry method. Specific adaptations for Stylus EdgeWrite (§4.2.5), Trackball EdgeWrite (§7.2.5), and Isometric Joystick EdgeWrite (§8.3.4) are left for their respective chapters. Empirical results are also saved for those chapters.

3.5.1 Motivation

Even the most efficient character-level input systems are slow because they enter only one character at a time (Kristensson and Zhai 2004). Unlike touch-typing with multiple fingers, unistroke methods do not support parallelism. This inherently limits the input rate.

One way to improve the input rate is to increase the efficiency of actions. For example, instead of one stroke producing one letter, one stroke could produce one word. A problem with word-level strokes, however, is the myriad of strokes required and the challenge of learning them. Unlike letters, words are not easily represented by "mnemonic" Roman letter-like stokes.

Stylus-based text entry methods—unistrokes and soft keyboards—range from 15–40 WPM (MacKenzie and Soukoreff 2002b, Zhai and Kristensson 2003). Recent attempts to address this limitation for stylus keyboards include optimized key layouts (Zhai *et al.* 2002) and keyboards that allow users to make word-level gestures over their surfaces (Zhai and Kristensson 2003, Kristensson and Zhai 2004). Such gestures may be called *word-level unistrokes*, which enable higher entry rates than character-level unistrokes. But stylus keyboards are not suitable for many of the smallest devices on the market today (e.g. PDA wrist watches). In contrast, unistroke letters are written on top of each other in the same space, and therefore may be suitable for particularly small devices. But unistroke entry is character-by-character, which limits its speed. To the best of my knowledge, there have been no word-level stroking solutions for character unistrokes.

Fisch provides a design for extending character-level unistrokes to word-level unistrokes using a new technique called *in-stroke word completion*. The idea is not to define overly verbose strokes that stand for complete words, nor to add a separate word

* Parts of this section are adapted from (Wobbrock and Myers 2006b).

completion list for selecting words, since list-based word selection often slows people down (Goodenough-Trepagnier *et al.* 1986, Soede and Foulds 1986). Instead, the idea is to allow character-by-character entry to remain unchanged while providing *minimal* extensions to character-level unistrokes that turn them into words. Users begin word-level strokes by stroking a word's first letter, and then, without lifting, fluidly complete their stroke to write an entire word. The same stroke always produces the same word, enabling users to memorize the strokes and ramp from character-level entry to word-level entry. Since natural language follows Zipf's law (Zipf 1932), it is possible that learning even a small number of common word-level strokes will produce an increase in overall entry rates. For example, the word "the" represents over 6% of the British National Corpus, and the most common 100 words account for over 46% (Zhai and Kristensson 2003).

The principles outlined above are similar to those behind the SHARK stylus keyboard (§2.3.3). This design for word-level unistrokes is therefore dubbed *Fisch*, for fluid in-stroke completion shorthand. Whereas SHARK uses a stylus keyboard, Fisch uses only a little more space than that required for letter unistrokes. With SHARK and Fisch, stylus keyboards and unistroke alphabets can both support character-level and word-level entry, thus better serving both novices and experts.

3.5.2 The Fisch Design

The Fisch design for word-level strokes can be applied to any character-level unistroke method, such as Graffiti, Jot, or EdgeWrite. Fisch integrates word-level strokes into these unistroke methods without altering their process of character-level entry. The explicit design goals of Fisch are that:

- Character-level strokes remain unchanged.
- Character-level strokes are *minimally* extended to produce words.
- The same stroke always produces the same word, enabling memorization through motor repetition.
- Users can gradually transition from character-level entry to word-level entry as they become experts.

The core idea in Fisch is for users to make a "subgesture" within a letter stroke that indicates that the letter itself is finished and subsequent motion is for the completion of a word. A natural and easily detectable subgesture is a “pigtail” loop like the one used in Scriboli (Hinckley *et al.* 2005). Other “subgestures” are possible, including stylus-dwell or explicit segmentation (i.e. lift). For now, we will assume a loop is used.

When a loop is detected, the stroke is recognized and the result serves as the prefix to potential word completions. Additionally, a bounding box is imposed around the stroke such that the stroke itself serves as the frame of reference for selecting word completions by crossing over the box’s boundary toward the desired word (Figure 3.23). If no loop is detected, the entire stroke is processed as a regular character-level stroke. Thus, letter strokes remain unchanged, but can be extended to produce words.

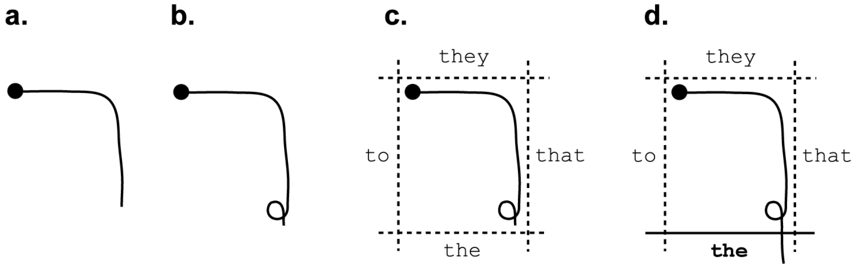


Figure 3.23. Making a Graffiti “t” and then extending it to produce the word “the”. Dashed lines represent the stroke’s bounding box and crossing goals.

In Figure 3.23a, the user has written a Graffiti unistroke for the letter “t”. If the user were to lift the stylus at this point, a “t” would be produced as usual. However, in Figure 3.23b, the user continues the stroke by making a loop subgesture. In Figure 3.23c, the system detects this loop and places an appropriately sized bounding box around the stroke. The “t” is recognized and the system presents the four most common words beginning with “t” at each side of the box. The sides on which the words appear are fixed such that the same word always appears on the same side for a given stroke, allowing users to reliably enter words in single strokes. In Figure 3.23d, the user fluidly continues the gesture to perform a crossing task, which selects the word across the penetrated boundary. Studies have shown crossing to be faster than pointing for close-range selection tasks (Accot and Zhai 2002). Thus, when the user lifts, the word “the” is entered. Importantly, the same stroke will always produce the word “the”.

For less common words, users can enter more letters to serve as a prefix before selecting a completion. For example, once a user has entered a “t”, a subsequent “o” stroke will produce “to-” completions “to”, “told”, “too”, and “today”. However, long prefixes are rarely needed, as a surprising amount of language coverage is achieved by showing just four words per entered letter. Figure 3.24 shows the weighted coverage of the 17,805 most common English words (Kucera and Francis 1967) for 1–5 letter prefixes with four frequency-based completions per letter. According to the graph, a user has a 49.0% chance of being able to enter the word they desire in just one fluid stroke!

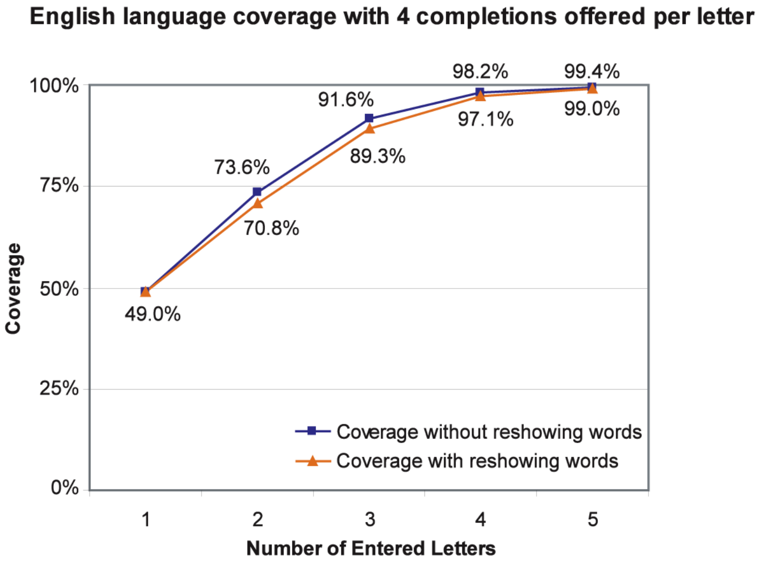


Figure 3.24. Weighted coverage of the 17,805 most common English words when showing four fixed frequency-based completions per entered letter.

The difference between the top and bottom lines in Figure 3.24 indicates a design decision: if the user enters a “t”, one of the words shown is “the”. If the user then enters an “h”, should “the” be re-shown, even though the user did not select “the” already? When not re-showing words in pilot studies, novices sometimes entered letters *past* the initial presentation of their desired word, losing sight of their intended completion. Since the gain obtained in not re-showing is minimal, Fisch opts to indeed re-show words.

Good user interfaces must allow users to abort actions underway. A user may cancel a word-level stroke in Fisch by simply retreating over the crossing boundary and terminating the stroke in the interior of the bounding box (Figure 3.25a). Depending on the application, designers can elect to have this enter the character-level result, or enter nothing at all. On the other hand, if a stroke ends outside the box, the completion that had its boundary crossed *last* will be the one that is selected (Figure 3.25b). Importantly, crossing goals extended infinitely far in both directions along their length. (Note the arrow on the bottom of the “to” goal in Figure 3.25b.)

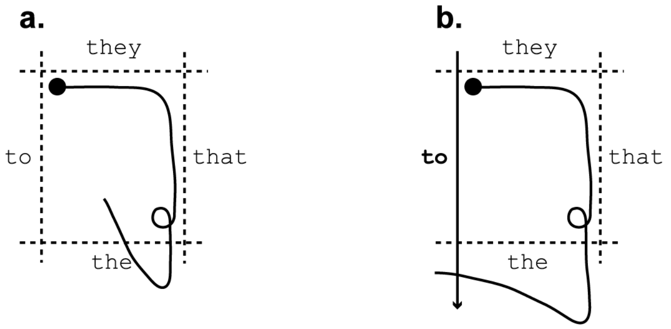


Figure 3.25. Word-level stroke cancellation (left) and a change in selection (right).

After a completion is entered, users can quickly undo it by performing a dedicated backspace stroke. In most stroke alphabets, a simple stroke from right-to-left enters a backspace. Thus, a right-to-left double-swipe, or some other backspace variant, may be used to undo the completion, restore the former prefix, and restore the former word completions at their previous corners.

Thus far, we have described how users can extend character-level strokes to complete words. However, in the early stages of exposure, users cannot fluidly complete words, but instead stop after each character stroke and look at the potential completions. Therefore, Fisch leaves completions displayed after each letter is entered and permits users to simply tap a word to select it. This is called *direct word selection*.

As stated, the Fisch design is suitable for most unistroke alphabets like Graffiti, Jot, and EdgeWrite. Adaptations exist for Stylus EdgeWrite (§4.2.5), Trackball EdgeWrite (§7.2.5), and Isometric Joystick EdgeWrite (§8.3.4).

3.5.3 Fisch Implementation

The word prediction and completion system implemented as part of the EdgeWrite DLL (§3.6.1) has four components: (1) a vocabulary list of words and frequencies, (2) an optional user-defined vocabulary list, (3) a trigram list with trigram frequencies, and (4) an adaptive bigram cache that stores users' words at runtime. The first and second provide Fisch's "fixed" frequency-based word completions as words are being made. The third and fourth are not part of Fisch, per se, but provide *context-dependent word predictions* after a word has been completed (i.e. after a SPACE has been entered).

The vocabulary list is stored in an alphabetically sorted array enabling binary search for fast lookups. Each array slot contains a word string and the word's frequency count. This is all the data necessary for Fisch's fixed frequency-based word completions.

Also in each slot of the vocabulary array is a hash table whose *keys* are word indices and whose *values* are a list of word indices. The slot's word string represents the first word of a trigram, its hash table keys represent second words, and its hash table list values represent third words. These data structures allow fast lookups for both fixed completions and context-dependent predictions.

When a letter is entered, words that begin with the current prefix are gathered from the vocabulary list. If a user-defined vocabulary list is loaded, its words with matching prefixes are also gathered. These pooled words are then sorted in a separate list according to their frequencies. The top four words are then offered as Fisch's completions. Since frequencies are pre-computed based on a large corpus of English words, these four completions will always be the same for a given prefix.

When four frequency-based words are retrieved from the language model, they are assigned to corners such that the highest priority word is given the corner in which the current stroke resides. The two adjacent corners receive the next two words, and the lowest priority word is placed at the diagonal away from the stroke's current corner. Once a word has been shown, it is stored in a hash table along with its corner and a "half-life." If a word is shown again, it will be shown in the same corner as it was before. If the word goes unused for some time, it will "decay" and be eligible for reassignment. If a collision occurs with two words vying for the same corner, the highest priority word wins.

When a SPACE is entered, context-dependent predictions are offered. The most recent two words are used to look up possible third-word predictions. The first word is found in the vocabulary array using binary search. The second word's index, which was found when the word was entered, is used as a hash key in first word's hash table for fast lookups. The value returned, if any, is a linked list of possible third words. The top four from this list are shown as predictions.

Predictions also come from an adaptive bigram cache. This cache holds recent bigrams so that when a user enters a previously used word, words that followed it can be offered as predictions. The cache is a list maintained in priority order such that when a new bigram is entered or an old bigram is reused, it is placed at the top. Unlike the trigrams, the adaptive bigram cache accommodates out-of-vocabulary words, enabling the prediction of last names from first names, etc.

The English vocabulary list and trigrams were built by parsing 850MB of news articles from the Wall Street Journal, Ziff Davis, Los Angeles Times, and Associated Press. This parsing was carried out with the CMU-Cambridge Statistical Language Modeling toolkit (Clarkson and Rosenfeld 1997). Custom parsers then pared down the toolkit's results, keeping 20,000 of the most common words, and only trigrams that occurred 20+ times. After certain abbreviations were removed, the result was a 258 KB vocabulary list of 19,122 words with frequency counts totaling 132,701,943. The maximum frequency count was for the word "the" at 7,686,122, or 5.79%. The trigram list is 10.6 MB and contains 517,988 trigrams with frequency counts totaling 40,230,622. The maximum frequency count is for the trigram "the United States" at 46,947, or 0.12%. Although news articles were parsed, this procedure could easily be run over other corpora (e.g. email, instant messaging, academic prose, etc.).

3.6 EdgeWrite Technology

Rather than have each version of EdgeWrite require its own implementation from scratch, EdgeWrite technology is encapsulated in a reusable library. The majority of EdgeWrite versions are implemented using this library, with the exception of Stylus EdgeWrite for the Palm OS (§4.2), which requires its own separate implementation.

3.6.1 EdgeWrite Library

The EdgeWrite *dynamic link library* (DLL) is written in Visual C# and can be used with any .NET-compatible language (e.g. VB .NET, C#, J#). The library defines three primary singleton classes central to different aspects of EdgeWrite. These classes are `Recognizer`, `Painter`, and `Logger`.

The *Recognizer* is the main class in EdgeWrite. It defines an input plane on which point input occurs. Besides points, it can be given corners directly, whereby it adds the centroid of the corner region as a point. The Recognizer also allows the location of the square to be interactively defined by the user, which is helpful for aligning a physical template with the software definition of the square. Its primary methods are `AddPoint()`, `AddCorner()`, `Peek()`, and `Recognize()`.

The Recognizer has embedded within it the `Charset.xml` file. This file encodes the EdgeWrite character set and is shown in Appendix A. A programmer who wishes to supply the Recognizer with a different XML character set may do so at runtime.

The Recognizer is also in charge of interfacing with a backend word prediction and completion system for implementing Fisch word-level stroking. By default, corner re-entries are used to distinguish character-stroking from word-selection. However, clients can also tell the Recognizer when character-stroking has ended and word-selection has begun using the `LockOrCycleWords()` method, which provides support for customized methods.

The *Painter* is in charge of the EdgeWrite view. This class supports various properties and methods for drawing, including support for fonts. Its main method is `Paint()`, and key properties include `DrawResult`, `DrawWords`, `ResultFont`, `WordFont`, `Regions`, and `PointConnection`. This last property holds one of three enumerated `PointConnectionStyle` values: `Unconnected`, `Straight`, or `Bowed`. By simply setting the connection style to one of these three values, the stroke drawing can correctly portray very different versions of EdgeWrite (Figure 3.26).

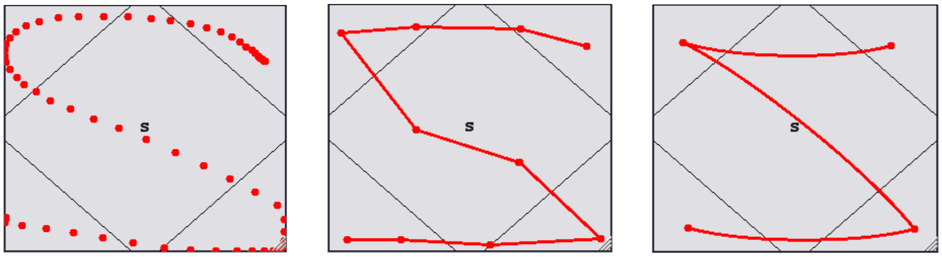


Figure 3.26. The Painter class supports three styles of stroke drawing: unconnected, straight, and bowed. Examples of each are EdgeWrite for the touchpad, displacement joystick, and trackball, respectively.

The `Bowed` style deserves special mention. When this style is active, successive points will be drawn with a Bezier curve between them. The amount and direction of a curve depends on any curves previously drawn between the same two points. Curves with increasingly greater bowing are drawn as more arcs connect the same two points.

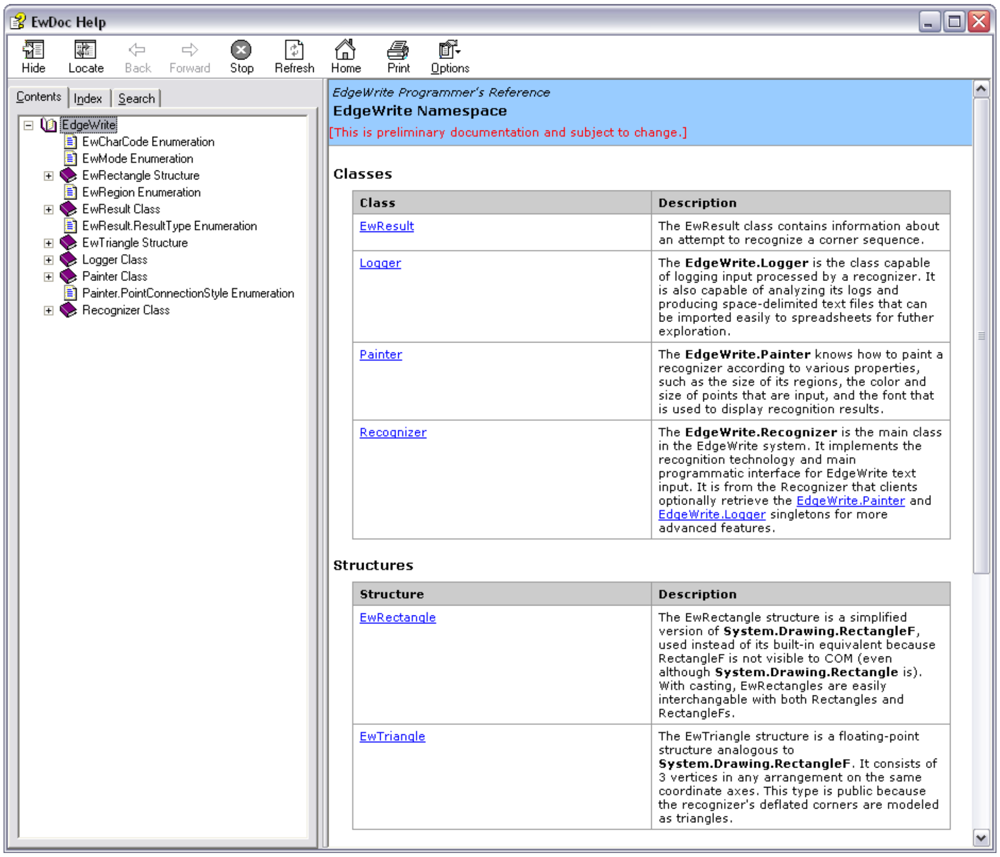
The *Logger* supports the writing of EdgeWrite activity to files. Corner sequences or even full point traces can be logged as a part of each stroke. The Logger's main methods are `Open()`, `Stroke()`, and `Analyze()`. The last method writes log results to a text file for statistical analysis.

The EdgeWrite library is versatile, being used in all versions of EdgeWrite except the one for the Palm OS. Without its word prediction files, which are about ~10 MB in size, the compiled EdgeWrite DLL is 188 KB. It is implemented in about 10,000 lines of C# code. Some versions of EdgeWrite may only add a few hundred lines to that, while others may add thousands. For example, Trackball EdgeWrite (§7) is implemented in about 5000 additional lines of C# beyond the DLL. This extra code is mostly application-specific, devoted to things such as focus handling, application preferences, window management, and so forth.

3.6.2 Programmer's Reference

The EdgeWrite DLL is fully documented with a compiled help file resembling those used by the Microsoft Developer Network (MSDN). This file, called the *EdgeWrite Programmer's Reference*, describes all public constructors, methods, and properties used in the EdgeWrite DLL—more than just those of the four singletons above. The documentation includes examples and sample code for those wishing to build an EdgeWrite application.[6] Figure 3.27 shows a screenshot.

[6] The EdgeWrite DLL can be downloaded from http://www.edgewrite.com/dev.html.



EdgeWrite Programmer's Reference

EdgeWrite Namespace

[This is preliminary documentation and subject to change.]

Classes

Class	Description
EwResult	The EwResult class contains information about an attempt to recognize a corner sequence.
Logger	The **EdgeWrite.Logger** is the class capable of logging input processed by a recognizer. It is also capable of analyzing its logs and producing space-delimited text files that can be imported easily to spreadsheets for futher exploration.
Painter	The **EdgeWrite.Painter** knows how to paint a recognizer according to various properties, such as the size of its regions, the color and size of points that are input, and the font that is used to display recognition results.
Recognizer	The **EdgeWrite.Recognizer** is the main class in the EdgeWrite system. It implements the recognition technology and main programmatic interface for EdgeWrite text input. It is from the Recognizer that clients optionally retrieve the EdgeWrite.Painter and EdgeWrite.Logger singletons for more advanced features.

Structures

Structure	Description
EwRectangle	The EwRectangle structure is a simplified version of **System.Drawing.RectangleF**, used instead of its built-in equivalent because RectangleF is not visible to COM (even although **System.Drawing.Rectangle** is). With casting, EwRectangles are easily interchangable with both Rectangles and RectangleFs.
EwTriangle	The EwTriangle structure is a floating-point structure analogous to **System.Drawing.RectangleF**. It consists of 3 vertices in any arrangement on the same coordinate axes. This type is public because the recognizer's deflated corners are modeled as triangles.

Figure 3.27. The EdgeWrite Programmer's Reference documentation.

3.6.3 Palm OS HACK

Because the Palm OS is not a Microsoft Windows-based platform, it requires a separate implementation of EdgeWrite. In fact, Palm OS 3.5–4.x requires one type of implementation known as a "HACK," and Palm OS 5.0 and above requires another.

The HACK version of EdgeWrite is able to replace Graffiti on Palm OS devices by remapping certain operating system calls, called "traps," to function addresses within the EdgeWrite codespace. This is necessary because Palm OS devices are not multithreaded and without separate address spaces. Thus, in order to have EdgeWrite work concurrently with any other application (e.g. MemoPad), EdgeWrite must appear to the operating system as if it *is* part of the operating system. Because a user may have multiple HACKs on their device at a time, a *HACK manager* is necessary to avoid HACKs interfering with others. The HACK manager keeps track of which traps are mapped to which HACKs, and ensures that the traps are passed along the "HACK chain" accordingly. EdgeWrite uses the *X-Master-Light* from LinkeSOFT (http://linkesoft.com/xmaster) as its freeware HACK manager of choice.

On Palm OS 5.0–5.4 devices, the operating system no longer fires traps in the same way, and so a different implementation is necessary. Instead, OS 5 devices broadcast a limited set of *notifications* to applications when certain events occur. Most applications do not register with the operating system to receive such notifications, but EdgeWrite does. When activated, EdgeWrite receives notifications for pen events and drawing, and responds accordingly. This is similar in spirit to the HACK method, but is sanctioned and governed by the operating system.

Chapter 4

Stylus EdgeWrite*

4.1 Motivation

As described in §3.1.1, the two predominant forms of stylus-based text entry—stroke alphabets and "soft" stylus keyboards—can be difficult for people with motor impairments. Stroke alphabets require fine motor control and consistent screen pressure. Stylus keyboards require the repeated acquisition of miniature on-screen targets. Both methods can be difficult for people with tremor, spasm, poor coordination, rapid fatigue, or low strength. As a result, it is necessary to design a better method for stylus text entry on PDAs for people with these challenges. In particular, stability, tactility, reduced visual search, and short stylus travel distances are of key importance.

Tremor is particularly problematic, as tremulous users may "bounce" the stylus repeatedly on the screen, triggering unwanted modes and unwanted characters. In fact, this problem renders Graffiti entirely unusable for some people. A similar problem is that users with low strength often cannot apply enough pressure to make a single, contiguous unistroke. The built-in digitizer on a Palm OS device, for example, will perceive such a stroke as a series of disconnected stroke segments, separated by momentary lifts of the stylus. A more accessible method of text entry must have tolerance to both problems.

* Parts of this chapter are adapted from (Wobbrock *et al.* 2003b, Wobbrock and Myers 2006b).

Able-bodied users may also benefit from more stable means of text entry. Since PDAs are designed to be used "on the go," many situations arise where added stability can be beneficial: riding a bus, walking, or annotating slides during a presentation while standing.

As the line tracing study discovered (§3.1.3), *physical edges* offer many desirable properties that may be useful in developing a more accessible stylus-based technique. Appling pressure against an edge while moving a stylus provides:

- *Greater stability*: Decreased movement variability and movement offset (MacKenzie *et al.* 2001a).
- *Greater speed*: Ability to move quickly yet remain on the target line.
- *Higher accuracy*: Targets along an edge or in a corner are easier to acquire.
- *Tactile feedback*: No longer is visual feedback the only means of self-correction during movement, as tactile feedback is available (Buxton *et al.* 1985).
- *Fitts' law win*: Edges allow for "target overshoot," where acquiring targets on edges is easier than acquiring targets "in the open" (Walker and Smelcer 1990).

The research goal in Stylus EdgeWrite is to exploit these "pure" benefits of physical edges in a text entry technique, and to prevent other factors such as cognitive difficulties from diluting them. Stylus EdgeWrite goes a long way toward realizing these goals, as it relies heavily on edges and corners, both interactively and algorithmically. These edges are imposed on the input area by a transparent plastic template with a square hole, inside which all text entry is performed.

Empirical results for errors during entry show that Graffiti KSPC is 18.2% higher than EdgeWrite KSPC (1.43 *vs.* 1.21, $p<.05$) for able-bodied users formerly unfamiliar with either technique.[7] This benefit comes without a significant difference in speed. Users with motor impairments succeeded at using EdgeWrite but were largely unable to use Graffiti due to excessive tremor and bounce. EdgeWrite's physical edges reduced the propensity of tremulous users to bounce the stylus on the screen, and the software tolerates such bouncing by not producing unwanted characters.

[7] Recall that lower KSPC corresponds to higher accuracy, with 1.0 being "perfect" (Soukoreff and MacKenzie 2001).

4.2 Design

This section describes aspects of the Stylus EdgeWrite design that are specific to this version of EdgeWrite. Readers are encouraged to first read chapter 3, which covers the core design concepts central to all versions of EdgeWrite.

4.2.1 The Feel of Stylus EdgeWrite

The Fitts' law benefits of edges—the ability to overshoot a target by an arbitrary amount—helps Stylus EdgeWrite letters feel more like their hand-printed counterparts (Figure 4.1). When users hit a corner with a high force as they move the stylus, they may feel as if they are moving farther in that direction than when they hit the same corner with less force. This is, at least in part, why EdgeWrite letters can be easily learned: they feel similar to Roman letters despite being mapped onto a square.

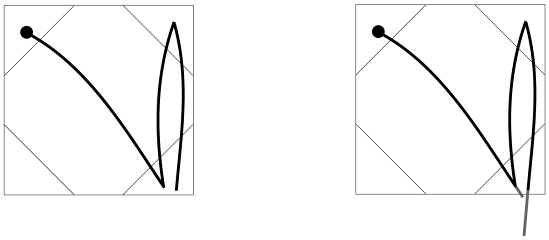


Figure 4.1. An EdgeWrite "y" (left) and a conceptual drawing for how the "y" may feel in Stylus EdgeWrite (right).

In Figure 4.1, the form for "y" impacts the bottom-right corner twice: once on the diagonal down-stroke, and once on the straight down-stroke (left). However, in making this form, it can feel like the second stroke goes farther down because of the greater force applied to the bottom edge on the second impact (right).

4.2.2 The Design of Corners

Although EdgeWrite's corner-based recognition algorithm is straightforward (§3.2), the design of Stylus EdgeWrite's corners was not obvious. The corners began naïvely as points instead of areas, and this proved to be inadequate, as users rarely hit the exact corners. This was because users held their styluses at various angles. An angled stylus impacts the edge of the plastic template a few millimeters above its tip, causing the tip to jut into the square a few pixels, even when the stylus feels flush against the edge (Figure 4.2).

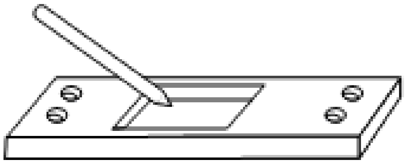


Figure 4.2. The tip of an angled stylus meets the PDA screen a small distance from the edge of the dominant-hand-side of the square. This image is of a left-handed user.

After increasing the corner size to an appreciable area, two other problems emerged. Once moving, users would often accidentally hit corners, particularly when doing a diagonal stroke, as in an "s." This seemed to suggest that the corners should be made smaller; but if the corners were too small, users would often fail to hit them on pen-down, particularly for rapid backspace strokes (across the top from right to left). It seemed that large corners were necessary when the stylus went down, but then small corners were necessary thereafter.

The next step in the design process added precisely this. The corners were *inflated* until the stylus was detected within one of them, and then *deflated* while the stylus was moving (Figure 4.3). Deflated corners are not rectangular but are triangular to make accidental corner-hits rare, especially during the making of a diagonal.

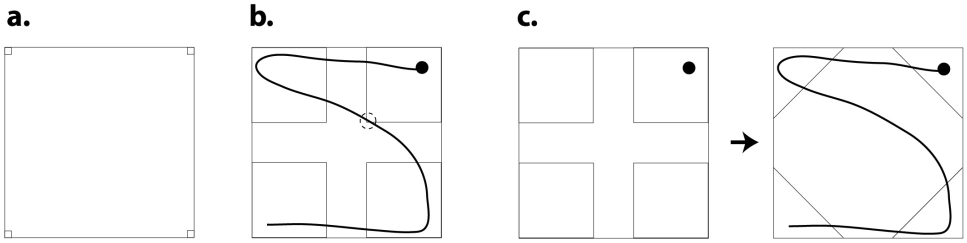


Figure 4.3. The evolution of corners in Stylus EdgeWrite. In (a), corners are just pixels. In (b), they are rectangular regions, but diagonals sometimes hit unwanted corners. In (c), they begin as rectangles, but "deflate" into triangles.

An observation during a user study prompted the most recent iteration on the corners. A user with a chronic wrist injury held the stylus at a fairly shallow angle relative to the PDA screen, similar to Figure 4.2. The result was that the elevated edge of the plastic square prevented the tip of the stylus from getting close to the side of the square. Figure 4.4 shows the resultant adjustments made for both left- and right-handed users. Extra corner area is provided along the *x*-axis for the dominant-hand side of the square in order to account for users who hold their styluses at steep angles. A nice property of this design is that it does not negatively impact users who hold their styli more vertically.

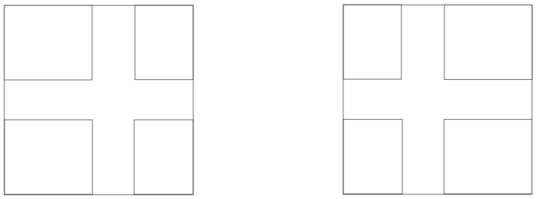


Figure 4.4. Left- and right-handed adjustments of the corners before deflation.

4.2.3 The Plastic Template

Stylus EdgeWrite's plastic template is important for EdgeWrite to work well. Informal testing of different physical square sizes using a Palm V*x* stylus showed that writing could be effective in squares ranging from 0.7–1.9 cm on a side, with 1.3 cm having the best combination of speed, accuracy, and "feel." Although some of the larger square sizes were slightly more accurate for able-bodied users, the stylus travel distances seemed too large for people with motor impairments, who need short diagonals since diagonals lack a physical edge. Thus, the size of the square hole in the "official" EdgeWrite template was set to 1.3 cm on a side, or 1.69 cm^2 (Figure 4.5).

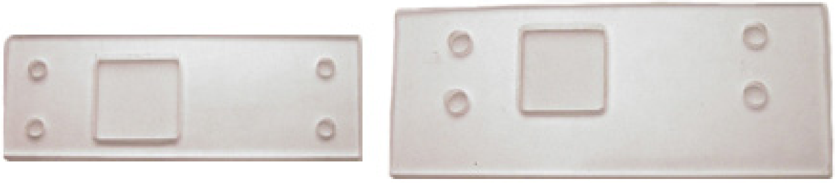


Figure 4.5. Two plastic templates, each with a 1.69 cm^2 hole.

Numerous prototypes exist, two of which are shown in Figure 4.5. The left model is smaller and sits directly on the Palm screen. We found this to work fine for able-bodied users, but users with motor impairments sometimes put pressure with their fingers on the template, causing it to press against the screen and interfere with the digitizer. We designed the right model in order to avoid putting pressure on the screen. This template sits on the Palm chassis and therefore cannot touch the screen's surface. In both cases, the templates are taped to the PDA chassis for anchoring. Both templates are 0.15 cm thick.

Different Palm PDAs require differently sized plastic templates. After carefully measuring a PDA's input area, the templates are laid out in a vector drawing program. Included in the layout are holes corresponding to the positions of any soft buttons that the template would otherwise obscure. The templates are then laser-cut from clear acrylic. Figure 4.6 shows a Tungsten E template schematic, which also works well on a Palm V*x*.

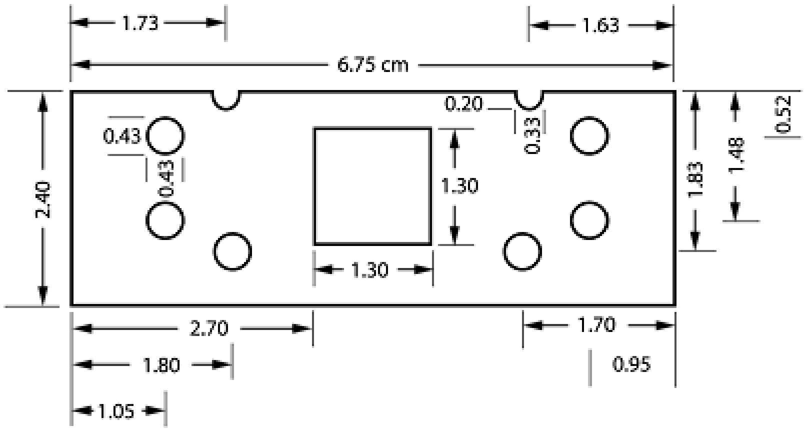
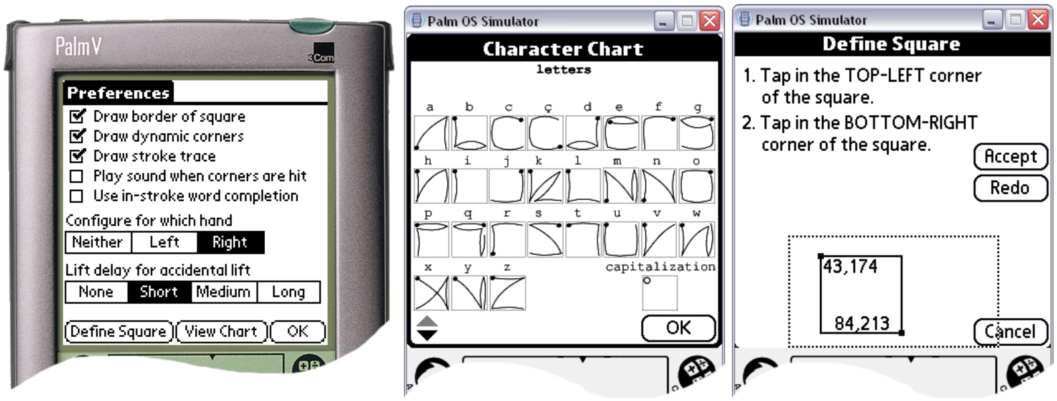


Figure 4.6. Template schematic for a Tungsten E, which also works well on a Palm V*x*. Holes provide access to the soft stylus buttons. This template is designed to sit on the device chassis, above the PDA screen. All measurements are in cm.

4.2.4 Application Preferences

Stylus EdgeWrite works as a HACK on Palm OS 3.5–4.x devices, and as a notification-based application on Palm OS 5.0+. See §3.6.3 for more details about this implementation.

Both versions of EdgeWrite for the Palm OS have configuration screens that support options for drawing, handedness, tolerance for low stylus pressure, and defining the square (Figure 4.7). The lattermost involves tapping in the top-left and bottom-right corners of the plastic hole to tell the software where the physical square is. Also, a software character chart can be shown whenever the application is active by tapping one of the soft buttons through one of the holes in the plastic template.



Preferences

☑ Draw border of square
☑ Draw dynamic corners
☑ Draw stroke trace
☐ Play sound when corners are hit
☐ Use in-stroke word completion

Configure for which hand: Neither | Left | [✓ Right]

Lift delay for accidental lift: None | [✓ Short] | Medium | Long

Define Square | View Chart | OK

Define Square

1. Tap in the TOP-LEFT corner of the square.
2. Tap in the BOTTOM-RIGHT corner of the square.

43,174

84,213

Figure 4.7. A preferences screen, a character chart, and a screen for defining the square. Defining the square aligns the software square with the physical plastic square.

The option for tolerating low stylus pressure sets a short lag after each pen-up event. This lag can be set to *None*, *Short*, *Medium*, or *Long* corresponding to 0, 125, 250, and 500 ms, respectively. The lag is a time window in which additional pen input will be appended to the current stroke. This provides tolerance for users who cannot apply consistent pressure to the screen during a single stroke; as a result, they may lift frequently and inadvertently. This feature proved useful to "Jim," a motor-impaired subject with low strength who also extensively tested Trackball EdgeWrite (§7.3.3).

4.2.5 Fisch in Stylus EdgeWrite

Recall the discussion of the Fisch technique for word-level stroking in unistroke text entry systems (§3.5). As described, the design will not work in Stylus EdgeWrite because a plastic template bounds the input area, preventing words from being selected via goal crossing. As a result, it is necessary to adapt Fisch to Stylus EdgeWrite.

The adaptation consists of three parts. First, Fisch no longer uses a "floating" bounding box that is determined by the location of the stroke. Instead, the bounding box is assumed to be the fixed square defined by the plastic template. Second, Fisch maps word completions to EdgeWrite's corners instead of to the sides of the box, since corners trap a moving stylus. Third, instead of detecting pigtail loops, Fisch detects corner re-entries, which amount to a similar thing. As before, users can cancel a stroke, now by lifting outside the corners. Figure 4.8 shows an EdgeWrite "t" and word-level strokes for "the" and "they".

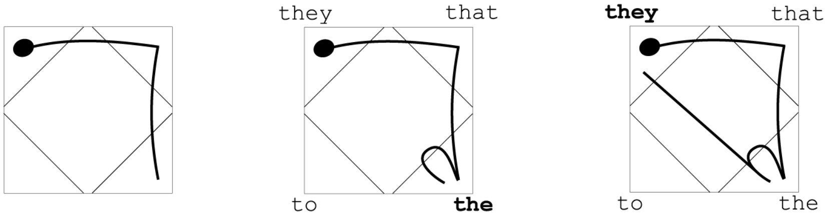


Figure 4.8. Stylus EdgeWrite unistrokes for "t", "the", and "they".

4.3 Evaluation

This section presents results from various laboratory studies of Stylus EdgeWrite. These studies were conducted prior to the finalized design of Stylus EdgeWrite. In particular, corners did not yet deflate as triangles (§4.2.2), there was no accidental lift tolerance (§4.2.4), non-recognition retry was not implemented (§3.4.2), and the EdgeWrite alphabet had not yet been refined through the guessability study (§3.3.3). With these refinements, results would only improve for the most up-to-date version of Stylus EdgeWrite. Unless otherwise noted, only the character-level version was tested.

4.3.1 Novice Able-bodied Use

Although Stylus EdgeWrite is targeted towards people with motor impairments, its speed and accuracy were assessed with able-bodied users to validate its design. Such an evaluation is also informative with respect to situational impairments.

4.3.1.1 Subjects

Ten subjects who had no prior experience with handheld text entry were recruited from the local community. Only half of the subjects were university students. Subjects were paid $10 for their time. The experiment lasted one hour and consisted of two parts: a practice session and a testing session.

4.3.1.2 Apparatus

A Palm V*x* device served as the handheld computer on which the study was run. Test software presented text phrases on the PDA screen, just above the text entry area where subjects reproduced the text. The close proximity of the presented phrases and the text entry area mitigated the two foci-of-attention problem (MacKenzie and Soukoreff 2002b).

Test phrases resembled realistic entries: letters, names, phone numbers, addresses, and URLs. These phrases were created to resemble those used by Sears and Arora (Sears and Arora 2002). It is worth mentioning that this study was run before MacKenzie and Soukoreff released their widely-used corpus of 500 phrases (MacKenzie and Soukoreff 2003), which later studies employed.

All performance data was logged on a nearby laptop over a serial cable attached to the Palm PDA. This data was then processed by a log file parser capable of computing numerous statistics like intra-character times, errors in the final string (MSD), keystrokes per character (KSPC), gestures per character (GPC), etc. Space precludes a detailed discussion of each measure. See §2.4.2 for more information about computing these measures.

A character chart was available to subjects for use if they could not remember how to make a character. Subjects were encouraged to guess "once or twice" if they could not recall a character, after which they could use the chart. Their uses of the chart were recorded.

4.3.1.3 Procedure

Each subject was assigned randomly to a between-subjects condition for *input method*: Graffiti *vs.* EdgeWrite. The experiment lasted 1 hour and consisted of two parts: a practice session followed by a testing session.

Before the practice session, subjects were taught the basics of the input method to which they were assigned. For Graffiti, this included things like pointing out the input area, pointing out the letter area and the number area, and demonstrating the appropriate pressure with which to hold the stylus against the screen. It did not include expert-level nuances like making the "v" backwards. For EdgeWrite, this explanation included the importance of pressing firmly against the edges during movement, and the importance of hitting corners in the correct order.

The practice session consisted of making each character twice successfully. Thus, if a character was not recognized correctly, it was retried until success. Subjects progressed through a character chart for EdgeWrite or Graffiti like the one shown in Figure 3.9.

The testing session consisted of entering 20 phrases, all about 50 characters in length. Phrases were presented in the same fixed order for every subject. Subjects were not artificially constrained in their entry of text: they could make errors or get out of sync with the presented phrase. They were told before the first trial to "proceed quickly and accurately, as you would enter text in the real world" (Soukoreff and MacKenzie 2001).

4.3.1.4 Adjustment to Data

Subjects continually improved over the first 12 trials, then began to flatten out in speed and accuracy, so these first 12 trials are treated as additional training exercises. Thus, only the last 8 trials are included in our statistical comparisons. In all comparisons, the EdgeWrite value is reported first, followed by the Graffiti value.

4.3.1.5 Results

The three measures of obvious importance are speed, accuracy *during* text entry, and accuracy *after* text entry is complete. All three of these measures are calculated with the methods described in §2.4.1–2.4.2.

A one-way repeated measures analysis of variance[8] yields a non-significant result for speeds: 6.6 *vs.* 7.2 WPM ($F_{1,56}$=0.27, ns). If we consider all 20 trials to examine learning, the means are even closer: 5.85 *vs.* 5.97 WPM ($F_{1,152}$=0.02, ns). Further testing would be required to differentiate the speeds of these two methods.

Accuracy during entry can be measured using keystrokes per character (KSPC). A one-way repeated measures analysis of variance yields a significant result in favor of EdgeWrite: 1.21 *vs.* 1.43 ($F_{1,56}$=9.32, p<.02). Graffiti is therefore 18.2% more prone to error; equivalently, EdgeWrite has 15.4% lower KSPC. Note that non-character-producing strokes such as mode-setters are not included in this measure, since that would unduly penalize any entry technique with modes. Figure 4.9 shows KSPC for Graffiti and EdgeWrite over the last 8 trials.

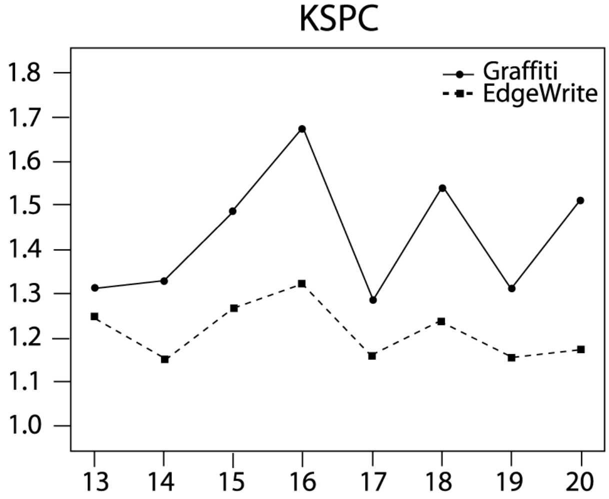


Figure 4.9. Average KSPC of EdgeWrite and Graffiti. Lower is better; 1.0 is perfect.

Accuracy *after* entry can be measured using minimum string distance (MSD). A one-way repeated measures analysis of variance is not significant: 0.34% *vs.* 0.39%

[8] The analysis used a mixed model analysis of variance with a fixed effect for *entry method* and a random effect for *subject*. Mixed model analyses result in higher denominator degrees of freedom than more tranditional approaches but do not increase the chances for statistical significance. See Chapter 3 in (Littell *et al.* 1996) for more details.

($F_{1,56}$=0.02, ns). For our trials, this is roughly equivalent to one erroneous character for every 250 characters entered, or about 5 trials. The fact that these values are so close means speeds can be equitably compared, since in both methods few errors were left in the transcriptions.

One also can examine the intra-character time (pen-down to pen-up) to determine which alphabet's strokes are faster. A one-way repeated measures analysis of variance yields a significant result in favor of Graffiti: 580 *vs.* 290 ms ($F_{1,56}$=20.57, p<.005). This explains, at least in part, why there was no significant difference in speed even though EdgeWrite was more accurate during entry (and thus required less time correcting errors). EdgeWrite makes fewer errors but Graffiti has faster characters, so overall speed is about the same.

The data can be analyzed with a new measure similar to the KSPC dependent measure called gestures per character (GPC). This measure is the same as KSPC but includes mode setters and all other recognized strokes—things KSPC must leave out. The result can be used to show which alphabet is more "verbose," requiring more multi-stroke characters and more modes. A one-way repeated measures analysis of variance yields a significant result in favor of EdgeWrite: 1.26 *vs.* 1.62 GPC ($F_{1,56}$=14.65, p<.005). This helps to support the claim that EdgeWrite uses less mode strokes, something important for motor-impaired users, for whom every stroke counts.

Finally, both alphabets were extremely learnable and guessable, even in spite of the fact that EdgeWrite had not yet undergone the formal guessability study (§3.3.3). For EdgeWrite, subjects had to use the character chart for an average of 4.6 total characters out of 1000 characters entered. For Graffiti, this average went up to 6.6. This comparison did not yield a significant result ($F_{1,152}$=1.72, ns).

Another measure of learnability is the amount of time spent on each trial. EdgeWrite and Graffiti were surprisingly similar. Over the 20 trials, mean times were nearly identical, at 118.2 seconds for EdgeWrite to 117.1 seconds for Graffiti ($F_{1,152}$=0.00, ns).

4.3.1.6 Discussion

Both EdgeWrite and Graffiti were quickly learnable by people who had never used either technique. Although this result was known for Graffiti (MacKenzie and Zhang 1997), at the time of this study it was not yet known for EdgeWrite. The main significant finding was that Graffiti was 18.2% more error prone than EdgeWrite during entry, although both methods produced similarly error-free transcriptions. Although there was not a significant speed difference, Graffiti did have significantly faster characters from pen-down to pen-up. However, EdgeWrite had fewer overall strokes as measured by GPC. It would thus seem that the two techniques compare rather evenly for able-bodied novices, except for EdgeWrite's significantly higher accuracy during entry.

4.3.2 Novice Motor-impaired Use

Obtaining statistical results with subjects who have motor impairments is often infeasible because of the high variance among people with disabilities, the difficulty in obtaining subjects, and the small amount of testing each subject can do before growing tired (Coyne 2005, Feng *et al.* 2005). Therefore, the text entry accuracy of just five users with various motor impairments was studied for both EdgeWrite and Graffiti or Graffiti 2. It was important to isolate a subject's ability to *perform* a letter from his or her *learning* of that letter, so a note card with each stroke drawn on it was used as a prompt. Subjects were given as much time as they needed to make each stroke.

Subject 1 was a middle-aged woman with Parkinson's disease and severe tremor. She was asked to enter each lowercase letter (*a–z*) and each digit (0–9) twice in Graffiti and then twice in EdgeWrite. Her recognition rates were 22/72 (30.6%) in Graffiti and 68/72 (94.4%) in Stylus EdgeWrite. In Graffiti, interspersed with numerous misrecognitions were a number of accidental periods and other punctuation due to "bounce" on the screen. Her tremor made it difficult for her to avoid setting the punctuation mode in Graffiti. When she entered a 50 character phrase with both methods, she made over 3 times the number of errors in Graffiti as she did in EdgeWrite.

Subject 2 was a 30-something woman with Spastic Cerebral Palsy. When asked to make each Graffiti letter and number, it turned out there were some she could never correctly produce despite repeated efforts: "c", "e", "f", "x", "2", and "4". For EdgeWrite, these were limited to just "2" and "7", probably owing to their use of diagonals. When she attempted the sentence, "The dog is going fast" in Graffiti she produced, "The g i gbsiangu% fast" (8 errors). She skipped "d" after repeated misrecognitions because it was too difficult. In EdgeWrite, she entered the sentence without leaving any errors. Her only complaint was that the diagonals were difficult. This prompted the later addition of alternative strokes that lacked diagonals. In the final

EdgeWrite alphabet (Appendix A), most letters and numerals can be entered without diagonals.

Subject 3 was a 30-something male with a form of Muscular Dystrophy. He had a small baseline tremor. Using Graffiti, he was unable to make “b”, “d”, and “f” after multiple tries and gave up on them. Using EdgeWrite, there were not any letters that he could not produce. In fact, in an attempt to do 3 versions of each EdgeWrite alphanumeric character, he made only 2 errors in 108 entries. When asked to produce the sentence, “the ugly underwater urchin eats putrid meat lovingly” in Graffiti he produced, “the ugl un erwater urchin eats putri meat lovingly” (3 errors). In EdgeWrite he produced the sentence without leaving any errors.

Subject 4 was a 40-something woman with Cerebral Palsy. She was able to enter all letters and numbers in both Graffiti and EdgeWrite, but upon switching to EdgeWrite from Graffiti, she exclaimed, “This is a lot easier than before. Much easier, my goodness.” She entered a 50 character phrase in Graffiti leaving 2 errors and in EdgeWrite without leaving any errors.

Subject 5 was tested two years after these other subjects; as a result, he used Graffiti 2 instead of Graffiti. He also used the EdgeWrite alphabet that was the result of the guessability study and subsequent refinements (§3.3.3). This subject was 50 years old and had had a spinal cord injury for 15 years. Although he did not have tremor, he had low strength and poor coordination in his hands and arms. Before using Graffiti 2, he trained the recognizer by entering sample strokes for “p”, “t”, “y”, and “$”. Then he entered each letter four times in Graffiti 2 and then in EdgeWrite (i.e. “aaaa”, “bbbb”, …, “yyyy”, “zzzz”). In Graffiti 2 his accuracy was 57/104 (54.8%). In four attempts, he could not successfully make “b”, “d”, “e”, “g”, “m”, and “w”. In EdgeWrite, his accuracy was 103/104 (99.0%). He only missed one attempt at “o” because he failed to enter the bottom-left corner. EdgeWrite was so successful for this subject that he said he felt he could finally use a PDA, which he had never been able to use before but wanted to.

Figure 4.10 shows the accuracy results of these five subjects. A two-tailed Wilcoxon signed-rank test revealed a trend in favor of EdgeWrite over Graffiti (z=-7.50, p=0.063).

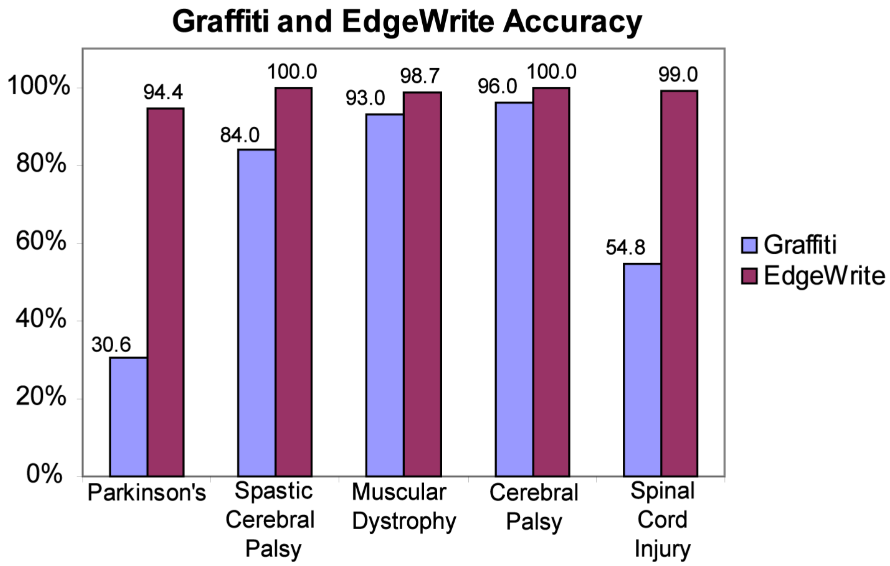


Figure 4.10. Graffiti and EdgeWrite accuracy for five subjects with motor impairments. Accuracy was measured independent of memorization and speed.

4.3.3 Expert Able-bodied Use

Stylus EdgeWrite has not been formally tested with a range of able-bodied experts. However, I have rigorously tested myself (an expert) on a variety of devices (Wobbrock and Myers 2005a), including character-level Stylus EdgeWrite. Over 10 phrases, my own speed averaged 24.0 WPM with 2.8% total errors, 0.0% of which went uncorrected—that is, all errors were corrected.

For informally evaluating Fisch, I conducted a self-test of “record speed” (Kristensson and Zhai 2004). Accordingly, I chose a random phrase from a text entry phrase set for repeated entry (MacKenzie and Soukoreff 2003). The phrase was, “for your information only”, which in Fisch can be entered in 6 fluid strokes. I entered the phrase 33 times in 7 minutes. Over the first 5 tries, my speed averaged 18.0 WPM. On the 10th try, my speed increased to 31.0 WPM; on the 15th, it was 55.7 WPM. My speed on the final try was 63.3 WPM. There were no errors left in any of the entered phrases.

Fisch has been more extensively tested during evaluations of Trackball EdgeWrite (§7.3.4) and Isometric Joystick EdgeWrite (§8.3.4).

Chapter 5

Joystick EdgeWrite[*]

5.1 Motivation

Displacement joysticks have served as input devices since the earliest computers (Herz 1997). The two-player version of *Computer Space* (1972), the first coin-operated arcade game, used two mounted joysticks. In 1978, Atari released its first game console, the Atari 2600, which had no keyboard, just a joystick. Joysticks have been studied in human-computer interaction since at least the seminal study by Card *et al.* in 1978 (Card *et al.* 1978). Yet despite joysticks' considerable tenure, no satisfying text entry techniques have been developed for them. The methods that do exist are mostly selection-based; they require screen real-estate to display options, are difficult to use without looking, are hard to customize, and are slow, requiring many movements per character.

Today's computer game industry might benefit from better text entry for game consoles, which often have only game controllers as input devices. If they have keyboards at all, they are sold separately at extra cost. Many game consoles are now networked, and require extensive text entry during configuration before they allow game play. For example, registration for the *Xbox Live!* service requires entering personal and billing information and can take more than 30 minutes using a joystick and an on-screen

[*] Parts of this chapter are adapted from (Wobbrock *et al.* 2004b, Wobbrock *et al.* 2004a).

selection keyboard. Furthermore, many networked games allow for communication among players using short bursts of instant messenger-style text. With only selection-based text entry methods for game controllers, this can be awkward.

Mobile devices have also placed high demands on text entry development. Numerous text entry methods have been investigated, including those driven by buttons, character recognition, virtual keyboards, thumbwheels, and voice. Many handheld devices, such as the Ericsson T68i mobile phone, are equipped with miniature joysticks for navigation and selection purposes, yet have no capability for joystick text entry. Joystick text entry on mobile devices reduces the need for screen areas devoted to stylus entry, for virtual keyboards that take up precious screen real-estate, and for multiple button-taps to select desired characters. They also can be used without looking, which may have positive implications for eyes-free use.

Another potential use of joystick text entry is for users of power wheelchairs (§6). Technology is already commercially available to enable a person to control a computer's mouse from a power wheelchair joystick, but options for text entry are limited to mouse-based selection techniques, like the WiVik on-screen keyboard (Shein *et al.* 1991).

This chapter presents a new joystick text entry method that is not based on selection, but on gestures using the EdgeWrite alphabet. The properties of this alphabet make it well-suited for text entry with joysticks. An experiment with able-bodied users shows that joystick EdgeWrite is faster, produces more accurate phrases, and is more satisfying to users than *date stamp* or *selection keyboard*, two prevalent selection-based methods.

5.2 Design

This section describes the design of Joystick EdgeWrite and the particular issues faced in the development of this version. Important differences from Stylus EdgeWrite (§4.2) exist in the design of Joystick EdgeWrite's segmentation scheme and corner regions.

5.2.1 Some Challenges of Writing with a Joystick

Displacement joysticks, like those found on game controllers (Figure 5.1), commonly operate in one of two modes: position-controlled or rate-controlled. With position-control, or "absolute mode," the physical range of the joystick is mapped to a plane (e.g. the screen), and the position of the stick corresponds to a position in the plane. Joystick-driven screen magnifiers have been designed using position-control (Kurniawan *et al.* 2003). In contrast, with rate-control joysticks, the further the stick is moved from its center, the faster the position or view changes. Rate-control, or "relative mode," is common in "first-person shooter" games and for joystick-controlled mouse cursors (LoPresti *et al.* 2004).

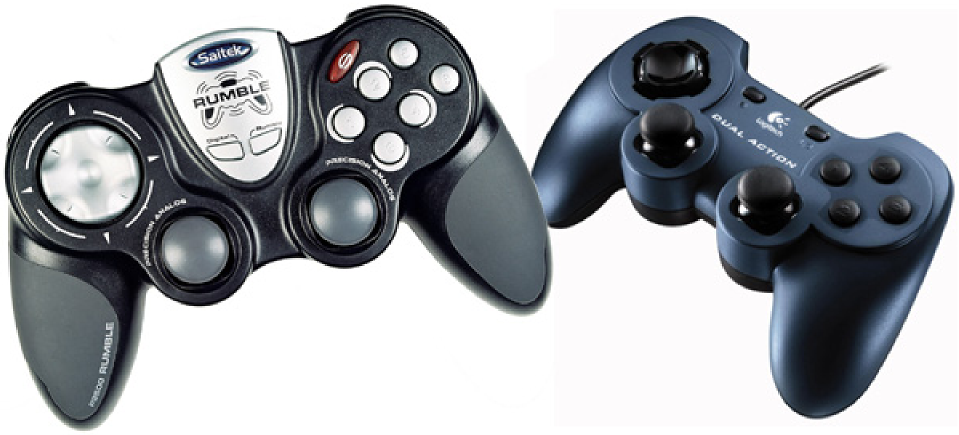


Figure 5.1. The Saitek P2500 and Logitech Dual Action Gamepad, both of which have square areas around their thumb-controlled sticks. If released, both sticks snap to center.

It would seem that position-control might be the ideal candidate for "writing" with a displacement joystick, as one could trace an (*x*, *y*) path much like one does with a stylus on a PDA. Many studies, however, confirm that joysticks are not particularly accurate for positioning as mice, trackballs, touchpads, and tablets (Epps 1987, Murata 1991, MacKenzie *et al.* 2001a). Indeed, our design explorations confirm the difficulty of making smooth letter-forms using a joystick. The prospect of writing in an alphabet like Graffiti is therefore dubious. If gestures are to be used, they will have to be designed to overcome this difficulty.

Human physiology also complicates joystick text entry. For example, the dexterity of the thumb changes with its position relative to the hand, causing changes in range of motion (Hirotaka 2003). The index finger has the highest Fitts' index of performance (Langolf *et al.* 1976), making it better suited than the thumb for control tasks (Ehrlich 1997). The velocity of a writer depends on whether she moves her arm or only her wrist, and upward strokes are generally faster than downward ones (Isokoski 2001). Some results show that humans have a difficult time returning a joystick to the same position it was before (Kurniawan *et al.* 2003). Such variables may subvert any attempts at joystick writing. While not a panacea, EdgeWrite is well-suited to overcoming these challenges.

5.2.2 The Suitability of EdgeWrite

EdgeWrite has properties that make it well-suited to meeting the challenges of joystick text entry. All motion in EdgeWrite is ideally in straight lines between corners, but straight lines are not required for good recognition, since recognition depends not on the path of movement but instead on the sequence of corners that are hit (§3.2).

Displacement joysticks are usually best used for control, not positioning, but EdgeWrite's use of stabilizing physical edges allows a joystick bounded by a physical square to be used in position-control mode for writing EdgeWrite characters. The areas bounding the thumb joysticks on the Saitek P2500 and many Logitech game controllers are squares (Figure 5.1). In the user study presented below (§5.3), the Saitek P2500 was used without modification.

With a displacement joystick, it is difficult to make smooth curvaceous characters like those required by Graffiti, but EdgeWrite characters are easy to make by pushing the stick from corner to corner along the physical edges. Edges naturally guide the stick, and corners naturally pocket it. EdgeWrite characters begin in one of four corners, easily accessed from the center of a square with a self-centering joystick.

5.2.3 Interaction Design

To understand how EdgeWrite works with a displacement joystick, we must understand how EdgeWrite partitions the joystick's coordinate plane. Using C# and DirectInput, the joystick is polled for its position every 55 ms, which proves sufficiently often. The (*x*, *y*) position of the stick falls within the range of the [-100 … +100] and centers at (0, 0). In practice, none of the joysticks surveyed centered perfectly; some were off by up to ±20.

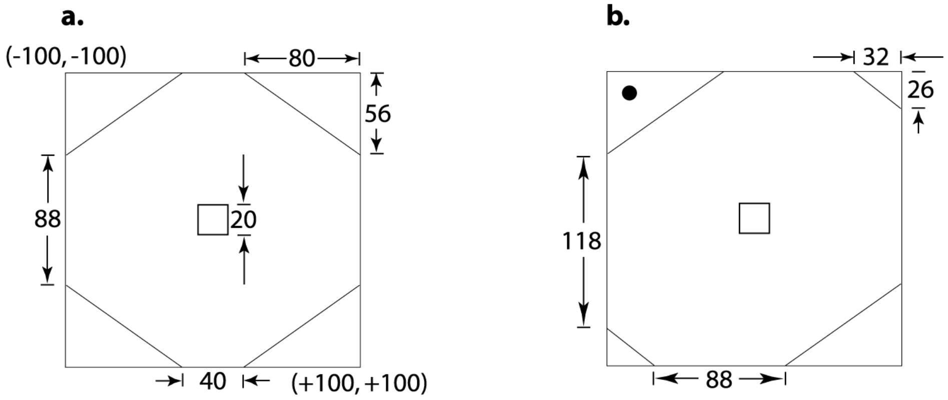


Figure 5.2. (a) The partitioning of the Joystick EdgeWrite input plane. (b) The "deflation" that occurs after a corner has been entered by a right-handed user.

Like in Stylus EdgeWrite (§4), Joystick EdgeWrite's corners are triangular regions so that diagonal strokes do not accidentally hit them. In addition, some corners "deflate" into smaller triangles to make diagonals easier to perform. Figure 5.2a shows the inflated dimensions of the joystick coordinate plane for a right-handed user. Figure 5.2b shows the deflated dimensions of the joystick coordinate plane. (The dot in the top-left indicates the current joystick position.) For a right-handed user, the deflated dimensions take hold whenever the joystick is in the top-left or bottom-right corners, which makes the difficult diagonals easier (Figure 5.3).

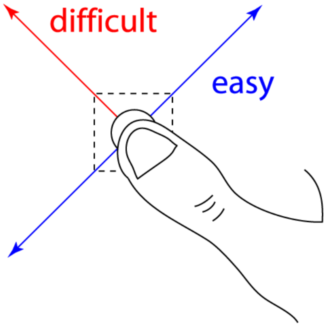


Figure 5.3. When on the joystick, a right-hand thumb finds one diagonal easy and the other more difficult. Deflation helps make the difficult diagonal easier.

The difficulty shown in Figure 5.3 arises because the thumb's dexterity and range of motion along the easy diagonal is much better than along the difficult diagonal. The easy diagonal is along the natural arc of the thumb, while the difficult diagonal is along the length of the thumb itself.

A design challenge is how to segment between letters. In unistroke text entry with a stylus, a pen-down event starts a character and a pen-up event ends it. There is no analog to this with a joystick. Versions that used button presses and center dwell-time for segmentation were built, but both proved awkward and slow. Instead, Joystick EdgeWrite segments characters by starting a character when a corner is entered, and ending it when the polling of the joystick yields two successive points in the center (Figure 5.4). From a user's perspective, this means momentarily relaxing the joystick just enough so that it naturally returns to its center. With this scheme, annoying pauses are not necessary between characters, as they are with center dwell-time segmentation. In user testing, users did not notice any delays, and there were no observed segmentation errors in thousands of entered letters.

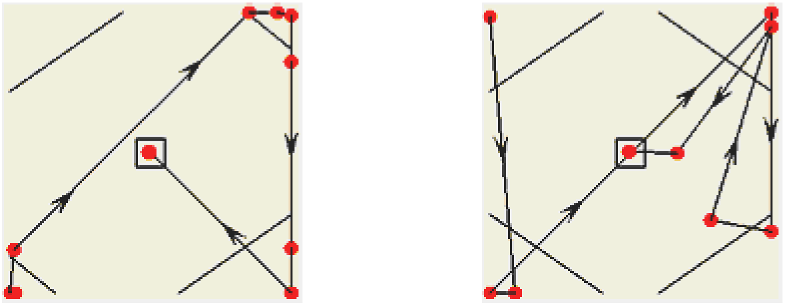


Figure 5.4. A clean trace of an “a” (left) and a sloppy but recognized trace of “w” (right). The “w” is sloppy because it fails to snugly impact the bottom-right corner.

Note the relatively few points actually sensed by the polled joystick for the letters in Figure 5.4. The “a” on the left has only 10 input points, but 7 of them fall within corners. The “w” on the right has only 9 input points, but 6 of them fall inside corners. This is because corners naturally “pocket” the joystick, slowing it momentarily and improving the chance that the stick position will be polled there. These examples highlight the minimal sensing necessary for a functional version of Joystick EdgeWrite.

5.3 Evaluation

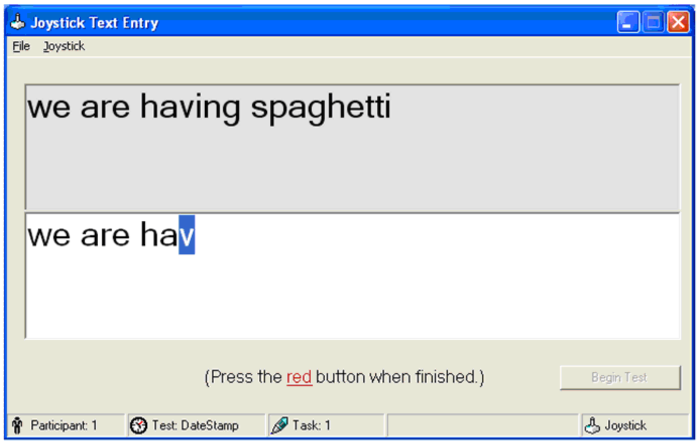
Joystick EdgeWrite was evaluated for both novices and experts against competitor methods. Like in the evaluation of Stylus EdgeWrite (§4.3), this evaluation used the EdgeWrite alphabet before the guessability study (§3.3.3). Also, continuous recognition feedback (§3.4.1) and non-recognition retry (§3.4.2) had not yet been implemented, and slip detection (§3.4.3) was only ever implemented for the “virtual” versions on the trackball and isometric joystick.

5.3.1 Competitor Methods

Although joystick text entry is not particularly common, two predominant methods exist: *date stamp* and *selection keyboard*. These methods were used as competitors to Joystick EdgeWrite. Both are described in general terms in §2.3.4. They are described here as they were implemented for the current study.

5.3.1.1 Date Stamp

Joystick-based date stamp is often implemented with a predetermined number of letter-slots (i.e. “stamps”) and the inability to backspace. This implementation is feasible for high-score screens in arcade games, but not for text entry studies where a presented phrase must be transcribed. Therefore, for the purposes of this study, no predetermined letter-slots were used. Instead, the user receives a new stamp, initialized with “a” and highlighted in blue, when he or she moves the joystick to the right (Figure 5.5). Moving the joystick to the left erases the previous letter and initializes the stamp *with that letter*. Thus, after backspacing a letter, a user is not forced to start from “a” again, but from the erased letter, making slight under- and overshoots easy to fix. As usual, moving the stick down cycles forward through the alphabet, and moving it up cycles backward.



we are having spaghetti

we are hav

(Press the red button when finished.)

Figure 5.5. The joystick test software showing date stamp with “v” currently highlighted.

The stamp employed in this experiment used Tarasewich’s thumbwheel sequence except without punctuation (Tarasewich 2003). The sequence is [*space*][*a..z*][0..9](repeat). If users hold the stick up or down, the date stamp cycles after an initial pause of 390.6 ms with a repeat delay of 62.5 ms. These values were taken from keyboard key-repeat times.

5.3.1.2 Selection Keyboard

The selection keyboard used in the study is shown in Figure 5.6. With it, a user can move the selector up, down, left, or right, but not diagonally. When the user presses a joystick button, the currently highlighted key is committed. When a key is committed, the halo remains where it is and does not jump to a home position. The halo can wrap around the keyboard horizontally or vertically, staying in the same row or column. Key-repeat behavior, identical in timing to the date stamp method, governs rapid movement of the halo. The layout in Figure 5.6 is copied from the Xbox *Live!* registration sequence and two popular Xbox games, *Halo* and *Brute Force*. Note that selection keyboards differ from point-and-click on-screen keyboards (§2.3.6) in that no mouse cursor is used and virtual keys cannot be missed. Of course, the wrong key can still be pressed.

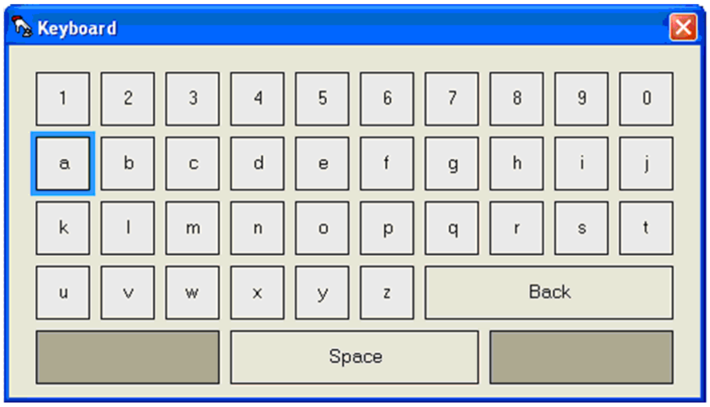


Figure 5.6. The selection keyboard used in the study. It replicates the design in use on the popular Xbox game console. The letter "a" is currently highlighted.

5.3.2 Novice Use

The first of two studies was a single session experiment involving novice users.

5.3.2.1 Subjects

Eighteen subjects from the nearby university communities were recruited. The median age was 21.5. Four were female and 1 was left-handed. Subjects were paid $20 for a 90-minute test in which they entered text using date stamp, selection keyboard, and EdgeWrite. No subjects had any prior experience with EdgeWrite.

5.3.2.2 Apparatus

Tests were conducted in a laboratory using an 866 MHz Pentium 3 machine running Windows XP with 256 MB RAM. Attached to it was a 16"×12.4" Hitachi monitor set to 1280×1024 resolution and 32-bit color. The test software shown in Figure 5.5 was implemented in Visual C# using DirectInput 9.0b. The font was Microsoft Sans Serif 24-point, and the joystick was a Saitek P2500 Rumble Force Pad (Figure 5.1). Importantly, this joystick has a physical square boundary around its stick.

Quantitative data were logged by the test software and then analyzed according to the measures in §2.4. These measures included speed in words per minute (WPM) and accuracy as corrected, uncorrected, and total error rates (§2.4.3). In addition, raw data rates were measured in keystrokes per second (KSPS). Joystick movements were also logged. Subjective data was obtained with a post-test questionnaire.

5.3.2.3 Procedure

Subjects used date stamp, selection keyboard, and EdgeWrite in a single-factor within-subjects design. The entry methods were assigned to subjects in a fully counterbalanced manner in order to neutralize learning effects and fatigue. Analyses of variance for *Method Order* show no significant differences, indicating the counterbalancing worked.

Subjects practiced each method immediately before testing with it. Practice was designed to provide the minimum amount of proficiency needed to perform the technique. For date stamp and selection keyboard, this was just a single phrase (about 30 letters), since these methods were trivial to learn. For EdgeWrite, this was 10 phrases, then each letter 3 times, then 2 more phrases, which took ~15 minutes.

Admittedly, practice for EdgeWrite was more extensive time-wise than practice for the selection-based methods. This is simply because the selection-based methods were easier to learn than EdgeWrite. Also, subjects quickly became bored with the selection-based methods, and requiring equal practice times among the three techniques would have caused undue boredom and fatigue. Section 5.3.3 contains results for users highly practiced in all three techniques. These results show that more practice with the selection-based methods does not result in improved performance for the selection-based methods, but it does for EdgeWrite.

Testing consisted of a fixed set of 10 phrases with each method. Phrase set assignment was even across entry methods to prevent bias. Subjects were instructed to proceed "quickly and accurately" while testing (Soukoreff and MacKenzie 2003).

5.3.2.4 Trials

A single trial consisted of entering one phrase. The phrases came from a published phrase set (MacKenzie and Soukoreff 2003). While the practice phrases were chosen at random from a set of 500, test phrases were fixed in sets of 10 and assigned evenly to each entry method. Table 5.1 shows phrase set characteristics.

Set	Phrases	Words	Characters	Correlation with English
1	10	61	297	89.9%
2	10	52	298	92.7%
3	10	55	298	86.8%

Table 5.1. Characteristics of the three sets of test phrases.

Consistent with MacKenzie and Soukoreff's reasoning, numbers were not tested, although they were implemented for each method because numbers are common in real-world text entry and should be present even if untested.

5.3.2.5 Adjustment to Data

The data were analyzed using a repeated measures mixed model analysis of variance with a fixed effect for *Method* and a random effect for *Subject* (Littell *et al.* 1996). Contrasts between trials 1–5 and trials 6–10 for each method's speed showed no significant differences for EdgeWrite and date stamp, suggesting that subjects had somewhat stabilized prior to testing. But such a contrast *did* show a significant difference for selection keyboard ($F_{1,493}$=8.51, p<.01), suggesting that subjects were still speeding up during testing. Most of this speed-up was on the first trial. When it is removed, these contrasts no longer show significant speed-ups for any method. Thus, all reported analyses are for trials 2–10.

5.3.2.6 Speed

Means and standard deviations for speeds in WPM are: EdgeWrite 6.40 (σ=1.60), selection keyboard 6.17 (1.18), and date stamp 4.43 (0.62). For comparisons, these are graphed with keystrokes per minute (KSPM) in Figure 5.8.

An omnibus F-test of WPM is significant for *Method* ($F_{2,466}$=217.20, p<.01). Contrasts show that EdgeWrite is faster than selection keyboard ($F_{1,466}$=5.11, p<.025) and date stamp ($F_{1,466}$=363.80, p<.01). Selection keyboard is also faster than date stamp ($F_{1,466}$=282.69, p<.01).

5.3.2.7 Error Rates

Figure 5.7 shows three error rates—uncorrected, corrected, and total errors (§2.4.3)—for the joystick text entry methods date stamp, selection keyboard, and EdgeWrite.

Omnibus F-tests of all three error rates are significant for *Method* (p<.01). Contrasts show that Joystick EdgeWrite has a higher corrected error rate than selection keyboard ($F_{1,466}$=132.16, p<.01) and date stamp ($F_{1,466}$=73.61, p<.01). Surprisingly, however, the opposite is true for uncorrected error rates. EdgeWrite produces significantly *more* accurate phrases than selection keyboard ($F_{1,466}$=6.24, p<.02), and nearly so compared to date stamp ($F_{1,466}$=3.68, p=.055). This discrepancy is discussed below.

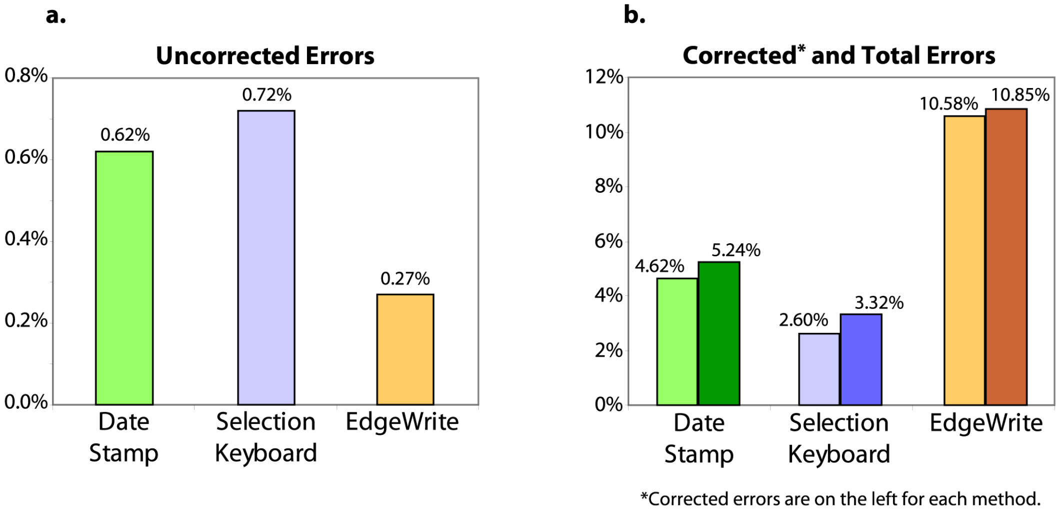


Figure 5.7. Uncorrected errors, corrected errors, and total errors for three methods.

Participant conscientiousness is a ratio of corrected errors to all errors (Soukoreff and MacKenzie 2003). A score of 1.0 indicates a subject corrected all errors; a score of 0.0 indicates all errors were left in the transcribed string. Means and standard deviations are: EdgeWrite 0.98 (0.09), selection keyboard 0.92 (0.26), and date stamp 0.89 (0.28). An omnibus F-test is significant ($F_{2,466}$=7.39, p<.01). Contrasts show that subjects were more conscientious with Joystick EdgeWrite than selection keyboard ($F_{1,466}$=7.15, p<.01) and date stamp ($F_{1,466}$=13.91, p<.01). Date stamp was not detectably different than selection keyboard.

Thus, despite making more errors during entry (Figure 5.7b), subjects' transcriptions had fewer errors with Joystick EdgeWrite (Figure 5.7a), because subjects were more thorough in correcting errors as they went. Although one should minimize both corrected and uncorrected errors in any text entry method, uncorrected errors are the more damaging of the two, since they are at odds with speed (§2.4.3). Thus, Joystick EdgeWrite produces more accurate text in less time, albeit with more errors made (and fixed) along the way.

5.3.2.8 Data Rate

Speed in WPM only considers the amount of text in the transcribed string and the time it took to produce. It is also interesting to consider the amount of *data* transmitted from the text entry device to the computer over time. This can be done by considering the *input stream*, which includes all entered characters, even those erased and the backspaces that erase them. Building off the KSPC acronym for *keystrokes per character* (Soukoreff and

MacKenzie 2001), data rate can be called *keystrokes per second* (KSPS). The KSPS metric gives us an idea of how fast users produce characters, regardless of how correct those characters are. Note that KSPS only considers transmitted characters, not selector movement in the case of the selection-based methods.

Means and standard deviations for KSPS are: EdgeWrite 0.66 (0.14), date stamp 0.41 (0.08), and selection keyboard 0.54 (0.10). An omnibus F-test of KSPS for *Method* is significant ($F_{2,466}$=361.03, p<.01). Contrast tests show that EdgeWrite is faster than selection keyboard ($F_{1,466}$=154.34, p<.01) and date stamp ($F_{1,466}$=720.72, p<.01). The selection keyboard is also faster than date stamp ($F_{1,466}$=208.02, p<.01).

We can use KSPS to compute an upper bound for WPM by assuming all transmitted characters were correct and persisted unto the final transcribed string. In this case, Joystick EdgeWrite's speed increases 23.0% from 6.40 to 7.87 (Δ=1.47), selection keyboard 5.3% from 6.17 to 6.50 (Δ=0.33), and date stamp 10.8% from 4.43 to 4.91 (Δ=0.48). Thus, EdgeWrite seems to have more potential than the selection-based methods when accuracy is improved, as would be the case with more practice. This is supported by our results for users with more practice (§5.3.3).

For the purposes of comparing wpm to KSPS, we can convert KSPS to keystrokes per minute (KSPM) with the usual definition of a word as 5 characters (Figure 5.8).

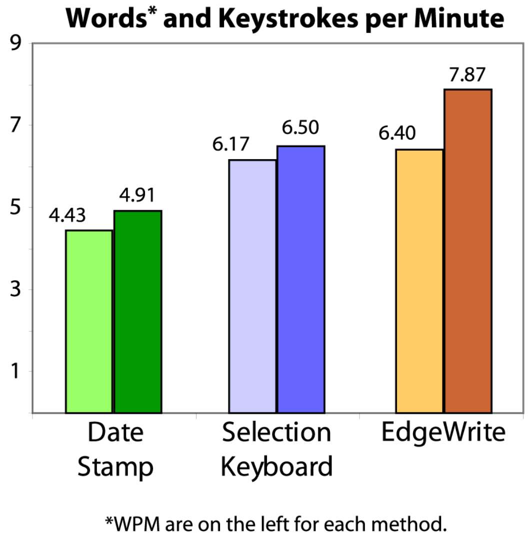


Figure 5.8. Speeds and data rates in WPM and KSPM for three joystick methods. Whereas WPM considers only the length of the transcribed string, data rate in KSPM is the speed with which all characters are transmitted, including backspaces.

5.3.2.9 Gestures

Non-recognitions sometimes occur with gestural interaction techniques. We can compare the number of EdgeWrite gestures made to the number recognized. The average number of gestures made per trial was 42.15 (9.73). The average number of gestures recognized per trial was 38.07 (7.92). Thus, about 10.6% of EdgeWrite gestures were unrecognized. If all gestures had been recognized and were correct, EdgeWrite's WPM would be 8.65 (1.75). This rate represents perfect performance given novice speeds.

5.3.2.10 Selector Movement

For date stamp and selection keyboard, it is interesting to compare the path of selector movement to the minimal selector path. This minimal path is trivial to compute for date stamp: for each letter, spin whichever direction (up or down) reaches the target letter first. For selection keyboard, the minimal path for a phrase can be found using any optimal path-finding algorithm, such as A*.

On average, subjects rotated through 329.81 (64.94) characters per trial in date stamp. The minimal path required an average of 271.77 (46.25) rotations per trial. Thus, subjects rotated about 21% more than necessary.

On average, subjects moved the selection keyboard selector 137.51 (25.27) times per trial. The minimal path required an average of only 93.19 (13.45) movements per trial. Thus, subjects moved the halo about 48% more than necessary. Note that the optimal path included using the wrap-around feature of the keyboard.

5.3.2.11 Questionnaire Results

Subjects were given four Likert scales (1–5) on which to rate the three entry methods. Figure 5.9 shows that subjects liked Joystick EdgeWrite better than the other two methods. They also felt it was easier, more enjoyable, and faster. A repeated measures analysis of variance of subjects' average Likert scores yields a significant difference ($F_{2,34}$=11.37, p<.001). Contrasts show that EdgeWrite was preferred over date stamp ($F_{1,34}$=14.39, p<.001) and selection keyboard ($F_{1,34}$=19.35, p<.001). No significant difference exists between preferences for date stamp and selection keyboard ($F_{1,34}$=0.37, ns).

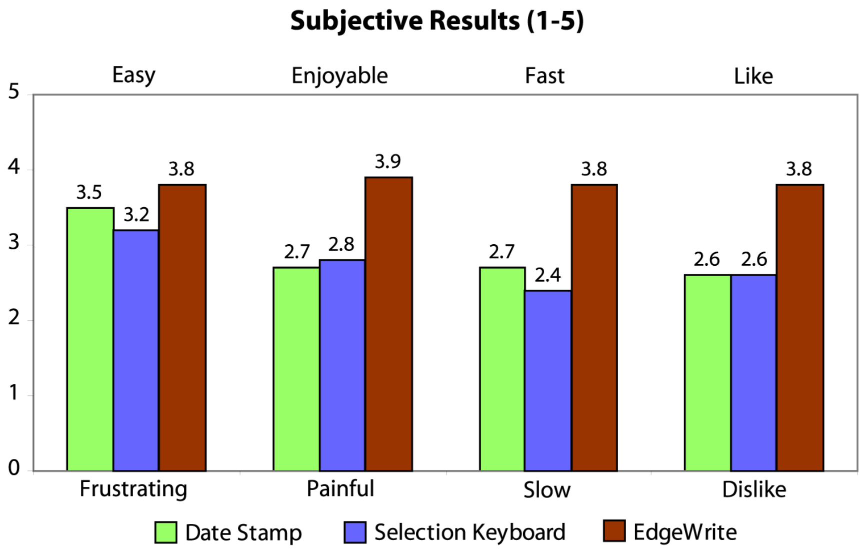


Figure 5.9. Subjective results for the three methods. Scales range from 1–5 (worst-best).

5.3.2.12 Discussion

One might expect a recognition-based method to be less accurate *during* entry than a selection-based method because of gestural slips and mistakes. But it is interesting that, despite these errors, novices produced *more* accurate phrases with Joystick EdgeWrite than with the other methods, and did so in less time. The selection keyboard requires a second point of visual focus besides the transcribed text, so it is conceivable that subjects may have left errors because they were attending to the keyboard and not the transcribed text. It also may be the case that it is simply too cumbersome to fix errors with the selection keyboard.

On average, however, EdgeWrite also produced more accurate phrases than date stamp, which requires no additional focus of attention and supports easy error correction. Perhaps with Joystick EdgeWrite, subjects feel quicker to remedy errors, or feel more “engaged” with their input than with the fairly monotonous selection-based methods. Or, perhaps because of the high error rate during entry, subjects are more vigilant in correcting errors.

Surprisingly, subjects felt that selection keyboard was the slowest of the three methods, even though date stamp was far slower. Subjects also felt selection keyboard was the most frustrating of the methods. Subjects’ comments indicated that this was due to the visual attention required.

In this study, Joystick EdgeWrite was faster than selection keyboard by a small margin with many novices over multiple trials. Although this result was statistically significant, it should be regarded as the *minimum* amount of practice (~15 minutes) required by a beginner to become reliably better with EdgeWrite than with selection keyboard. Any further practice, as our results for expert users will show in the next section, only increases EdgeWrite's advantage.

5.3.3 Expert Use

To see how more practiced users fared with Joystick EdgeWrite, 3 more subjects were tested. These subjects were already familiar with Stylus EdgeWrite (§4) and the EdgeWrite alphabet, but not with Joystick EdgeWrite. They practiced with all three joystick text entry methods for over 30 minutes, targeting difficult letters and entering many phrases in each method. The performance of these 3 users with date stamp and selection keyboard was near to that of the novices in the main study (Figure 5.10, Figure 5.11). Put simply, there simply wasn't much room to improve. But with Joystick EdgeWrite, these users were faster and less prone to errors than the novices before them. The three experts made 13.2% fewer gestures: 36.60 per trial compared to 42.15 for novices.

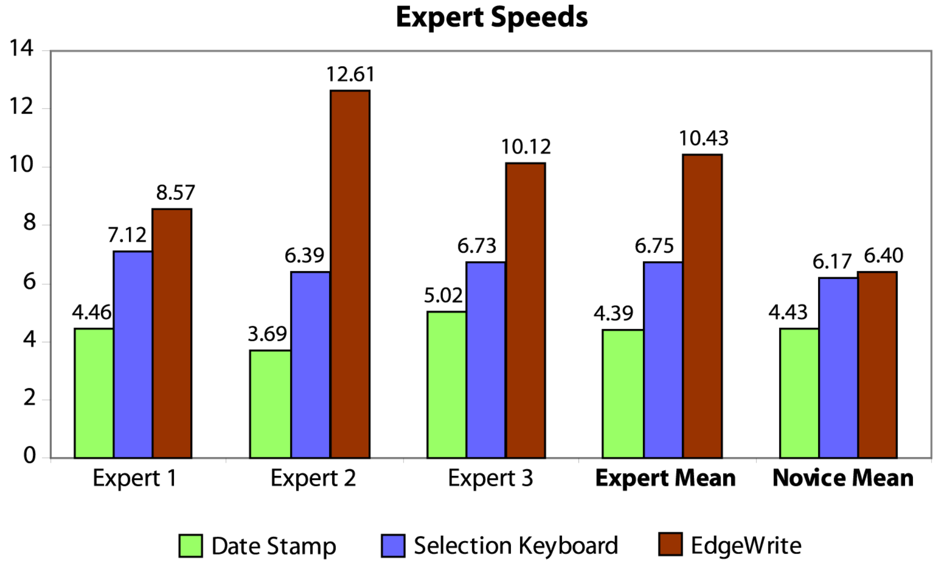


Figure 5.10. Speeds of three experts with all three joystick methods. Means from novices are included for comparisons.

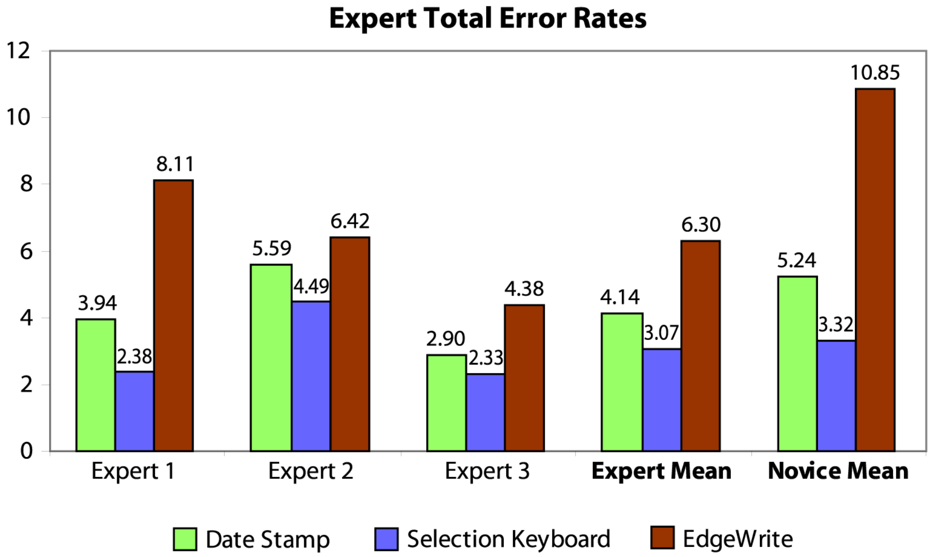
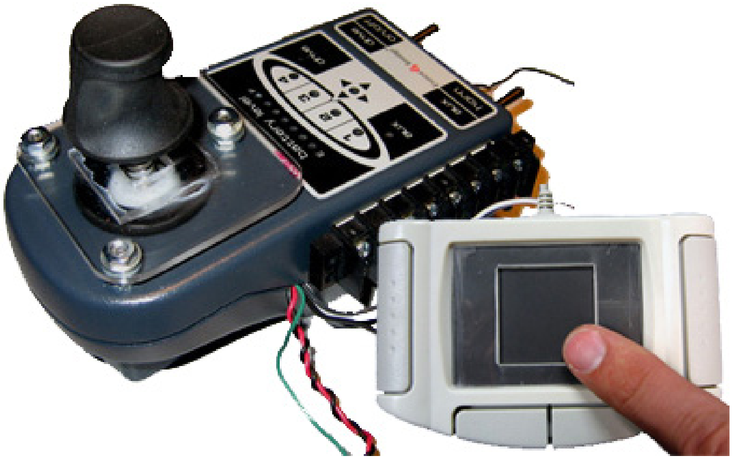


Figure 5.11. Total error rates of experts in all three joystick methods. Means from novices are included for comparisons.

I have also tested myself on a variety of devices (Wobbrock and Myers 2005a), including Joystick EdgeWrite. Over 10 phrases, my own speeds averaged 14.7 WPM with 8.8% total errors.



Chapter 6

EdgeWrite for Power Wheelchairs*

6.1 Motivation

Inspired by the technology and results from Joystick EdgeWrite (§5), we began to investigate how to improve text entry for people in power wheelchairs, since power wheelchairs also use displacement joysticks. Many people with motor impairments use power wheelchairs. An estimated 1.4 million people in the USA depend on wheelchairs for mobility (Kraus *et al.* 1996). Of these, about 10% are in power wheelchairs, about half of whom require more than one assistive technology to participate in daily activities (Cook and Hussey 2001).

As noted in §2.3.4, commercial technology already exists for enabling mouse cursor control from a power wheelchair joystick, for example, the *Mouse Driver* from Switch-It, Inc. (http://www.switchit-inc.com). But mouse control is only part of a computer access solution. The ability to enter text is also a cornerstone of successful human-computer interaction. However, an integrated text entry method to accompany joystick mouse control is not yet available. Instead, text entry from power wheelchairs usually takes the form of point-and-click or point-and-dwell with an on-screen keyboard (Figure 6.1).

* Parts of this chapter are adapted from (Wobbrock *et al.* 2004c, Wobbrock *et al.* 2005a, Wobbrock and Myers 2006c).

Unfortunately, such text entry is unsatisfactory for a number of reasons (§2.3.6). Alternatively, some people with motor impairments use speech recognition, but these technologies have high abandonment rates due to technical difficulties, the need for maintenance, and poor recognition rates (Koester 2003). In contrast, an *integrated* text entry method for power wheelchair joysticks would give fuller access without requiring additional devices (Guerette and Sumi 1994, Spaeth *et al.* 1998).

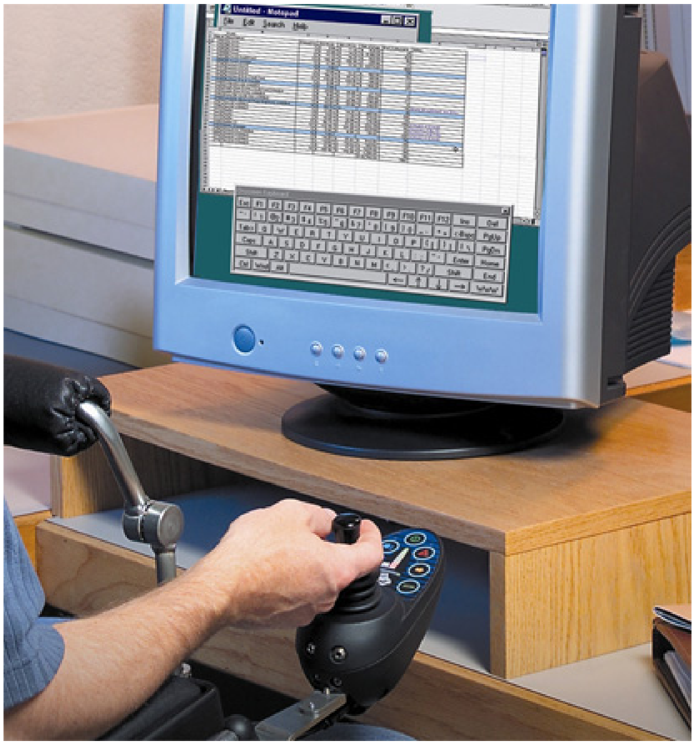


Figure 6.1. The Mouse Driver system from Switch-It, Inc. enables a wheelchair joystick to control a mouse cursor on a desktop PC. Unfortunately, text entry reduces to pointing with an on-screen keyboard. Image from http://www.switchit-inc.com.

Although they are less common than joysticks, touchpads also can be used to control power wheelchairs. An example is the *Touch Drive* system from Switch-It, Inc. (http://www.switchit-inc.com). With it, a person's finger acts as a rate controlled joystick: the farther it moves from the touchpad's center, the faster the wheelchair turns or moves in that direction.

Touchpads require less strength to operate than joysticks and little or no calibration. However, touchpads have not generally been considered text entry devices. People in power wheelchairs might benefit from an integrated device that could control their chair, mouse, and text entry solution. This requires a versatile text entry method for touchpads.

As further motivation, public information terminals are appearing in building lobbies, libraries, bus stations, and community centers. Accordingly, the ability to access these terminals is becoming more important. Just as the Americans with Disabilities Act requires that many buildings have access ramps, future information kiosks may be required to be accessible electronically via Bluetooth or another wireless technology. It would be advantageous to have an integrated control system where the power wheelchair joystick or touchpad could be used as the input device for mousing *and* text entry on such terminals.

The last chapter (§5) presented a design for Joystick EdgeWrite targeted mainly at game controllers and recreational joysticks. This chapter presents *PW-Joystick EdgeWrite*, a similar design for the Everest & Jennings 1706-5020 power wheelchair joystick. It also presents a design for Touchpad EdgeWrite using a Synaptics touchpad. These two methods were evaluated with real power wheelchair users in single-session and longitudinal studies.

6.2 Design

6.2.1 PW-Joystick EdgeWrite

With the help of A.T. Sciences, Inc. of Pittsburgh, Pennsylvania, PW-Joystick EdgeWrite was implemented in C++ for the Everest & Jennings 1706-5020 power wheelchair joystick, which was removed from its chair (Figure 6.2). Wires attached to the joystick and the left auxiliary switch provide the voltage signals corresponding to the absolute (*x*, *y*) position of the stick and the state of the switch. A National Instruments 6024E DAQCard reads the voltage signals and makes them available to PW-Joystick EdgeWrite.

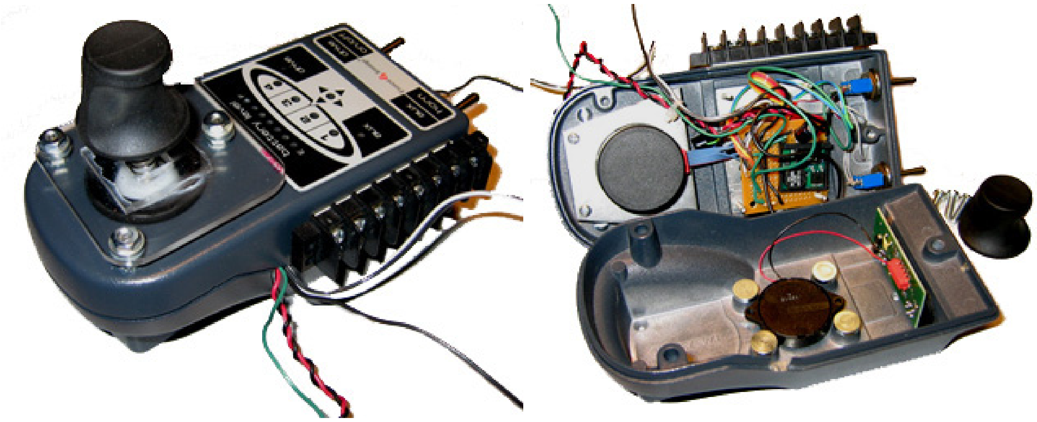


Figure 6.2. The Everest & Jennings 1706-5020 power wheelchair joystick.

The software polls the joystick for its position every 5 ms. When the stick enters one of the four EdgeWrite corners, a stroke begins. When the stick returns to the center of the square for a short duration, the trace is deemed complete and recognition of the corner sequence occurs. This segmentation scheme is the same as in Joystick EdgeWrite.

The joystick's coordinate plane is restricted to the square hole that bounds the stick. One design consideration was the size of this square (Figure 6.3). Through iteration, we found that an edge length of 13.75 mm worked well. It was small enough to reduce the amount of necessary movement, but big enough to reduce the risk of accidental corner-hits. Another consideration was the thickness of the plastic. Our initial studies showed that a thin piece of plastic often got stuck in the joystick spring, so a thicker piece had to be used. The template was mounted using three bolts that were installed from the underside of the joystick chassis (Figure 6.2).

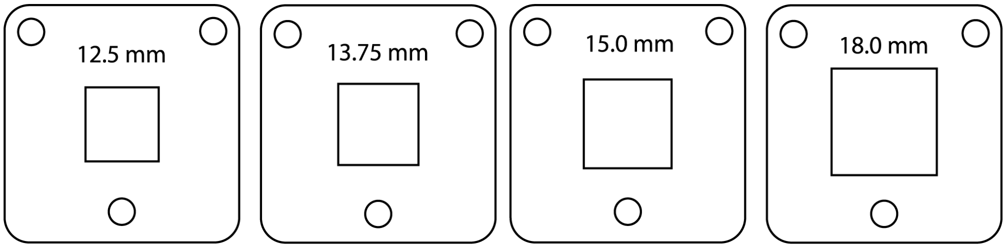


Figure 6.3. The design involved iterating over different plastic template sizes..

The (*x*, *y*) position of the joystick was very noisy, in essence containing a great deal of electronic "tremor" (Figure 6.4a). To filter out this noise, the last *n* points were used to compute a running average, treating the result as a single point (Figure 6.4b). Trial and error yielded *n*=12 as the value that removed sufficient noise while decreasing the inevitable lag introduced by a running average.

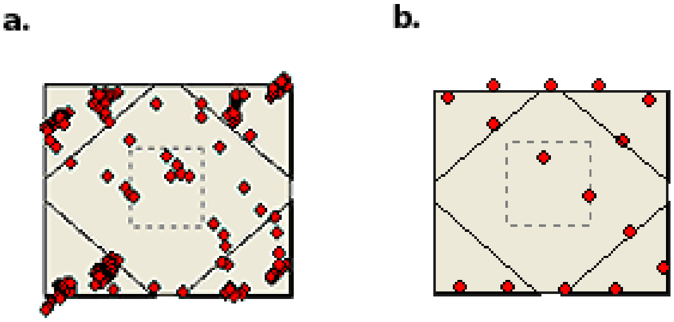


Figure 6.4. Filtering was necessary to clean up the noisy joystick input.

It is important to emphasize that the joystick is not used to drive a mouse cursor. In other words, the joystick is not being used as a relative position device. Instead, the *absolute* position of the stick within the physical plastic square is read, so that when a user feels an edge or corner, their digital position is in agreement.

PW-Joystick EdgeWrite strokes generate key events as if they were typed on the computer keyboard, allowing characters to be sent to any Windows application as long as it has the current input focus.

6.2.2 Touchpad EdgeWrite

Another version of EdgeWrite was implemented in C++ for a Synaptics touchpad (Figure 6.5). Like the stylus and joystick versions of EdgeWrite, the touchpad version uses a plastic template to delineate a square boundary. Although PW-Joystick EdgeWrite was rather sensitive to the size of the square boundary, Touchpad EdgeWrite is not. The physical square shown in Figure 6.5 is 30 mm on a side.

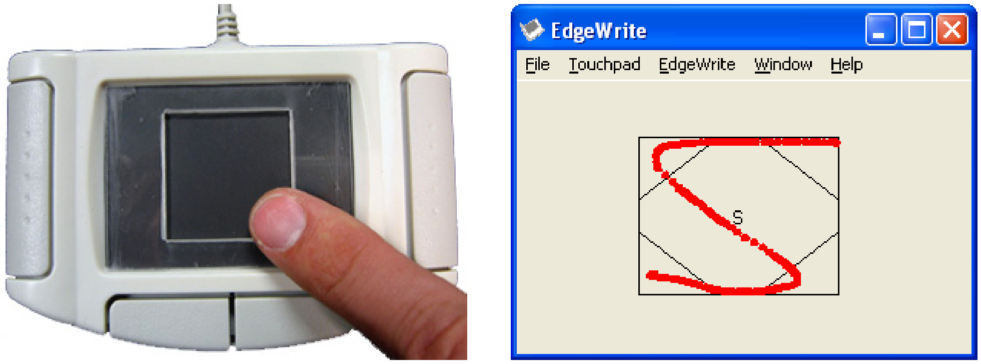


Figure 6.5. Touchpad EdgeWrite uses a plastic template to provide a square boundary on the touchpad's surface. The boundary is mirrored in the application window.

The edges of the touchpad's plastic template aid tremulous finger motion in the same way that physical edges aid stylus motion on a PDA. Users can feel the smooth plastic edges as they move, exerting pressure against them for stability. The touchpad surface is a capacitive sensor that senses human skin, so pressure on the plastic template does not interfere.

Touchpad EdgeWrite is similar to Stylus EdgeWrite (§4) in that stroke segmentation is accomplished when the finger (or pen) is lifted. Before a finger goes down on the touchpad, the corners are rectangular. Once a finger enters a corner, however, the corners deflate into triangles, preventing diagonal strokes from clipping unwanted corners.

Like the stylus and joystick versions, Touchpad EdgeWrite does not drive a mouse cursor but receives input as the *absolute* position of a finger within the EdgeWrite square. Synaptics drivers allow the software to read the absolute position of the finger on the touchpad's surface and prevent touchpad events from moving the mouse cursor.

With help of the Synaptics drivers, Touchpad EdgeWrite receives touchpad events in the background, so the input focus can remain on another window. EdgeWrite strokes generate key events as if they were typed on the computer keyboard, allowing Touchpad EdgeWrite to send text to any Windows application provided it has the input focus.

6.3 Evaluation

The first of two studies were participatory design sessions with 7 power wheelchair users, 6 of whom had Cerebral Palsy and 1 of whom had Multiple Sclerosis. The subjects entered text phrases using Touchpad EdgeWrite, PW-Joystick EdgeWrite, and the WiViK on-screen keyboard using the power wheelchair joystick.[9] It was difficult for subjects to learn the EdgeWrite alphabet in a single session compared to learning the on-screen keyboard, which many subjects had used before. Despite this, Touchpad EdgeWrite was the fastest and most liked method, PW-Joystick WiViK was second, and PW-Joystick EdgeWrite was a close third. These results were promising since subjects had little time to learn EdgeWrite before testing. For this reason, this initial study was followed by a multi-session study as a second investigation.

In the second study, 2 subjects with Cerebral Palsy from the first study were tested over 10 sessions on consecutive workdays. Each session consisted of entering text with Touchpad EdgeWrite, Touchpad WiViK, PW-Joystick EdgeWrite, and PW-Joystick WiViK. The goal was to discover a "crossover point" (MacKenzie and Zhang 1999): the session at which EdgeWrite overtakes WiViK, if any. Our results show that the touchpad methods were faster than the joystick methods, and that EdgeWrite overtook WiViK on both devices after an initial learning period. Results are discussed in depth below.

Both of these investigations confirm that gestural text entry methods often take longer to learn than selection-based methods. But a quality gestural method offers a number of advantages over selection-based methods: it does not require precious screen

[9] Due to time constraints, they did not use the touchpad to control WiViK. However, they *did* use it in the second study.

real-estate; it can be used without looking; it can be customized (or “trained”); and it can require less motion per character, since, at least in theory, gestures can be quite small but keyboards can only be shrunk so much before their keys become too difficult to acquire, especially for users with motor impairments.

Both of the studies that follow took place with the EdgeWrite alphabet prior to the guessability study (§3.3.3). In addition, continuous recognition feedback (§3.4.1), non-recognition retry (§3.4.2), and word-level stroking (§3.5) had yet to be devised for EdgeWrite.

6.3.1 Power Wheelchair Users: Single Session

This section describes the first study, which took place in a single session with 7 real power wheelchair users. In particular, the section conveys the lessons learned and the parameters identified for improvement.

6.3.1.1 The WiViK On-Screen Keyboard

In order to compare EdgeWrite to a currently available means of text entry with a wheelchair joystick, the on-screen keyboard WiViK was used (Shein *et al.* 1991) in conjunction with the wheelchair joystick (§2.3.6). In order to enable subjects to use the WiViK software, proportional mouse control was implemented for the wheelchair joystick. Also, a joystick switch was used to simulate a mouse click. When the switch was pressed, it acted as a mouse-down. When the switch was released, it acted as a mouse-up.

WiViK was used with the default settings, which included no spacing between keys, no word prediction or completion, and click-triggering of keys rather than dwell-triggering. The keyboard consumed the entire width and about 1/3 of the height of a 1024×768 screen. The WiViK keyboard was chosen because of its familiarity as a mouse-driven on-screen keyboard.

Prior research shows that among the possibilities for joystick-driven mouse control, a rate-controlled approach is both fastest and most accurate for on-screen keyboard text entry, as opposed to absolute positioning or a hybrid mode (LoPresti *et al.* 2004). Therefore, a rate-controlled joystick was used, the velocity and acceleration of which were comparable to that used in prior work. When using WiViK with the joystick, the plastic template used by EdgeWrite (Figure 6.3) was removed because it would otherwise restrict the joystick's normal range of motion.

6.3.1.2 Subjects

The three text entry methods were tested and iteratively improved with the help of 7 power wheelchair users.[10] Six of the 7 had Cerebral Palsy and were clients of the United Cerebral Palsy Center of Pittsburgh, Pennsylvania. One subject had Multiple Sclerosis. The average age of the subjects was 25.9 years, with a low of 21 and a high of 67. Subjects had been in wheelchairs for an average of 14 years, with a low of 3 and a high of 30. Two of the 7 subjects were male. Four were right-handed. All but one of them used a conventional QWERTY keyboard for text input, but nearly all of them said that they could only do so for short periods of time before becoming fatigued. None of the subjects had ever used EdgeWrite, but 6 of 7 had used WiViK. Thus, subjects were very familiar with the competitor technique, but not with EdgeWrite.

6.3.1.3 Procedure

Subjects practiced each technique before entering a single test phrase (~30 letters). Practice consisted of entering each letter 4 times in a row with a given technique (i.e. "aaaa", "bbbb", …, "yyyy", "zzzz"). This took 25–35 minutes with the EdgeWrite techniques, and about 10–20 minutes with WiViK. The practice period was followed by a test period in which, due to time constraints and rapid fatigue, subjects entered one complete sentence of about 40 characters in length with each method. All 7 subjects used PW-Joystick WiViK and PW-Joystick EdgeWrite, but only 4 subjects used Touchpad EdgeWrite because of time constraints. A comparison of the joystick data from these 4 subjects to the other 3 subjects shows similar results, suggesting that touchpad results for all 7 subjects would not be substantially different.

An EdgeWrite character chart was visible during the test (Figure 6.6). With the slow pace of practice and subjects' limited endurance, it did not seem appropriate to burden subjects with memorizing the EdgeWrite characters. Instead, subjects were taught how to read the chart and their behavior was observed. Reading the chart greatly slowed them down compared to using WiViK, which requires no chart. The *inter-character time*—the time from the end of one character to the start of the next—gives some idea of the delay caused by reading the chart. The average inter-character time was 6.23 seconds. As

[10] Eight subjects began the study but one was unable to perform any of the techniques.

subjects become familiar with letters, this value goes down. The second study confirms that after 10 sessions, the inter-character time was 3.74 seconds. This certainly affects the overall speed of EdgeWrite.

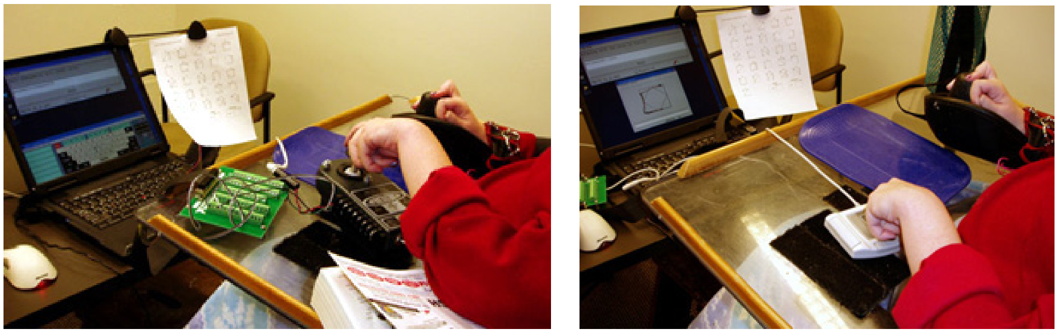


Figure 6.6. A subject using PW-Joystick EdgeWrite (left) and Touchpad EdgeWrite (right). A paper character chart was shown during the study.

Responses were solicited from subjects during the practice period and more formally using a post-test questionnaire. In addition, subjects were encouraged to share their impressions and ideas while practicing.

6.3.1.4 Results

As often is the case in accessibility research, the results lack sufficient power for statistical significance because of the small number of trials and the high variability among subjects. However, one can still compare the means of the three techniques and associate performance with subjects' comments, which were illustrative. Standard deviations are reported in parentheses.

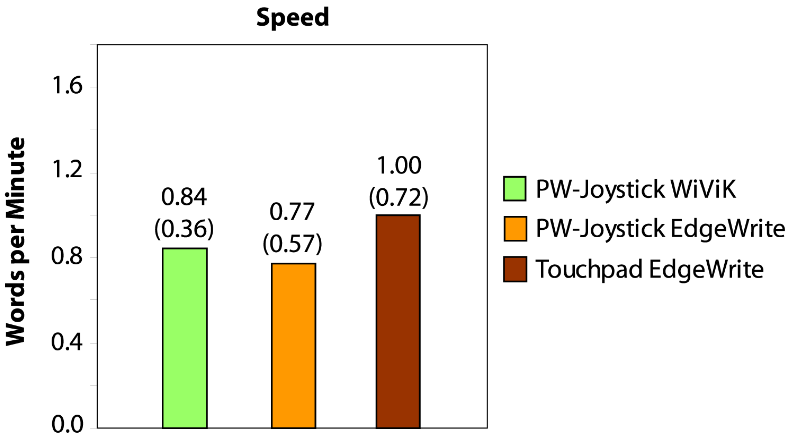


Figure 6.7. Text entry speeds in words per minute for the three methods.

Speeds in WPM are shown in Figure 6.7 and error rates are shown in Figure 6.8. Touchpad EdgeWrite was the fastest at 1.00 WPM (0.72), PW-Joystick WiViK was second at 0.84 WPM (0.36), and PW-Joystick EdgeWrite was a close third at 0.77 WPM (0.57). Although these speeds are slow, they are not uncommon for subjects with motor impairments. In fact, many of our subjects used WiViK in everyday use, often with a trackball, which may result in similar speeds.

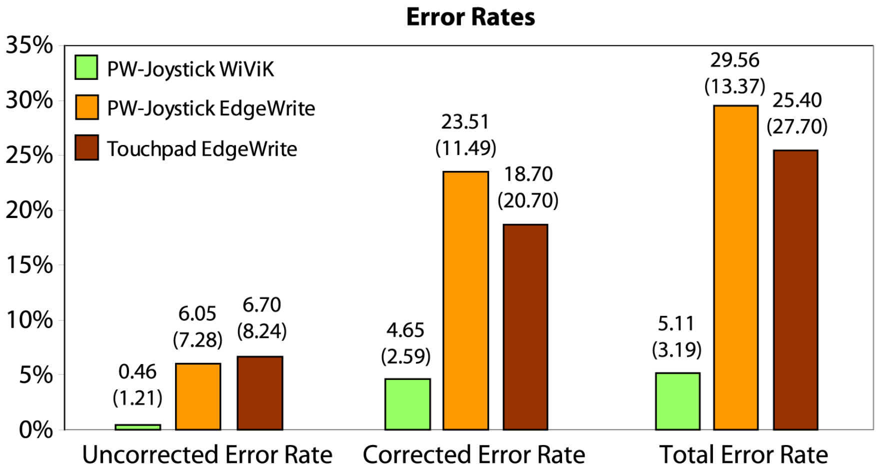


Figure 6.8. Uncorrected, corrected, and total error rates for the three methods.

Clearly, subjects made more errors with the EdgeWrite methods than with PW-Joystick WiViK. This is not unexpected of gestural input techniques compared to selection-based ones, since when learning new gestures, users often perform them incorrectly. On the other hand, to make an error with WiViK, a subject must place the mouse cursor over the wrong key and *still* choose to press and release the switch—a lengthy process easily avoided most of the time.

The results show a speed/accuracy tradeoff with Touchpad EdgeWrite and PW-Joystick WiViK. That said, before this study, the sheer feasibility of Touchpad EdgeWrite and PW-Joystick EdgeWrite for people with motor impairments was unknown. Although these results are mediocre for EdgeWrite, we must remember that nearly all subjects were familiar with WiViK and had used it extensively. The fact that subjects could do EdgeWrite at all was enough encouragement to conduct a longitudinal study in which subjects would have more time to become proficient with the techniques.

Questionnaire results showed that, of the 3 methods, subjects felt that Touchpad EdgeWrite was the easiest to use, easiest to learn, fastest, most accurate, most enjoyable, most comfortable, and most liked (Figure 6.9). Subjects rated PW-Joystick WiViK second in all of these categories, and PW-Joystick EdgeWrite third. A repeated measures analysis of variance[11] of subjects' average Likert scores yields a significant difference ($F_{2,8.47}$=10.06, p<.01). Contrasts show that Touchpad EdgeWrite was preferred to PW-Joystick EdgeWrite ($F_{1,8.45}$=19.62, p<.01) and to PW-Joystick WiViK ($F_{1,8.52}$=5.93, p<.05). PW-Joystick WiViK was preferred to PW-Joystick EdgeWrite ($F_{1,8.26}$=7.40, p<.05).

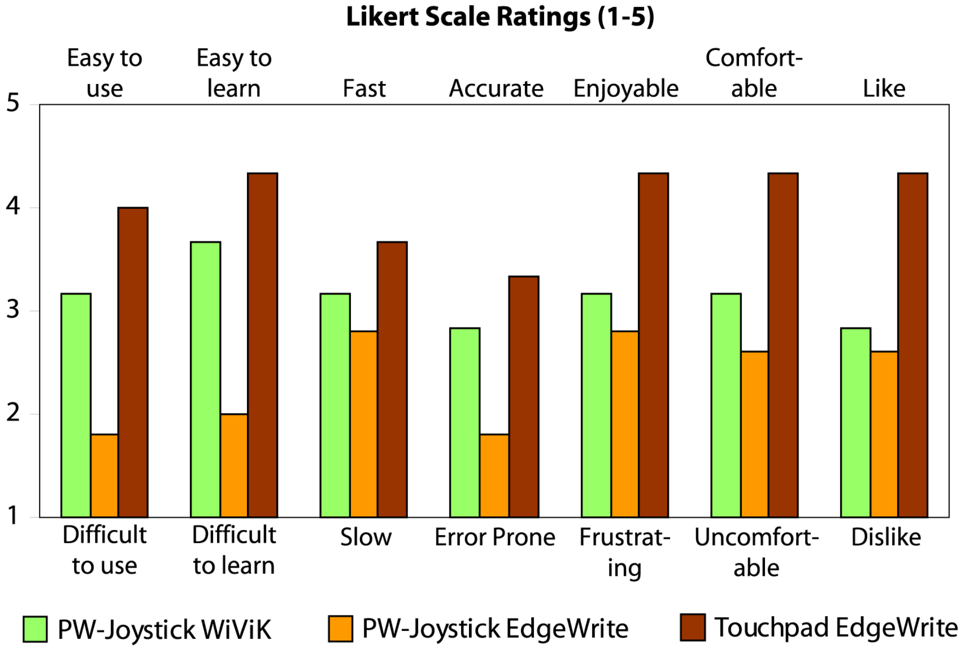


Figure 6.9. Subjective ratings for the three methods. Ratings are on a Likert scale (1–5). Higher values are better.

6.3.1.5 *Lessons from Subjects*

Subject #1 was a 67 year-old retired school teacher with Multiple Sclerosis. He was notable for two reasons: he was the only person without Cerebral Palsy, and he was only one of two subjects who was faster with PW-Joystick EdgeWrite than PW-Joystick WiViK (1.91 *vs.* 1.22 WPM). The other was Subject #8, who was a 22 year-old female with good fine motor control. She was only slightly better with PW-Joystick EdgeWrite than PW-Joystick WiViK (0.52 *vs.* 0.50 WPM). Subject #1 demonstrated that the plastic template should be thicker to prevent the exposed spring on the joystick post from catching the template's edge. After using PW-Joystick WiViK for a few minutes he said,

[11] A mixed model analysis of variance can handle the missing data for Touchpad EdgeWrite. However, this results in fractional denominator degrees of freedom (Littell *et al.* 1996).

"It takes the patience of Job to do this." Upon switching from WiViK to PW-Joystick EdgeWrite, he said, "I'm much faster with this; don't you think I'm much faster?" indicating his first impression.

Subject #2 was a 21 year-old student. She initially had trouble with the diagonal strokes with PW-Joystick EdgeWrite because she would move too slowly through the center, and EdgeWrite would try to segment the stroke she had already done. She motivated a change to the center dwell time required for segmentation. If a polled joystick point falls outside the center area before the dwell time elapses, the dwell time counter resets. The time that worked well for Subject #2 was 500 ms. This subject also thought it would be easier to use PW-Joystick WiViK with the EdgeWrite template still on the joystick because it would help prevent target over-shooting. This suggested that joystick mouse control and PW-Joystick EdgeWrite could co-exist on the same device without having to remove the EdgeWrite template.

A long dwell time was not sufficient for Subject #4, a 40 year-old volunteer. She moved inconsistently with PW-Joystick EdgeWrite, sometimes making letters very quickly, other times pausing for many seconds to think. For her, the ability to trigger recognition with the joystick switch was added, which removed return-to-center segmentation altogether. She enjoyed Touchpad EdgeWrite because she said it was the easiest method with which to fix mistakes. "Once you understand what you are doing, it goes completely well," said Subject #4 of Touchpad EdgeWrite. Subject #7 echoed this when she said, "If you get used to it, you'd be really fast I suppose."

While the females tended to interact too gingerly with the joystick, the males, Subjects #1 and #3, were too forceful at first. Discovering the right speed and pressure to exert against the joystick template was an obvious part of learning PW-Joystick EdgeWrite.

A common problem was that participants did not always start in the corner of the plastic template before making their gestures with the joystick. This was less of a problem with the touchpad. The reason may be that the joystick must be pushed from the center to reach the starting corner, whereas a finger can begin in the corner of the touchpad.

Subject #4 provided an important insight into the design of the touchpad template. Originally, the edge of the touchpad template was smooth and slightly beveled. But this caused subjects' fingers to slip up onto the template's surface, actuating a "finger up" and prematurely triggering recognition. This insight led to the fabrication of a thicker touchpad template, the edges of which were left vertical and unbeveled. A settable lift tolerance was also added, as it eventually was in Stylus EdgeWrite (§4.2.4).

Subject #6 highlighted the importance of end-user customizability. While using Touchpad EdgeWrite, this subject's finger did not always press flush against an edge of the physical square. After the software square had been defined for her along the plastic edges, it became clear that her fingers moved *inside* this square, and that the actual square in which she moved was smaller than the one defined. When *she* defined the EdgeWrite square for herself, her accuracy improved a great deal.

Finally, the diagonal strokes were difficult for many users of PW-Joystick EdgeWrite. This is not surprising, because it is along the diagonals that one does not have an edge to press against. The letter "k" (Figure 6.10a) was particularly problematic because of its two diagonals in a row. For the second study, a new form of "k" was designed (Figure 6.10b). The new "k" is still reminiscent of a Roman "k" but without a diagonal. The new "k" proved much easier to perform and has become a part of the EdgeWrite alphabet (Appendix A).

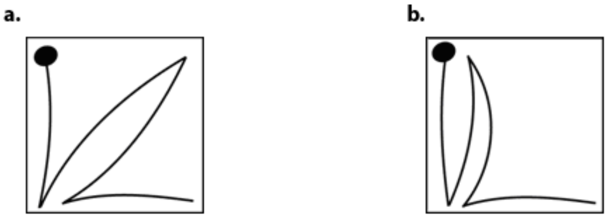


Figure 6.10. The original "k" with diagonals and a new "k" without diagonals.

6.3.2 Power Wheelchair Users: Multiple Sessions

The findings from the first study, which largely represent "walk up and use"-ability, inspired a second investigation over multiple sessions. Such a study can identify a "crossover point" where EdgeWrite, although initially harder to learn, overtakes WiViK in speed or accuracy. Subjects #2 and #4 from the first study agreed to partake in a 10-session study over consecutive workdays. There was about 3 months separating the end of the first study and the beginning of the second one.

6.3.2.1 Subjects

Subject #2 is female and 22 years old. She uses a computer more than a few hours a day, largely for email, surfing the Web, and word processing. She reports being able to use a standard physical QWERTY keyboard for up to 1 hour, after which point she switches to an alternative method, usually a WiViK on-screen keyboard accessed with a standard mouse, because of fatigue. She is able to write her name with a pen, but it takes her many seconds, and it is legible only about 4 out of 5 times. The joystick on her power wheelchair has a short stick with a plastic ball at the top.

Subject #4 is female and 41 years old. She uses a computer only about once a week for email, surfing the Web, or word processing. She, too, reports being able to use a standard physical QWERTY keyboard for up to 1 hour, at which point she either stops using the computer or uses the WiViK on-screen keyboard with a standard mouse. She can write her name legibly with a pen but it takes many seconds if not minutes, and is legible only about 4 out of 5 times. The joystick on her power wheelchair is about twice as long as and a great deal skinnier than the Everest & Jennings 1706-5020 model used in these studies (Figure 6.2). Her joystick also had a much weaker spring. Subject #4 was a great deal weaker than Subject #2, which affected her ability to move the joystick snugly into the corners while using PW-Joystick EdgeWrite.

6.3.2.2 Design Improvements

Before conducting the second study, some design changes were made to the techniques based on our observations from the first study. For example, tolerance was added for a brief lifting of the finger from the touchpad's surface. (Previously, when a finger was lifted the stroke was immediately ended and recognition commenced.) The new tolerance, which took the form of a customizable lift-delay, allowed subjects' fingers to lift briefly from the surface and return. Both subjects worked well with a 275 ms lift-tolerance.

The triangular corner regions in Touchpad EdgeWrite were also reduced slightly from 47.5% of the square's width and height in each dimension to 42.5%. This was because subjects would sometimes hit an unwanted corner while making a diagonal, particularly from the bottom-left corner to the top-right for right-handed users. (Subject #2 and Subject #4 were both right-handed.)

The area considered the "center" for PW-Joystick EdgeWrite was reduced by about 11% to make accidental recognitions less common, since subjects would often move too slowly through the center region while making a diagonal. In the first study, this caused the software to think the joystick had been returned to center for segmentation between letters. Subject #2 required a 500 ms center dwell segmentation threshold, while Subject #4 required 1000 ms. Lower values resulted in some unwanted attempts at recognition when moving through the center region.

Finally, as noted above, a new form of "k" that contains no diagonals was added to the EdgeWrite alphabet. A new form of "e" was also added, one that is tolerant to the omission of the bottom-left corner, since this corner was occasionally missed for "e".

As stated above, the joystick used with WiViK was rate-controlled, so the farther it was moved from its center, the faster the mouse cursor moved. The acceleration transfer function was linear from the center of the joystick to its extremes. For PW-Joystick WiViK, this acceleration was reduced from a maximum of 1.2 pixels/ms to 0.8 pixels/ms. This change was made because of some occasional target overshooting that occurred when subjects tried to acquire keys on the WiViK keyboard. With the reduced acceleration, target overshoots in the second study were rare, and yet the mouse cursor moved at a comfortable speed.

6.3.2.3 Procedure

The experiment was a 2×2×10 within-subjects factorial design with factors for *Method* (EdgeWrite, WiViK), *Device* (PW-Joystick, Touchpad), and *Session* (1–10). With only 2 subjects, the experiment was aimed less at achieving statistical significance and more at observing how long it takes users to learn EdgeWrite, and whether EdgeWrite would outperform WiViK given more practice.

Each subject performed all 4 techniques (*Method*×*Device*) during each session. The order of techniques was assigned randomly by the software for each session. Subjects first practiced each technique by entering a short 2-word phrase (~10 letters). During the test, subjects transcribed 2 phrases of about 6 words (~30 letters) each. This took from 10–30 minutes per technique, depending on the session, technique, technique order, and other factors. Thus, a session consisted of about 70 characters for each of the four techniques.

The test phrases were drawn randomly from the published test corpus of 500 phrases (MacKenzie and Soukoreff 2003). These phrases only contain letters, but as MacKenzie and Soukoreff argue, unless the entry of numbers or punctuation involves a qualitatively different mechanism (e.g. two hands instead of one), letters alone can be considered representative. In the case of EdgeWrite and WiViK, numbers and punctuation are accessed in the same general manner as letters.

Unfortunately, Subject #2 only finished 6 sessions with PW-Joystick WiViK due to technical problems during the 7th and 8th sessions. She finished 8 sessions with the other 3 techniques. These imbalances are taken into account in the statistical analyses.

6.3.2.4 Analysis

The data were analyzed using a mixed model analysis of variance with a random effect for *Subject*.[12] Mixed models with random effects give wider confidence intervals (i.e. larger standard errors) than fixed models and therefore set a higher bar for determining statistically significant differences. They also result in greater denominator degrees of freedom. The aforementioned imbalance in the number of sessions was accommodated in part by using least squares estimates for means (LS Means).

6.3.2.5 Speed

Overall results show a main effect of *Device* on speed ($F_{1,131}$=142.05, p<.001). The touchpad was faster than the joystick (1.28 *vs.* 0.82 WPM). There was no main effect of *Method* on speed ($F_{1,131}$=2.95, ns). This is because the EdgeWrite methods were slower in the early sessions but faster at the end. There was also no significant *Method*×*Device* interaction ($F_{1,131}$=0.01, ns), since touchpads were similarly faster than joysticks for both EdgeWrite and WiViK.

[12] Modeling *Subject* as a random effect is necessary for this analysis because the subjects represent a sample from a larger population. The model also takes into account the fact that measurements within subjects are not independent. For more information on mixed model analyses, see Chapter 3 in (Littell *et al.* 1996).

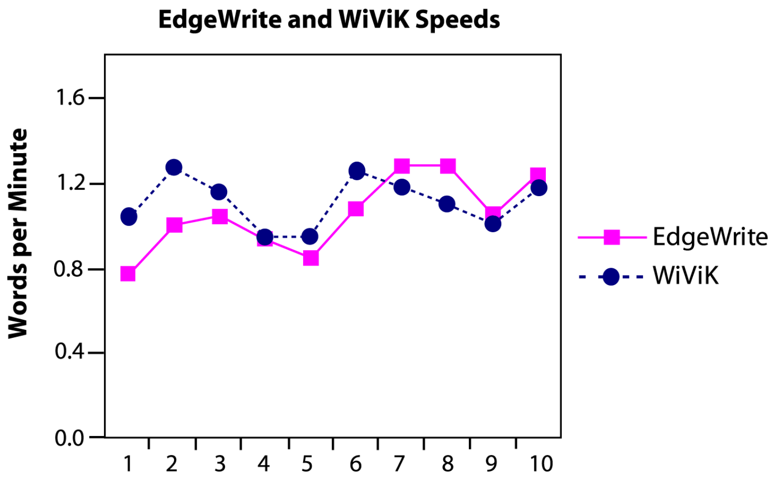


Figure 6.11. EdgeWrite and WiViK speeds over sessions.

Figure 6.11 gives a sense of the rate at which both methods were learned. As noted above, sessions 7 and 8 lack data for PW-Joystick WiViK for Subject #2. This actually *improves* WiViK's speed for those 2 sessions because Touchpad WiViK was significantly faster than PW-Joystick WiViK ($F_{1,131}$=70.07, p<.001).

As one might expect, there was a significant main effect of *Session* on speed ($F_{1,131}$=9.84, p<.003). Subject #2 improved her overall average from 0.91 WPM in session 1 to over 1.17 WPM by session 6. (Sessions 7 and 8 were even faster at 1.41 and 1.22 WPM, respectively, but these lacked data for PW-Joystick WiViK.) Subject #4 improved her overall average from 0.87 WPM in session 1 to 1.11 WPM in session 10. There was no significant *Session*×*Device* interaction ($F_{1,131}$=0.03, ns), since the touchpad and joystick were learned at similar rates. However, there was a significant *Session*×*Method* interaction, indicating methods improved over sessions at different rates ($F_{1,131}$=10.35, p<.002). Contrast tests show that this improvement was due mostly to EdgeWrite and not to WiViK, as there was significant speedup from the first 5 sessions to the second 5 sessions for EdgeWrite ($F_{1,99}$=25.61, p<.001) but not for WiViK ($F_{1,99}$=1.73, ns). Figure 6.12 breaks down speed for each of the four techniques.

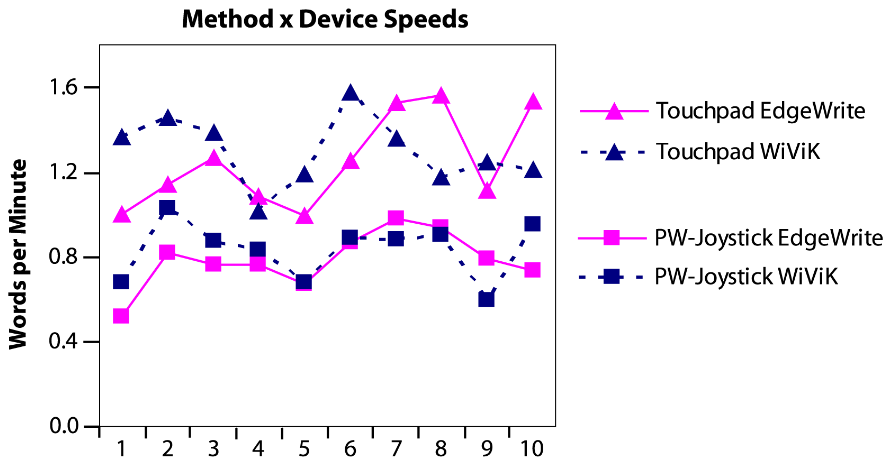


Figure 6.12. The speeds of the four techniques over sessions.

6.3.2.6 Total Error Rate

Results show a main effect of *Device* on total error rate ($F_{1,131}$=23.39, p<.001). The touchpad was more accurate than the joystick (5.56% *vs.* 10.74%). There was also a significant main effect of *Method* on total error rate ($F_{1,131}$=148.99, p<.001). As in the first study, WiViK was more accurate than EdgeWrite (1.62% *vs.* 14.69%).

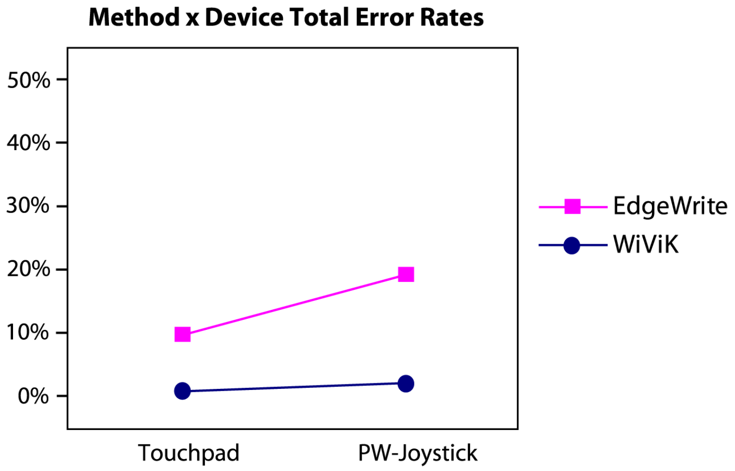


Figure 6.13. The touchpad and joystick affected EdgeWrite and WiViK differently.

A significant *Method*×*Device* interaction (Figure 6.13) shows that the two devices affected each method's accuracy differently ($F_{1,131}$=15.58, p<.001). Contrasts show that PW-Joystick EdgeWrite was significantly less accurate than Touchpad EdgeWrite (19.39% *vs.* 9.98%, $F_{1,131}$=39.94, p<.001). However, PW-Joystick WiViK was *not* significantly less accurate than Touchpad WiViK (2.09% *vs.* 1.14%, $F_{1,131}$=0.38, ns).

As expected, there was a main effect of *Session* on total error rate ($F_{1,131}$=15.75, p<.001). There was also a significant *Session*×*Method* interaction ($F_{1,131}$=10.14, p<.002), indicating method accuracy changed for each method differently over time (Figure 6.14). Contrasts show that this was due to EdgeWrite improving over sessions while WiViK remained about the same. A contrast shows that EdgeWrite improved from the first 5 trials to the second 5 trials ($F_{1,99}$=26.92, p<.001), while WiViK's accuracy did not ($F_{1,99}$=0.39, ns). With more sessions and further refinements, particularly to the joystick's physical parameters (e.g. spring strength, stick length), EdgeWrite error rates would probably drop further and become competitive with WiViK.

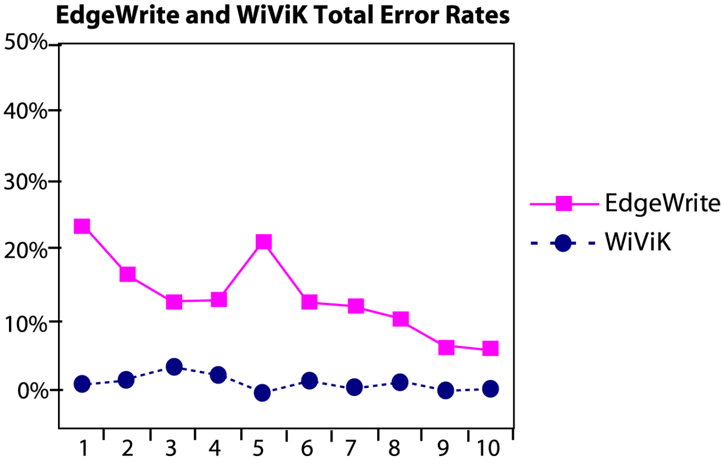


Figure 6.14. EdgeWrite and WiViK total error rates over sessions.

There was a significant *Session*×*Device* interaction ($F_{1,131}$=5.55, p<.02), indicating different rates of accuracy improvements with each device. From the first 5 trials to the second 5 trials, subjects became more accurate with the joystick ($F_{1,99}$=18.42, p<.001) but not with the touchpad ($F_{1,99}$=1.76, ns). Figure 6.15 breaks down total error rates by each of the four techniques.

That the overall error rates are higher for EdgeWrite than WiViK is not unexpected, since learning and performing a gestural entry technique will usually be more error prone than selecting from an on-screen keyboard. But EdgeWrite results show dramatic improvements in total error rates for each subject: from 21.2% in session 1 to 11.7% in session 8 for Subject #2; and from 26.8% in session 1 to just 6.0% in session 10 for Subject #4.

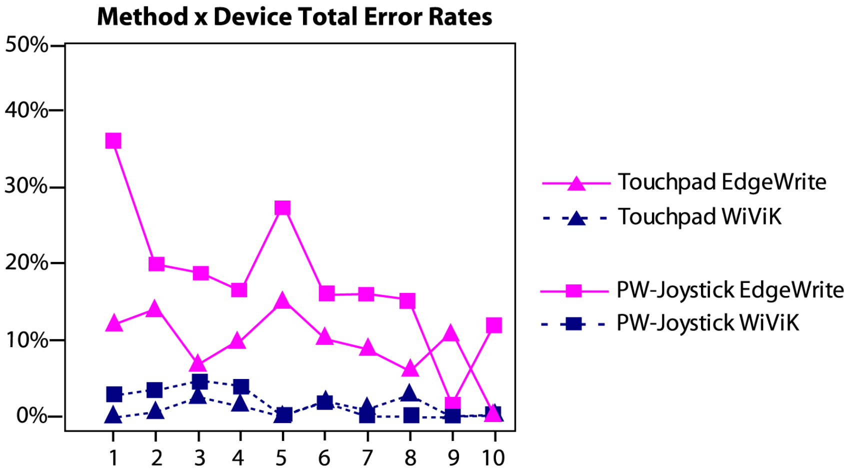


Figure 6.15. The total error rates of the four techniques over sessions.

6.3.2.7 Learning Curves

It is customary in input studies to fit regression curves to speed data based on the power law of learning (Card *et al.* 1983, MacKenzie and Zhang 1999). Such curves are of the form $y = bx^c$, where y is speed and x is session, and b and c are regression coefficients. These performance models allow researchers to predict how a subject might perform in future sessions. Fitting these curves is quite speculative for the current data, however, since there are only two subjects. Nonetheless, the curves give a sense of how their performance may proceed beyond session 10.

The regression curves for the 4 *Method*×*Device* combinations are shown in Figure 6.16. The data is highly varied, so obtaining high correlations is not possible. The learning curves show clear upward trends for the two EdgeWrite methods. The flat or downward slopes for WiViK are due to subjects' prior familiarity with WiViK from extended use. In addition, on-screen keyboards require little learning and offer little room for improvement. Thus, the WiViK curves are governed more by fatigue than by learning.

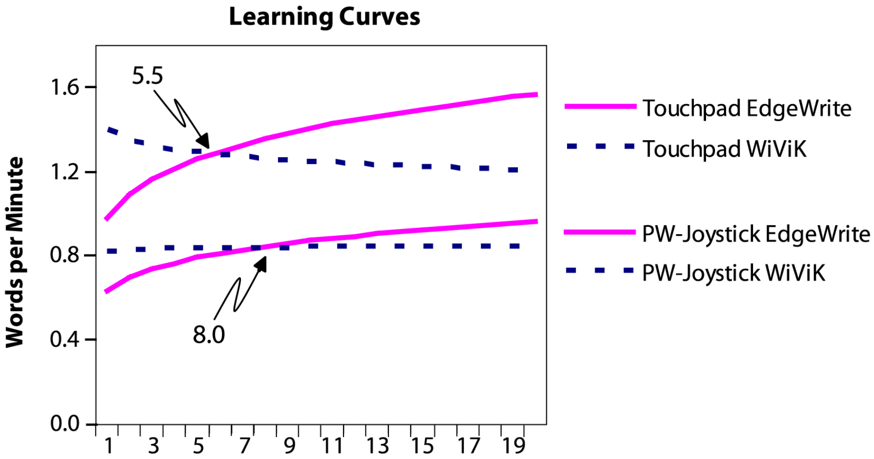


Figure 6.16. Learning curves show crossover points for both EdgeWrite techniques.

The power law equations and R^2 values are as follows:

• $y = 0.978x^{0.1574}$	$R^2 = 0.38$	Touchpad EdgeWrite
• $y = 1.397x^{-0.0492}$	$R^2 = 0.09$	Touchpad WiViK
• $y = 0.632x^{0.1406}$	$R^2 = 0.36$	PW-Joystick EdgeWrite
• $y = 0.819x^{0.0110}$	$R^2 = 0.003$	PW-Joystick WiViK

Crossover points for the touchpad and joystick techniques occur at about sessions 5.5 and 8.0, respectively, with EdgeWrite overtaking WiViK in both cases. While these curves are speculative, they do give a sense of the overall trends: EdgeWrite did indeed speed up and reduce errors over sessions. The higher R^2 values for both EdgeWrite techniques suggest that more learning took place for these methods than for the WiViK methods, and would likely continue to do so with further practice.

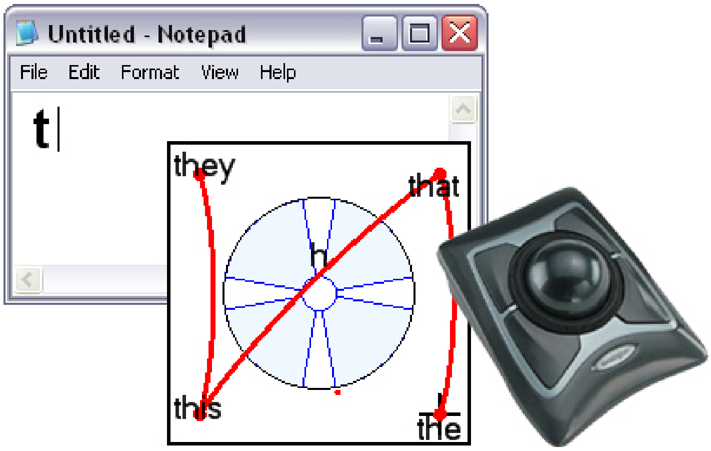
6.3.2.8 Discussion

Overall, the results for speed and accuracy confirm both the challenge of learning a gestural text input method and the potential benefits. The initially poor accuracy of EdgeWrite, particularly the joystick version, is not surprising, and could be mitigated with further design. For example, both subjects' personal power wheelchair joysticks were longer and had much weaker springs than the one used in the study. Optimizing design parameters such as stick length and spring strength would be one way to improve performance. The general advantage of the touchpad over the joystick points to touchpads for future inclusion in computer access solutions.

A post-test questionnaire showed similar results for the two subjects as from the first study (Figure 6.9). Both subjects preferred Touchpad EdgeWrite overall, followed by Touchpad WiViK, PW-Joystick EdgeWrite, and PW-Joystick WiViK. For both devices, the WiViK methods were considered easier to learn but the EdgeWrite methods were preferred for their perceived speeds.

6.3.3 Expert Able-bodied Use

To appreciate the differences among these text entry methods, I entered 10 phrases with PW-Joystick EdgeWrite and Touchpad EdgeWrite (Wobbrock and Myers 2005a). My speeds were 12.9 and 19.1 WPM, respectively. My total error rates were 8.4% and 4.7%, respectively. However, *all* errors made during entry were corrected, which reduced speeds, so these data represent “perfect transcription.” While this only reflects one expert, it permits ballpark comparisons.



Chapter 7

Trackball EdgeWrite*

7.1 Motivation

Trackballs are the preferred pointing devices for numerous computer users, particularly for people with some form of motor impairment (Fuhrer and Fridie 2001, Wu *et al.* 2005). For people with low strength, poor coordination, wrist pain, or limited ranges of motion, rolling a trackball can be easier than shuttling a mouse across the surface of a desk. Trackballs' accessible properties include: they do not require the wrist or forearm to be elevated; they do not occupy much physical space, making them suitable for placement in a person's lap or on a wheelchair tray; they are easy to manipulate, as rolling a trackball requires relatively little strength; if clutching is necessary, one must only lift one's finger or hand, not the device itself (as with a mouse); and trackballs are simple, cheap, robust, and available, factors that when absent are known to be barriers to adoption (Fichten *et al.* 2000, Dawe 2004, Dawe 2006).

Not surprisingly, many people who prefer trackballs due to motor impairments also cannot use a conventional physical keyboard. For these people, some form of text entry besides typing is required. Using the trackball itself for text entry reduces physical

* Parts of this chapter are adapted from (Wobbrock and Myers 2006d, Wobbrock and Myers 2006a).

movement among devices and the need for multiple devices to be within reach (Guerette and Sumi 1994, Spaeth *et al.* 1998). Thus, a common solution is to use a trackball with an on-screen keyboard, and to click or dwell on the virtual keys. But, as described in §2.3.6, this method of text entry has many problems (e.g. two foci-of-attention, exacerbated mousing, consumption of screen real estate). In particular, on-screen keyboards can only be used by sight in a "hunt-and-peck" fashion, making them visually tedious (Anson *et al.* 2005). What is needed is a trackball text entry method that leverages the strengths of trackballs but does not require an on-screen keyboard.

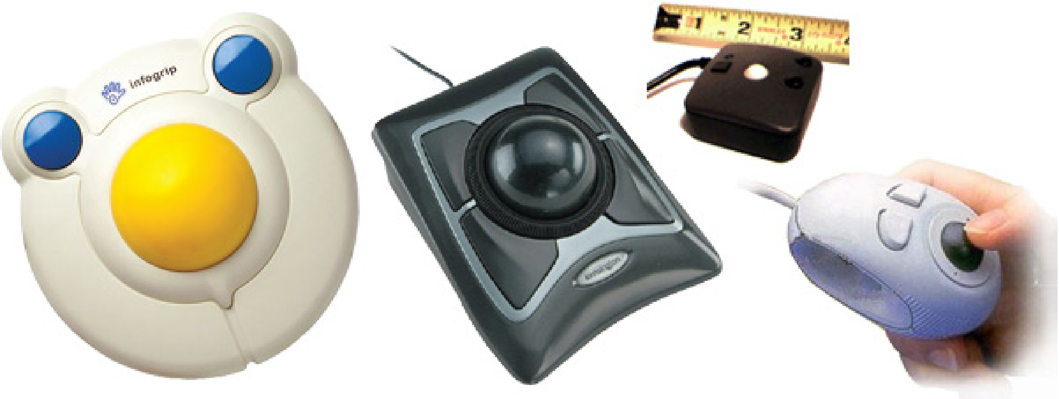


Figure 7.1. Trackballs come in many different sizes making them appropriate as computer access and mobile technologies. From left to right: Infogrip *BIGTrack* (http://www.infogrip.com), Kensington *Expert Mouse* (http://www.kensington.com), Appoint *Thumbelina*, and Infogrip's *Mini Trackball*. Relative image sizes are maintained.

Trackballs may also be preferred for reasons other than physical impairment. Trackballs need little space in which to operate, unlike mice, which have large "desktop footprints" (Card *et al.* 1990). Trackballs can be embedded in consoles or keyboards, making them suitable for public terminals since they cannot be easily stolen. Trackballs also offer rapid, fluid control and have been used in arcade games like *Centipede*. Finally, trackballs can be made quite small (Figure 7.1), making them suitable as a thumb-controlled device for mobile computers.

This chapter presents Trackball EdgeWrite as an alternative to on-screen keyboard text entry with a trackball. The result is a faster and less tedious method of trackball text entry for people who already use trackballs but cannot touch-type on a physical keyboard. This includes people with repetitive stress injuries, spinal cord injuries, arthritis, and some neuromuscular disorders.

Trackball EdgeWrite is first evaluated in a single session with 3 able-bodied users. Then, results are presented from an extended field deployment with a veteran trackball user with a spinal cord injury. Lastly, results are shown for Trackball EdgeWrite's Fisch adaptation for word-level stroking (§3.5). After 15 years of using a trackball with an on-screen keyboard, the veteran trackball user has switched to Trackball EdgeWrite for everyday use.

7.2 Design

Thus far, this dissertation has described versions for styli, displacement joysticks, and touchpads. All of these versions have relied on the *absolute physical position* of an input device within a square. Trackball EdgeWrite is the first version to use *relative* positioning, since trackballs themselves express no absolute position or rotation, only change in rotation (Card *et al.* 1990). In relative positioning, the mouse cursor itself becomes the locus of input, and issues arise that are different than in previous versions. In cursor-based relative positioning, physical edges are no longer possible. An initial design was therefore to reproduce physical edges as impenetrable *virtual edges*. However, this proved insufficient for many reasons, described below (§7.2.1). Ultimately, the right feel was obtained with *goal crossing* (§7.2.2) (Accot and Zhai 1997, Accot and Zhai 2002).

7.2.1 Initial Design: Impenetrable Virtual Edges

The initial design for Trackball EdgeWrite essentially replicated the physical design for Stylus, Joystick, and Touchpad EdgeWrite in a virtual space. A window with impenetrable virtual edges was used to bind the movement of the mouse cursor, which was freely driven within the square from corner to corner (Figure 7.2a).

In an effort to make this free motion more accurate, a simple technique was used to move the cursor directly toward intended corners. In Figure 7.2b, a virtual cursor and the underlying real cursor are both positioned at A. The user moves the real cursor a distance d to A'. A projection along this path intersects the square's edge at E, where the nearest corner is C. The distance d is then applied to the virtual cursor at A to move it to A'' on a straight-line to C. The position of A'' is computed with Equation 7.1. Finally, the real cursor at A' is realigned with the virtual cursor now at A''. This is done for all movements within the virtual square, allowing the user to change direction instantly but always be on a straight line to their intended corner.

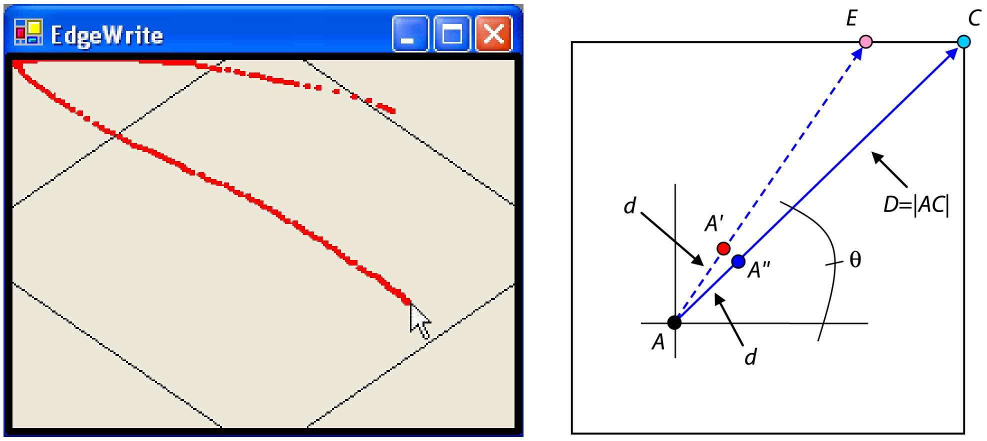


Figure 7.2. (a) The initial design for Trackball EdgeWrite used impenetrable virtual edges and a freely-driven mouse cursor. (b) An attempt to make cursor movement more accurate by inferring the intended corner and always moving directly toward it.

$$A'' = \left(A_x + \frac{d}{D}\left(C_x - A_x\right),\ A_y + \frac{d}{D}\left(C_y - A_y\right) \right) \tag{7.1}$$

Admittedly, the use of impenetrable virtual edges was an obvious starting point for the design of Trackball EdgeWrite. But although the straight-line movement scheme helped, in the end this design for Trackball EdgeWrite did not “feel right.” This was because there was only a loose correspondence between a user’s motion on the trackball and the location of the mouse cursor within the virtual EdgeWrite square. For example, if the user rotated the trackball forward-and-left, there was no guarantee that the mouse cursor would be in the top-left corner—it depended on where the mouse cursor started and on how much the user moved the trackball. Thus, another approach was necessary, one that more tightly coupled users’ trackball movements with their location in the EdgeWrite corners. Goal crossing would become this approach.

7.2.2 Crossing to Stroke

Recall the discussion of goal crossing from §2.2.2. Accot and Zhai compared different types of pointing and crossing using a stylus (Accot and Zhai 2002). They found that the fastest arrangement for short-range reciprocal trials was “continuous orthogonal goal crossing,” where the pointing device, in their case a stylus, was continually held on the surface and the goals were perpendicular to the movement trajectory. They speculated that goals could rotate to always remain orthogonal to the cursor and thus offer the maximum target width (Figure 7.3a). An extreme form of this idea is a cursor placed inside a circle, where the circumference is the goal (Figure 7.3b). This insight is one inspiration for the design of Trackball EdgeWrite.

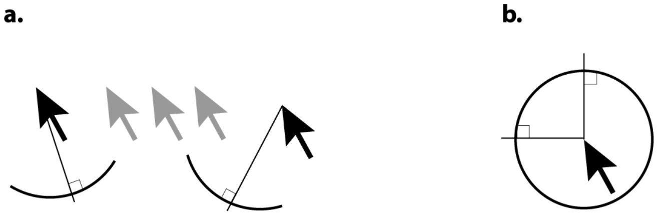


Figure 7.3. Accot and Zhai speculate that crossing goals could rotate to always remain orthogonal to the cursor, thereby offering maximum target width. An extreme form of this idea is a cursor in the center of a circle, where the circumference itself is the goal.

Using Trackball EdgeWrite, one writes by making short "pulses" towards intended corners. When these pulses cause the mouse cursor to cross the circumference of a circle (Figure 7.3b), the resultant angle indicates the next corner. Thus, the cursor does not actually *travel* among corners to acquire them as targets, but crosses a radius and snaps to the next corner instantly. In essence, it is the vector along which the cursor moves that determines the next corner. Figure 7.4 depicts this process for writing the letter "z".

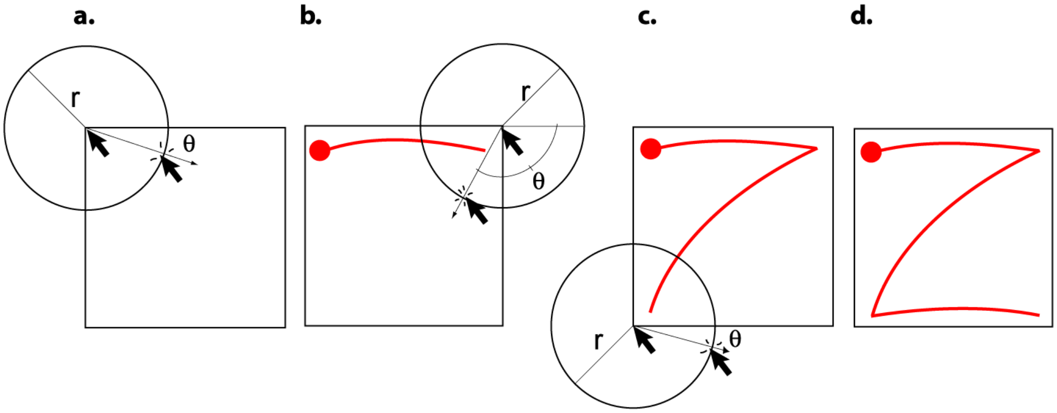


Figure 7.4. The repeated crossings involved in writing the letter "z".

In Figure 7.4a, the trackball cursor moves a distance *r* at an angle *θ* from the top-left corner of the EdgeWrite square. The angle determines that the next corner is the top-right, and the first stroke of the letter "z" is drawn (Figure 7.4b). Having snapped to the top-right corner, the cursor now moves diagonally at an angle that indicates the bottom-left corner, and the second stroke is drawn (Figure 7.4c). Note that for the third stroke, the cursor moves towards the *outside* of the virtual EdgeWrite square, but at an angle that still indicates the bottom-right corner. The completed "z" is thus drawn (Figure 7.4d).

In essence, a series of short crossing tasks forms a letter. But clearly, much depends on how the circle is partitioned and how θ is interpreted. For instance, consider the move across the bottom of the "z" in Figure 7.4c. At what angle θ should the cursor instead move diagonally back to the top-right?

Figure 7.5 shows the next-corner outcomes for different departure angles from the bottom-left corner of the EdgeWrite square. The same scheme can be easily extrapolated to the other three corners of the EdgeWrite square.

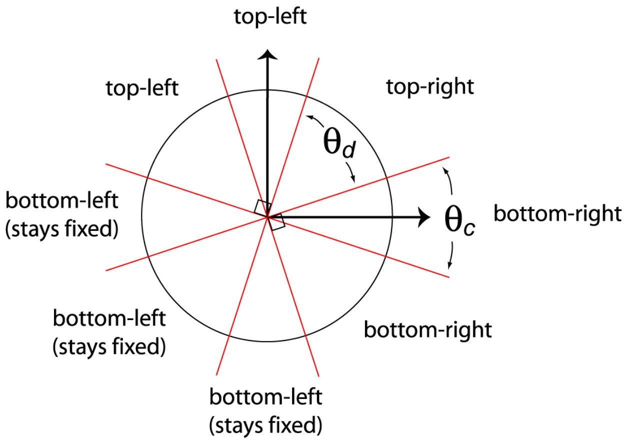

top-left
top-left
top-right
θ_d
bottom-left (stays fixed)
θ_c
bottom-right
bottom-left (stays fixed)
bottom-right
bottom-left (stays fixed)

Figure 7.5. Next-corner outcomes for different angles of departure from the bottom-left corner of the EdgeWrite square. θ_d is the diagonal angle and θ_c is the cardinal angle.

Three features of the design in Figure 7.5 are important. First, any movement *left*, diagonal *down-and-left*, or *down* keeps the cursor fixed at the extreme bottom-left corner. More generally, if the cursor moves at an angle that, were it to cross the circumference it would remain in the same corner, it is held fixed in the center of the circle. This will always hold for $(180 - \theta_d)°$ of the circle, where θ_d is the angular amount allotted to each diagonal. This "pinning" to the circle's center ensures that one always starts from a corner's center when moving to a different corner.

Second, the value in having big cardinal angles (θ_c) is that users do not always move perfectly to the *left*, *right*, *up* or *down*. Larger values of θ_c create more room inside the EdgeWrite square to move in the cardinal directions. However, adding to θ_c detracts from the diagonals θ_d because $\theta_c + \theta_d = 90°$.

Third, although increasing θ_c gives more room *inside* the square to move in the cardinal directions, one always has 90° with which to move along an edge, regardless of

θ_c. That's because one always has the angles to the *outside* the EdgeWrite square, as in the bottom-left image in Figure 7.4c. So although θ_d and θ_c are tradeoffs, a total of 90° remains for moving along an edge. In practice, however, having a larger θ_c provides a "cardinal error band," giving more room to move inside the EdgeWrite square.

It is important to emphasize that despite these underlying mechanics, users neither see a visual cursor nor aim for particular angles when writing with Trackball EdgeWrite. Instead, users simply pulse the ball towards intended corners, creating the feeling of fluid writing.

7.2.3 Determining the First Corner

The example in Figure 7.4 assumes that one starts in the top-left corner. But of course, not all EdgeWrite letters start there. This begs the question of how one *begins* a letter, since unlike a stylus landing on a PDA or a finger landing on a touchpad, the trackball cursor is persistent and cannot simply "appear" in the starting corner.

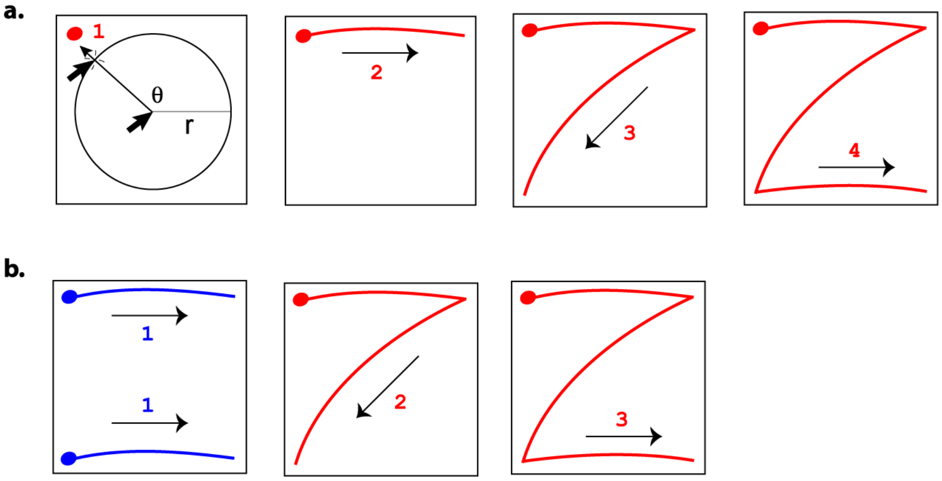


Figure 7.6. Two different schemes for determining the first corner.

Two different schemes were implemented for determining the first corner in Trackball EdgeWrite. The first scheme is to have users enter the initial corner as if they were starting from the center of the EdgeWrite square (Figure 7.6a). Although this scheme adds an extra movement to the start of every letter, the accuracy demands are low because the user has a full 90° with which to indicate the starting corner.

In the second scheme, one assumes one *starts* in the appropriate corner, and the software disambiguates that corner as the gesture unfolds (Figure 7.6b). For example, in making a “z”, one would first move to the right. At this point, one may have intended to move along either the top or bottom edge. Both possibilities are entertained until the next move is diagonally *down-and-left*, at which point the ambiguity is resolved. Gestures that occur along a single edge, like “i”, space, and backspace, never resolve into unambiguous strokes. Such gestures can be defined to be the same on both sides of the square, so the ambiguity can be made irrelevant.

Although the second scheme requires one less pulse per letter than the first scheme, it proves slower and more difficult in practice because the initial stroke has *eight* possible outcomes (Figure 7.7), not just four as in the first scheme. Even when the angular regions of the circle are adjusted proportional to the probability of beginning a letter in that region, the second scheme proved too difficult to perform reliably.

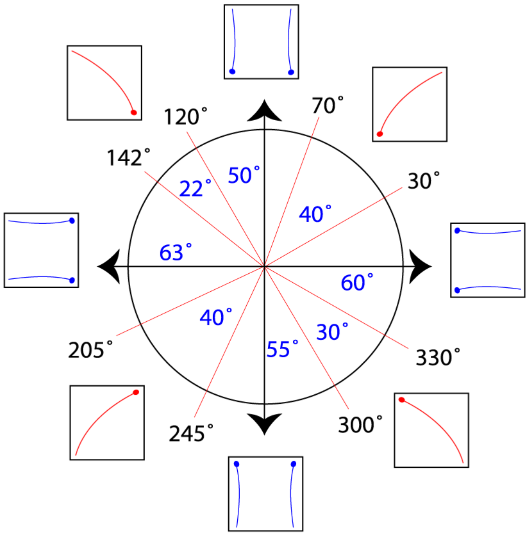


Figure 7.7. With the second scheme, users have eight angular regions to choose from on the first pulse of the trackball. Region sizes are weighted by letter probabilities.

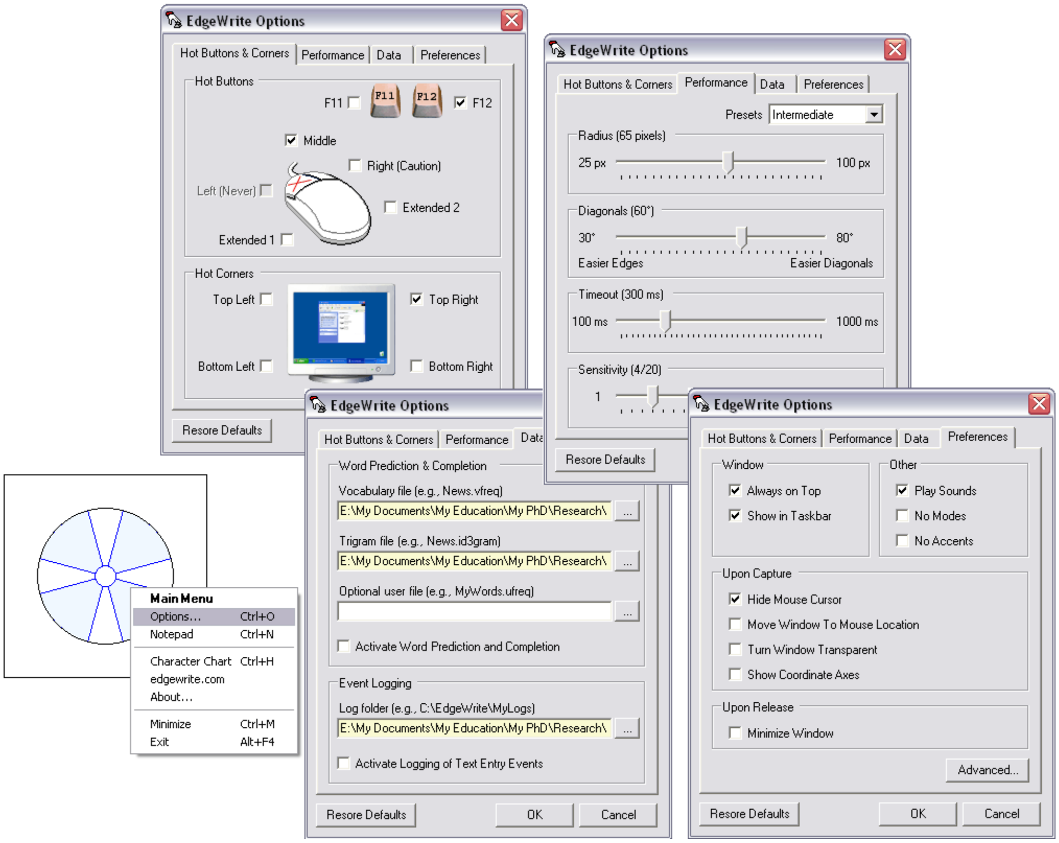
An interesting theoretical result is that an analysis of each scheme actually shows the first scheme to be about 3 WPM *faster* despite the additional stroke required for each letter! This is because of the high accuracy required for the second scheme’s initial stroke. Thus, the first scheme was preferred. This was an unexpected theoretical finding, as we had initially assumed the second scheme would be about 30% faster based on the reduced number of strokes. That both empirical and theoretical results proved otherwise shows the importance of basing designs on quantitative and theoretical measures instead of intuition. The full theoretical analysis for the first scheme is given in §7.3.1.

7.2.4 Interaction Design

Two other key issues arose when building Trackball EdgeWrite: how to switch between mousing and writing, and how to segment between letters.

7.2.4.1 Capture and Release

Trackball EdgeWrite is designed to run invisibly in the background until it is needed for text entry. When the trackball is being used for mousing, the mouse cursor is said to be "released." When the trackball is being used for writing, the cursor is said to be "captured." The mouse can be captured by clicking a dedicated trackball button (a "hot button"), pressing a dedicated keyboard key (a "hot key"), or by dwelling in a corner of the desktop (a "hot corner"). Hot buttons, keys, and corners are configurable in the application's preferences dialog (Figure 7.8).



Hot Buttons
F11 □ ☑ F12
☑ Middle
□ Right (Caution)
Left (Never) □
□ Extended 2
Extended 1 □

Hot Corners
Top Left □ ☑ Top Right
Bottom Left □ □ Bottom Right

Presets Intermediate
Radius (65 pixels) 25 px 100 px
Diagonals (60°) 30° 80° Easier Edges Easier Diagonals
Timeout (300 ms) 100 ms 1000 ms
Sensitivity (4/20) 1

Word Prediction & Completion
Vocabulary file (e.g., News.vfreq) E:\My Documents\My Education\My PhD\Research\
Trigram file (e.g., News.id3gram) E:\My Documents\My Education\My PhD\Research\
Optional user file (e.g., MyWords.ufreq)
□ Activate Word Prediction and Completion

Event Logging
Log folder (e.g., C:\EdgeWrite\MyLogs) E:\My Documents\My Education\My PhD\Research\
□ Activate Logging of Text Entry Events

Window
☑ Always on Top
☑ Show in Taskbar

Other
☑ Play Sounds
□ No Modes
□ No Accents

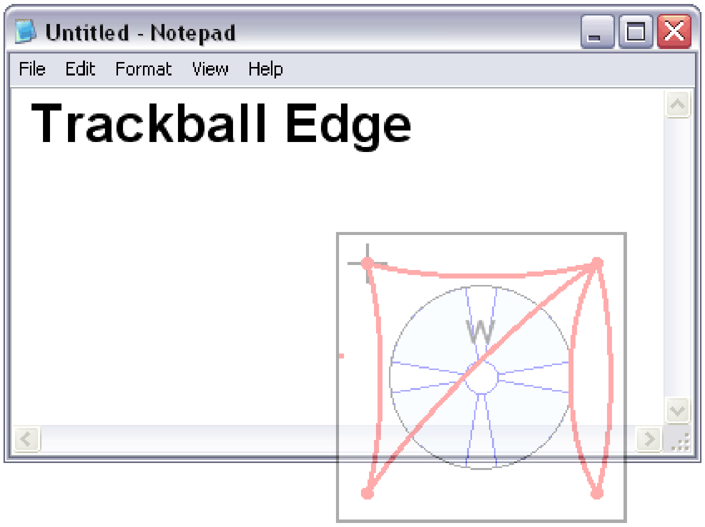
Upon Capture
☑ Hide Mouse Cursor
□ Move Window To Mouse Location
□ Turn Window Transparent
□ Show Coordinate Axes

Upon Release
□ Minimize Window

Figure 7.8. Trackball EdgeWrite supports multiple options for capturing the trackball with hot buttons, hot keys, and hot corners (top left). Other options are also available.

When captured, the mouse cursor vanishes and the Trackball EdgeWrite window appears (Figure 7.9). To release the cursor, the user can click any trackball button or perform a dedicated release stroke. Then the EdgeWrite window disappears and the mouse cursor is restored to its previous location. Thus, Trackball EdgeWrite never requires that it be navigated to; instead, it comes to the user when summoned.

Other preferences exist for setting performance parameters like the radius of the crossing circle, the amount of diagonal degrees, the segmentation timeout, and the mouse sensitivity. Additional options include for window transparency (Figure 7.9), playing sounds, and controlling where the EdgeWrite window appears.



Trackball Edge

Figure 7.9. A semi-transparent Trackball EdgeWrite is writing in Notepad. Notepad retains the input focus even though EdgeWrite receives mouse events.

7.2.4.2 Segmentation

Trackball EdgeWrite letters are segmented when the underlying mouse cursor stops moving for a very short time. Timeout values can be set in the preferences dialog (Figure 7.8), and range from 100 ms for experts to 750 ms for novices. This simple scheme works reliably since the timeout timer is restarted after every mouse movement. A beginner can still "stop and think" while making a gesture by slowly rolling the trackball toward the corner they're already in; this will not move them to a new corner but it will prevent the stroke from segmenting. Subjectively for the user, segmentation involves a slight pause between letters.

7.2.4.3 Advanced Features

Trackball EdgeWrite supports all advanced EdgeWrite features including continuous recognition feedback (§3.4.1), non-recognition retry (§3.4.2), slip detection (§3.4.3), and Fisch word-level stroking (§3.5). The last item is explained in the next section.

7.2.5 Fisch in Trackball EdgeWrite

Like Stylus EdgeWrite (§4.2.5), Trackball EdgeWrite has an implementation of Fisch word-level stroking. In contrast to Stylus EdgeWrite, however, Trackball EdgeWrite does not allow for loops. The adaptation of Fisch therefore requires users to first segment their letter strokes with a brief pause and then "pulse" the trackball into the corner of the desired word completion. Although this is a two-step process, the same words always appear in the same corners for the same preceding letters, enabling these "compound strokes" to be used reliably.

As with all Fisch designs, character-level stroking remains unchanged in Trackball EdgeWrite. This is important for two reasons: (1) it allows current users to remain effective with the software, and (2) it allows users to gradually ramp up to using word completions at their own pace.

While a user strokes, candidate words are shown at the four corners of the EdgeWrite square (Figure 7.10). In order to provide appropriate completions, the current stroke is recognized after each corner is entered.

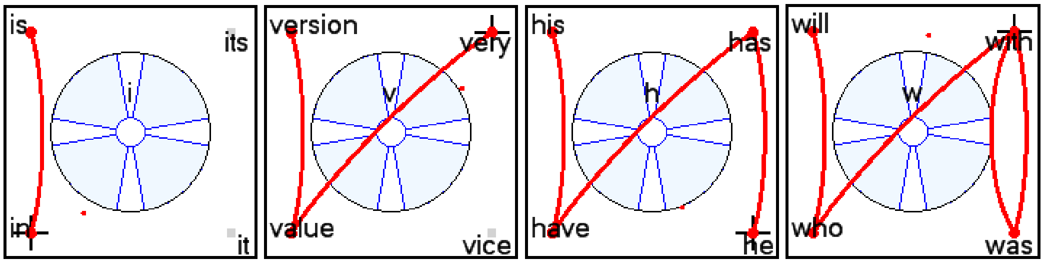


Figure 7.10. Candidate words shown while stroking a "w" include words that begin with an "i", "v", and "h" along the way, since these letters are subsets of "w".

After a stroke is segmented and a letter is produced, the user can continue stroking letters or, alternatively, make a short "pulse" to select a word. This pulse is a simple motion from the center of the EdgeWrite square to the corner containing the desired word, followed by a brief pause for segmentation.

When the text cursor is positioned after a word that has not been completed, a word-level backspace from right-to-left across the bottom of the EdgeWrite square erases the entire previous word. However, if the previous word was completed, then a word-level backspace will erase *only* the completed suffix, restoring the word completions as they appeared before selection. Importantly, the restored completions appear in the *same* corners as before, allowing the user to quickly select a different completion if desired. Completed words always remember the character position at which they were completed (if any), allowing future word-level backspace strokes to remove completed suffixes.

In addition to showing frequency-based word completions according to the Fisch design (§3.5.2), Trackball EdgeWrite also shows context-dependent word *predictions* after a word ends. Word predictions are, by definition, contextual and thus cannot be stroked by feel. However, the selection mechanism for word predictions is the same as for word completions: stroke into a single corner and then pause briefly to segment.

7.2.6 Implementation

Trackball EdgeWrite is implemented in C# using DirectInput 9.0c to receive mouse events in the background, which is necessary so that focus can remain on a target application (e.g. Microsoft Word) even while Trackball EdgeWrite receives mouse input. A fair amount of code in Trackball EdgeWrite is devoted to keeping the input focus on the target window and *off* Trackball EdgeWrite itself. Cases that must be handled include when the user left-clicks while captured, or left-drags to reposition the Trackball EdgeWrite window. After these actions, focus is returned to the target window.

Recognized characters are sent through the low-level keyboard input stream. Trackball EdgeWrite works with any pointing device, but is best suited for devices without absolute position, e.g. trackballs and isometric joysticks. With a miniature trackball or an isometric joystick, EdgeWrite provides full text entry in very little space.

7.3 Evaluation

This section presents multiple evaluations of Trackball EdgeWrite. The first is a theoretical analysis based on goal crossing. After that are three empirical evaluations, one for able-bodied users and two for a user with a spinal cord injury. The EdgeWrite alphabet *after* the guessability study was used (§3.3.3), and all advanced features (§3.4) had been implemented. The first two evaluations are for character-level Trackball EdgeWrite only. The third evaluation is for word-level Trackball EdgeWrite.

7.3.1 Theoretical Model

The time it takes to make a single letter can be predicted using Accot and Zhai's discovery that crossing follows the Fitts' formulation in Equation 2.1 (Accot and Zhai 1997, Accot and Zhai 2002).

For diagonal movement, the size W_d of the crossing goal is:

$$W_d = \frac{\theta_d}{360} \cdot 2\pi r \tag{7.2}$$

The index of difficulty for diagonal movement ID_d is therefore:

$$ID_d = \log_2\left(\frac{r}{\frac{\theta_d}{360} \cdot 2\pi r} + 1\right) \tag{7.3}$$

Thus, the time for diagonal movement T_d is given by:

$$T_d = a + b \cdot \log_2\left(\frac{180}{\theta_d \cdot \pi} + 1\right) \tag{7.4}$$

For movement in the cardinal directions, recall that there is a constant target size of 90°. The target width W_c is therefore:

$$W_c = \frac{90}{360} \cdot 2\pi r \tag{7.5}$$

So, the index of difficulty for cardinal movement ID_c is:

$$ID_c = \log_2\left(\frac{r}{0.5\pi r} + 1\right) \tag{7.6}$$

The movement time along cardinal edges T_c is thus given by:

$$T_c = a + b \cdot 0.710719 \tag{7.7}$$

For the pulse from the center to the first corner, there are four target angles each of 90°. Thus, the movement time T_f for the pulse to the first corner is the same as T_c:

$$T_f = a + b \cdot 0.710719 \tag{7.8}$$

Using Equations 7.4, 7.7, and 7.8, the theoretical movement time for each EdgeWrite letter and SPACE can be computed. For example, using a typical 65° for diagonal angles θ_d and 150 ms for segmentation timeout τ, the time (in ms) to enter the letter "z" is given by:

$$\begin{aligned} T_{"z"} &= T_f + T_c + T_d + T_c + \tau \\ &= 3 \cdot T_c + T_d + \tau \\ &= 4a + 3.04402b + 150 \end{aligned} \tag{7.9}$$

Obviously, the result for $T_{\text{“}z\text{”}}$ (and any other letter) is dependent upon our choice of *a* and *b*—our Fitts' regression coefficients. To my knowledge, no studies have elicited these coefficients for continuous reciprocal goal crossing with a trackball. However, Accot and Zhai have done so for a stylus (Accot and Zhai 2002), and also for a stylus and a trackball for non-reciprocal straight-tunnel steering (Accot and Zhai 1999). Extrapolating from these studies, *a* and *b* can be estimated for continuous reciprocal goal crossing with a trackball. The values used are: *a* = -363.0, *b* = 642.1.

To compute the time (in ms) of the "average character" T_{avg}, each character's time T_i is weighted by its linguistic frequency F_i.

$$T_{avg} = \sum_{i \in C} T_i \cdot F_i \tag{7.10}$$

In Equation 7.10, *i* is a character in character set *C*. For simplicity, the primary letters shown in Figure 3.9 are used for *C*, omitting "ç" and BACKSPACE, but including SPACE. Next, Equation 7.11 is used to calculate the theoretical speed for the method (the numerator is ms/min):

$$wpm = \frac{60{,}000}{5 \cdot T_{avg}} \tag{7.11}$$

Using a typical θ_d = 65° and τ = 150 ms, Equation 7.11 yields 23.1 WPM.

Note that this rate represents "perfect" entry—no errors, no error correction, and no hesitation between letters other than the time τ required for segmentation. Also, trackballs differ significantly in their sizes, friction, gain settings, etc. Thus, our estimate can only be considered a ballpark measure. However, it is useful for comparing different stroking schemes, such as the two first-corner schemes discussed in §7.2.3.

This theoretical model can be extended to incorporate Fisch's frequency-based word completions. To do this, a computer program is needed to calculate the WPM of each word in Trackball EdgeWrite's list of 19,122 words (§3.5.3) assuming that each completion is selected when it appears. The word list is large enough to accommodate most words used in everyday English.

The speed S_{cps} for our corpus can be calculated using Equation 7.12:

$$S_{cps} = \sum_{w \in K} \left(\frac{|w| + 1}{T_w} \times F_w \right) \times 1000 \qquad (7.12)$$

Here, S_{cps} is the weighted speed of text entry in characters per second (CPS), w is a word in corpus K with length $|w|$, T_w is the time to write word w (in ms), and F_w is the frequency of word w such that $\Sigma F_w = 1.00$. The "+1" in the numerator is for the space that is added after a completion is selected, and the "×1000" converts from characters per ms (CPMS) to CPS.

To calculate T_w (in ms) for each word in the corpus, we calculate the time T_ℓ to perform each letter $\ell \in w_p$, where w_p is the minimum prefix that will show w as a completion ($1 \leq |w_p| \leq |w|$). To this we add T_{select}, the time to select the completion itself, which is equivalent to T_f (Equation 7.8). As in Equation 7.9, $\tau = 150$ ms must be included in the sums for T_ℓ and T_{select} to account for the segmentation time after a letter or word selection is made. Note that the computation of T_ℓ is akin to Equation 7.9 for each letter (*a*-*z*) and SPACE. Thus, the time T_w to write word w is:

$$T_w = \left(\sum_{\ell \in w_p} T_\ell \right) + T_{select} \qquad (7.13)$$

For words which themselves are prefixes of at least four other more common words (e.g. "ad"), there is no such w_p that will show w as a completion. For these words, w must be entered in full along with a trailing space:

$$T_w = \left(\sum_{\ell \in w} T_\ell \right) + T_{space} \tag{7.14}$$

To convert S_{cps} in Equation 7.12 from CPS to WPM, we use the standard definition of 5 characters per word:

$$S_{wpm} = S_{cps} \times \frac{60\,\text{sec}}{1\,\text{min}} \times \frac{1\,\text{word}}{5\,\text{chars}} \tag{7.15}$$

Using Equations 7.12–7.15, the model yields an upper-bound entry rate of 52.5 WPM. This is 227% faster than the 23.1 WPM obtainable with only character-level strokes. Like before, this result is unachievable by a real user. It represents perfect entry, lacking considerations for hesitation, cognitive processes, visual search, slips, or mistakes. Still, it is useful as an upper-bound for theoretical comparisons with prior models.

For a more realistic estimate, our model can be enriched with a term for visual search time based on the Hick-Hyman law (Hick 1952, Hyman 1953). This term T_n is added after the entry of every letter ℓ and represents the time it takes for a user to find their word amidst n choices, where n is the number of completions offered for the current prefix ($0 \le n \le 4$). Drawing on prior rationale (Soukoreff and MacKenzie 1995), the formula for T_n (in ms) is:

$$T_n = 0.2 \times \log_2(n) \times 1000 \tag{7.16}$$

Incorporating the Hick-Hyman law, Equations 7.13 and 7.14 become:

$$T_w = \left(\sum_{\ell \in w_p} \left(T_\ell + T_n \right) \right) + T_{select} \tag{7.17}$$

$$T_w = \left(\sum_{\ell \in w} \left(T_\ell + T_n \right) \right) + T_{space} \tag{7.18}$$

Using Equations 7.16–7.18, the result drops 36.2% from 52.5 WPM to 33.5 WPM. This is still 45.0% faster than our character-level result of 23.1 WPM. Note, however, that even with the addition of visual search time, this result still represents perfect entry.

A limitation of this model is that it only accounts for word completion, not word prediction. However, modeling word prediction is more difficult because it depends on context, including the user's adaptive cache of recent words. Such a model is beyond the current scope of this theoretical analysis.

7.3.2 Novice Able-bodied Use

For baseline comparisons, a study of Trackball EdgeWrite was conducted with able-bodied subjects who knew the EdgeWrite alphabet but had never used trackballs. Their knowledge of the EdgeWrite alphabet came from a previous study with a different EdgeWrite version. This allowed trackball performance to be separated from learning the alphabet. The study was not intended to compare two competitor methods, but to provide an estimate of the performance of Trackball EdgeWrite for trackball novices.

7.3.2.1 Method

Three subjects ages 27–33 took part in the study. All were daily computer users who owned a desktop mouse. None had ever owned or used a trackball. They were paid $15.

MacKenzie and Soukoreff's text phrases (MacKenzie and Soukoreff 2003) were shown in Courier 20pt on an IBM A21p laptop set to 1280×1024 resolution. The phrases were transcribed with Trackball EdgeWrite using a Kensington Expert Mouse trackball (Figure 7.1). The pointer speed was set to 40% of maximum on the mouse control panel. Backspace was supported and subjects were not required to stay synchronized with the presented text. They were told to balance speed and accuracy.

Subjects practiced for 45 minutes with Trackball EdgeWrite before transcribing 15 test phrases. During practice, the three subjects transcribed 58, 26, and 40 practice phrases, respectively. Although they knew the EdgeWrite letters, they had a low level of comfort with the trackball device. All of them said that they felt uncomfortable with the device itself and needed more time to become proficient with it.

7.3.2.2 Results

Subjects' speeds and error rates are shown in Figure 7.11 and Figure 7.12. As an additional point of comparison, my performance is shown at the right of each graph. For the three subjects, the average entry rate was 9.82 WPM. Their average respective uncorrected, corrected, and total error rates were 0.59%, 3.31%, and 3.90%. These total error rates are quite low (<4%) for a gestural method with an unfamiliar device.

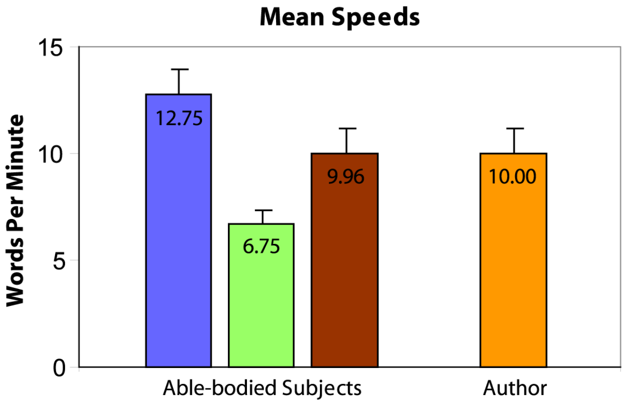


Figure 7.11. Average speeds for three able-bodied subjects using character-level Trackball EdgeWrite. Subjects knew the alphabet but had never used trackballs.

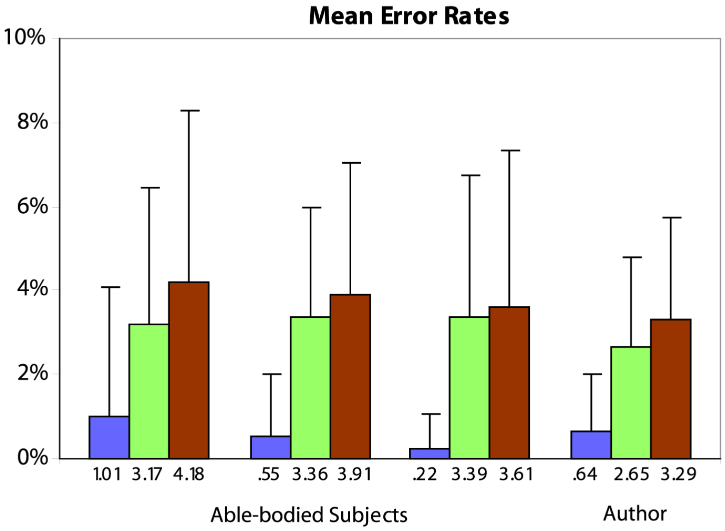


Figure 7.12. Average error rates for each able-bodied subject. From left to right, error rates within each subject are uncorrected, corrected, and total.

7.3.3 Longitudinal Motor-impaired Use

Although ~10 WPM is not particularly fast, a trackball user with motor impairments resigned to using an on-screen keyboard at 4–5 WPM would probably welcome that speed. In view of this, Trackball EdgeWrite was evaluated over the course of nine participatory design sessions conducted from May to November 2005 with a veteran trackball user who will be called "Jim." Jim has a spinal cord injury that reduces the dexterity in his arms, hands, and fingers such that he cannot satisfyingly use a conventional keyboard or mouse. For 15 years he has relied upon trackballs and on-screen keyboards for computer access. His favorite trackball is the *Stingray* from CoStar Corporation (Figure 7.13).

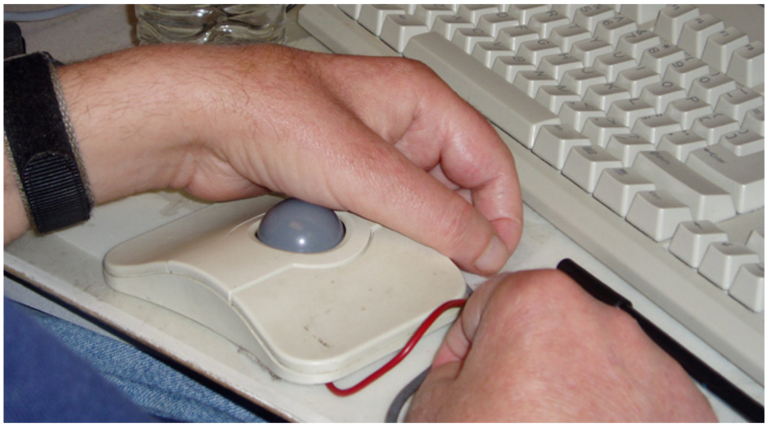


Figure 7.13. Jim's favorite trackball is the Stingray because he can press its wide left button with the palm of his left hand while his left thumb rolls the ball to drag.

Jim is now 46 years old and is still an avid trackball user. But he is a reluctant on-screen keyboard user. For extended periods of text entry, Jim uses *Dragon Naturally Speaking*, but he nonetheless frequently relies on an on-screen keyboard. He complains that his speech recognition often "acts up" and ceases to work well. Sometimes his voice is altered from medications or fatigue and his recognition rates drop. For short replies to emails, putting on a headset and microphone is an arduous task, so Jim uses an on-screen keyboard when he needs to enter just a few words. Also, certain tasks don't work well for him with speech recognition, like naming files, entering email addresses, editing proper names, entering spreadsheet formulae, and filling out web forms. Thus, Jim's text entry solution has been a mixture of speech recognition and an on-screen keyboard using a trackball to dwell over letters. At one time he used *Click-N-Type* (http://www.lakefolks.org/cnt), but not long before this study started he had switched to the Microsoft Accessibility Keyboard.

Jim has three main complaints about using an on-screen keyboard. First, the visual attention required is enormous, as he constantly must look from the keyboard to his document and back. Second, moving over keys to dwell requires that the dwell time be long enough to avoid unwanted keys but short enough to produce text quickly, a difficult balance to strike. He currently uses 500 ms as the dwell time. Third, the tedium of making repeated keyboard selections is, according to Jim, "mind numbing." Although word prediction can help, Jim feels that word prediction slows him down as often as it speeds him up, a sentiment consistent with findings in the literature (Goodenough-

Trepagnier *et al.* 1986, Soede and Foulds 1986, Horstmann and Levine 1991). Plus, word prediction adds more items to visually scan, adding to the tedium (Anson *et al.* 2005).

7.3.3.1 Quantitative Results

To ensure that the design iterations were beneficial, short "checkpoint studies" were conducted when meeting with Jim. During these studies, Jim would use the latest version of Trackball EdgeWrite to enter 5–7 test phrases. He would also enter the same number of test phrases with the Microsoft Accessibility Keyboard. Word prediction was not available in either method at this stage. Both methods were controlled by Jim's personal trackball (Figure 7.13), and neither method required any button presses. Jim's speed and accuracy results are shown in Figure 7.14.

Note that the checkpoint studies were not evenly spaced in time, particularly in the jump from week 4 to week 10 and week 22 to week 32. During the first gap, Jim did not use his computer. Thus, the data for week 10 can be viewed as retention results. Week 10 was the only week after week 1 in which the two methods were nearly equal in speed (Figure 7.14a). The second gap was simply a break in testing.

After week 1, Jim's speeds were consistently higher with EdgeWrite than with the on-screen keyboard (Figure 7.14a). His average speed across all weeks with EdgeWrite was 5.61 WPM (σ=1.40, max=8.25). With the keyboard it was 4.86 WPM (σ=1.17, max=6.90). A Wilcoxon sign test shows that his EdgeWrite speeds were significantly faster than his on-screen keyboard speeds (z=-19.5, p<.02).

The drop in performance during week 15 was due to a nagging tremor in Jim's hand. The drop in speed occurred about evenly for both methods. Accuracy results, on the other hand, show that both uncorrected and corrected errors were worse for the on-screen keyboard than for Trackball EdgeWrite (Figure 7.14c and Figure 7.14d), suggesting that EdgeWrite accuracy may be less affected by such tremors.

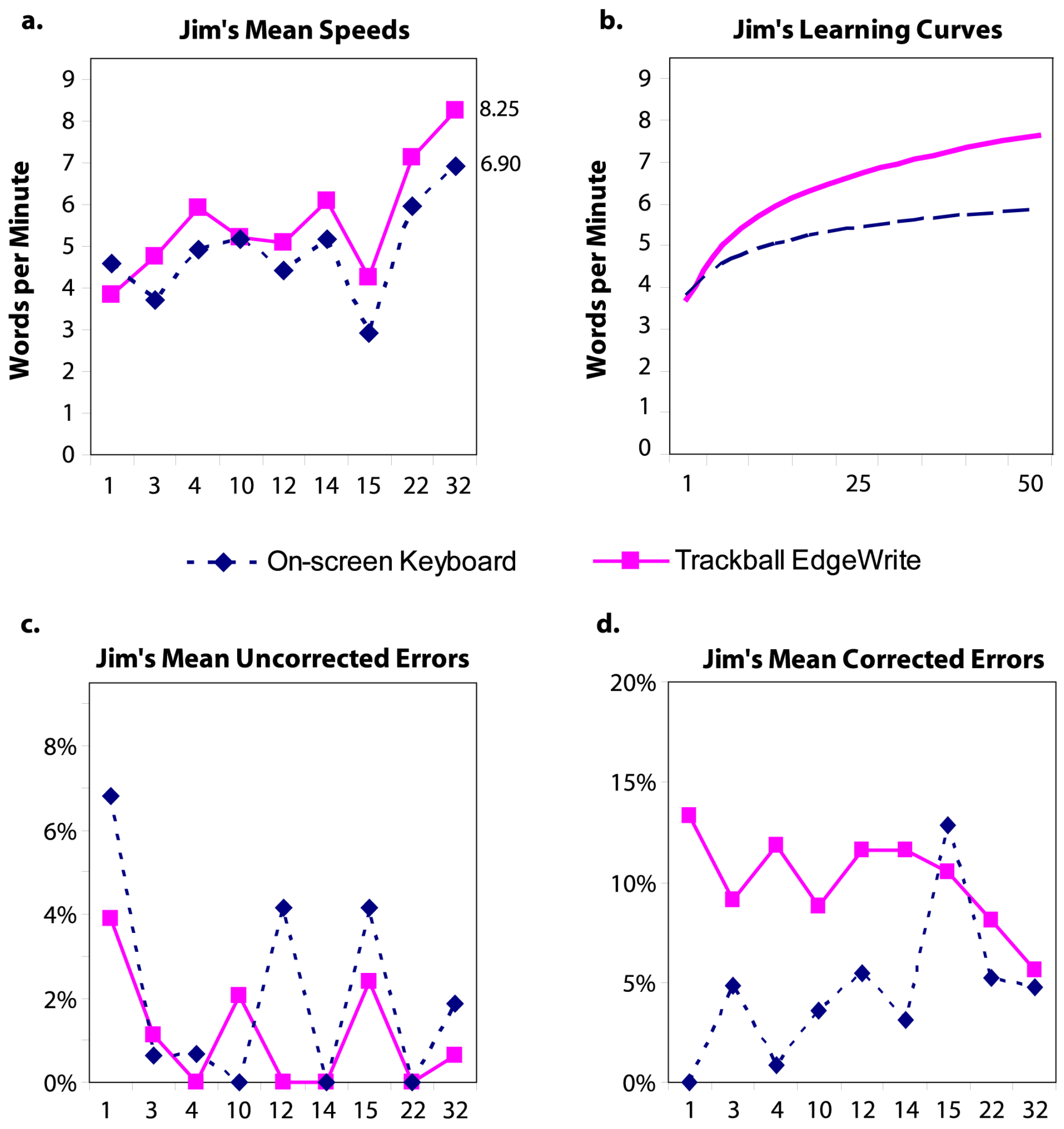


Figure 7.14. Jim's performance with the two trackball methods over many weeks.

Jim's speeds are modeled as learning curves and extended to week 50 in Figure 7.14b. Although such an extension is speculative, it gives us an idea of the possible trends for Jim's data. The power law equations and R^2 values for Jim's curves are:

- $y = 3.713x^{0.1850}$ $R^2 = 0.53$ Trackball EdgeWrite
- $y = 3.796x^{0.1127}$ $R^2 = 0.21$ On-screen Keyboard

Jim's average uncorrected error rate was 1.12% (σ=1.39) with EdgeWrite and 2.03% (σ=2.45) with the on-screen keyboard (Figure 7.14c). Although the trend was in EdgeWrite's favor, a Wilcoxon sign test is not quite significant (*z*=8.0, p=.22).

Jim's average corrected error rate was 10.06% (σ=2.37) with EdgeWrite and 4.52% (σ=3.68) with the keyboard (Figure 7.14d). A Wilcoxon sign test for corrected errors yields a significant result in favor of the on-screen keyboard (z=-20.5, p<.02). However, an analysis of variance indicates that *Session* had a larger effect on decreasing corrected errors for EdgeWrite ($F_{1,7}$=5.21, p=.05) than for the on-screen keyboard ($F_{1,7}$=2.92, p=.13).

Although corrected errors—which are any letters backspaced during entry—are higher with Trackball EdgeWrite, they are not necessarily damaging if the correction operation is efficient (Soukoreff and MacKenzie 2003). Of course corrected errors should be reduced, but the main tradeoff is between speed and *uncorrected* errors, which are errors left in the transcribed string. A method can be successful even if it quickly produces and repairs errors during entry if, in the end, it yields more accurate text in less time, which is the case here.

7.3.3.2 Qualitative Results

In general, the time Jim spent using EdgeWrite each week varied according to his personal computer use and his text entry needs. With Jim's permission, Trackball EdgeWrite wrote log files on his computer whenever it was being used. These logs show that EdgeWrite was always left running in the system tray as of week 3, and was used intermittently over the course of most weeks. The total use per week varied from just a few minutes to a few hours. Jim's usage increased in the later sessions.

Jim no longer uses an on-screen keyboard, but keeps Trackball EdgeWrite running at all times, able to be called up at a moment's notice. To begin writing, Jim simply places the mouse cursor in the top-right corner of his screen, waits a moment, and then watches as the cursor is "captured" automatically within the EdgeWrite square. This notion of using hot corners to capture the mouse was Jim's idea. When Jim is done writing, he makes a dedicated stroke (see Appendix A) to "release" his mouse cursor and dismiss the EdgeWrite window. The release stroke was also Jim's idea. Thus, Jim can operate Trackball EdgeWrite without clicking mouse buttons. Jim also helped to improve the feel of the software with suggestions for altering the on-screen depiction and stroke feedback.

Jim's reasons for preferring Trackball EdgeWrite over an on-screen keyboard are, in his words:

- “The on-screen keyboard is so terribly boring. EdgeWrite is fun, like a video game. The on-screen keyboard is not fun. I don’t care which is faster.”

- “With EdgeWrite, you can keep your eyes on your document and just write as you would holding a pencil. I don’t feel disabled when I’m using EdgeWrite.”

- “The on-screen keyboard requires too much visual scanning and concentration. In EdgeWrite, if you know the letter, you can just bang it out by feel.”

- “I feel like I can write again.”

These results show that Trackball EdgeWrite is faster and produces just as accurate, if not more accurate, text than an on-screen keyboard. Having reached a maximum speed of 8.25 WPM in week 32, it seemed unlikely that Jim could go much faster with the character-level version of Trackball EdgeWrite. After further development, Fisch word-level stroking (§3.5) was incorporated into the application. What follows is a second-stage evaluation with Jim using word prediction and completion.

7.3.4 Motor-impaired Use with Word-level Stroking

In order to empirically test Fisch word completion in Trackball EdgeWrite, two more evaluations were conducted with Jim. The first was a comparison to the WiViK on-screen keyboard (Shein *et al.* 1991). The second was an analysis of Jim’s log files over two months of intermittent use. These evaluations began about 3 months after the end of the character-level study, and about 2 weeks after Jim had been introduced to word-level stroking in Trackball EdgeWrite.

7.3.4.1 Apparatus

Since Jim’s preferred on-screen keyboard, the Microsoft Accessibility Keyboard, does not have word prediction or completion, the popular *WiViK* on-screen keyboard (http://www.wivik.com) was configured to match Jim’s desired settings: 550 ms dwell, about 650×250 pixels in size, and no “dead space” between keys. For word prediction and completion, WiViK uses a program called *WordQ*, which was loaded with its fullest dictionary, the “US Advanced” database containing 15,000 words. WiViK shows a vertical 6-item word list to the left of the keyboard (recall Figure 2.11). The same action for selecting a key selects a word in the word list—in Jim’s case, by hovering for 550 ms.

7.3.4.2 Procedure

The study was a single-subject 2-factor design, with factors for *Method* (WiViK, EdgeWrite) and *Word Prediction* (on, off). Jim did the word prediction versions second within both methods. A coin toss determined that he would use WiViK first. Thus, the technique order was: WiViK, EdgeWrite, WiViK+WP, EdgeWrite+WP. Jim entered 3 practice phrases and 8 test phrases in each condition. Each phrase was approximately 30 characters long.

7.3.4.3 Quantitative Results

Figure 7.15a shows Jim's speeds for the four conditions in the current study. It also shows Jim's peak speeds from the prior study (§7.3.3). Note the substantial speedups of both methods due to word prediction and completion.

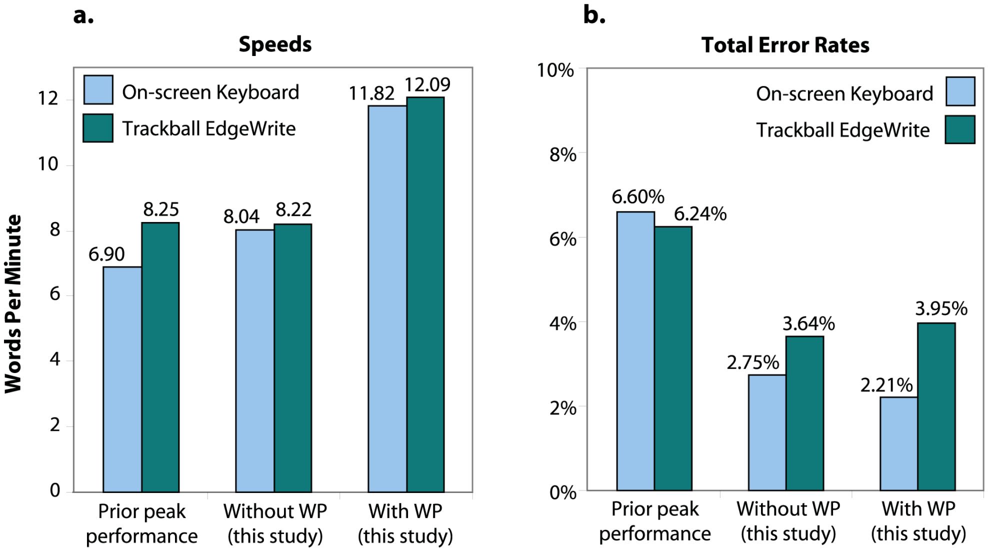


Figure 7.15. Jim's speeds and corresponding total error rates with an on-screen keyboard and word-level Trackball EdgeWrite.

Figure 7.15b shows corresponding total error rates. However, because Jim fixed almost every error during entry, these total error rates are really just corrected error rates. Thus, Trackball EdgeWrite is producing similarly accurate text in a tad less time, albeit with more errors made (and fixed) along the way.

A Wilcoxon sign test for speed is not significant (*z*=3.0, p=.25). However, the general trend is in favor of Trackball EdgeWrite. This advantage is only slight for the current study, however, probably because WiViK is superior to Jim's preferred keyboard, the Microsoft Accessibility Keyboard, although WiViK was configured with Jim's settings.

A Wilcoxon sign test for total errors is also not significant (*z*=2.0, p=.50). However, both methods were producing error-free text in the end, since uncorrected errors for both methods were ~0.0%. It is interesting that character-level Trackball EdgeWrite's errors were low in the current study even *without* word completion, probably because Jim has had more practice since his prior peak performance at the end of the last study.

It is worth noting that the speed of WiViK improved 32.0% with word prediction compared to without. This highlights the strength of WiViK's commercial word prediction and completion technology.

Taken together, these results show a 46.5% increase in speed and a 36.7% decrease in errors for word-level Trackball EdgeWrite compared to Jim's prior peak performance with character-level Trackball EdgeWrite. The results also show that word-level Trackball EdgeWrite is 75.2% faster and 40.2% more accurate than Jim's prior peak performance with his own on-screen keyboard. Finally, the results show that word-level Trackball EdgeWrite is competitive with a major commercial product, the WiViK on-screen keyboard with word prediction and completion.

7.3.4.4 Qualitative Results

Jim was asked to describe his experience with each of the four conditions in his own words:

- *WiViK*: "[Y]ou are constantly either scribbling around so you don't accidentally trigger the wrong letter, checking to see if you typed the right thing, or looking for the next key to hover over. Too much work both mentally and visually."

- *WiViK+WP*: "Somewhat of a relief to hover over large words but it just increased the amount of mental and visual work required. [It's] one more section of the screen you need to scan constantly. Only thing is, I wish EdgeWrite had its vocabulary."

- *EdgeWrite*: "EdgeWrite without word prediction is like using a 286 or something. It's much better than a keyboard or an on-screen keyboard, but the ultimate is when you can flick the cursor into a corner and just pop the rest of the word in."

- *EdgeWrite+WP*: "The best thing about EdgeWrite is there is no eye strain or constant scanning between programs, letters, words, etc. The word choices are right there where your eyes already are. It actually helps you stay focused on what you're writing."

Jim's sentiments confirm what prior studies of on-screen keyboards have found: that they are exceedingly tedious and visually intense (Anson *et al.* 2005). Although word prediction and completion improved WiViK's speed by 32.0%, they did not resolve these drawbacks. Trackball EdgeWrite, on the other hand, proved to be just as fast but without the same visual tedium because it is gestural instead of selection-based, supporting writing "by feel" rather than "by sight." Jim also expressed a clear preference for EdgeWrite with word prediction and completion.

7.3.4.5 Analysis of Extended Use

A single-session lab study allows one to formally quantify speed and errors, but it is over the long-term that Trackball EdgeWrite must prove to be useful. Indeed, prior studies of word prediction systems have shown that long-term use is critical to accurate assessment (Magnuson and Hunnicutt 2002). Furthermore, Fisch word completion supports *gradual* adoption as users familiarize themselves with the consistent positions of words.

Although log files do not enable one to rigorously quantify speed and accuracy, they do allow one to measure the stroke savings gained by using word completion. One can also look at the number of completions undone as an approximation of selection accuracy, and compare this to the number of letters undone (backspaced).

Figure 7.16 shows these quantities graphed over two months of Jim's intermittent use. It represents 897.52 hours of software running-time for 13,288 total strokes. Of these, 8774 were character strokes, 2201 were word-selection strokes, 1451 were backspaces, and 249 were selection undos. In all, 15,629 characters were entered, 6855 of which were from completions. For example, if Jim stroked the letters "t" and 'h" and then selected "there", this would result in 6 entered characters: 2 from character strokes and 4 from the selected completion. The sixth character is a trailing automatic space.

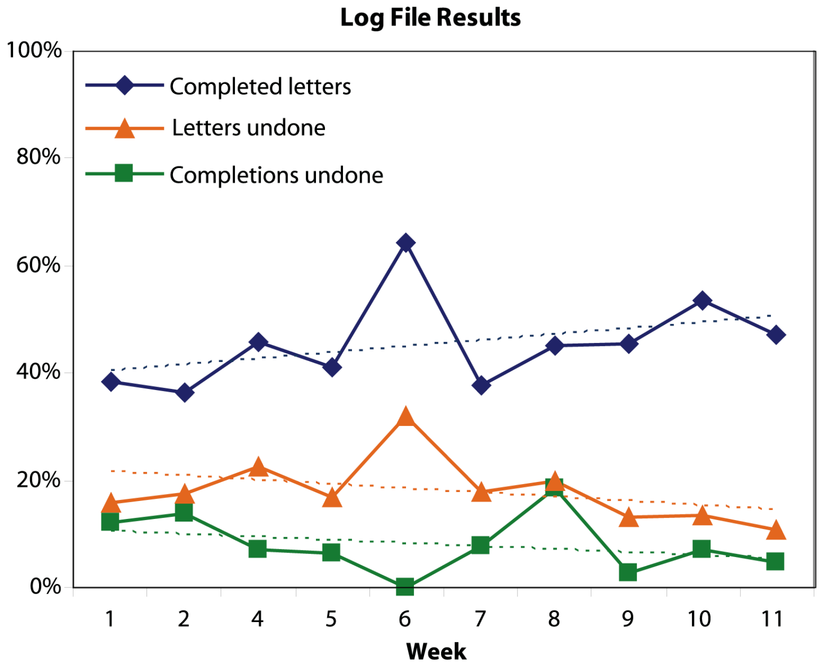


Figure 7.16. Results over 11 weeks from extended use showing usage of word completion and backspace. Week 3 is omitted because Jim did not use his computer.

The top line shows the percent of letters entered as predictions or completions. Without stroke-based word completion, these letters would all have to be entered in full. The weighted mean over all weeks is 43.9%. The spike in week 6 is an outlier due to a week of relative inactivity. Only 70 letters were entered that week, compared to most weeks which saw 1500–3500 letters. A regression line shows this trend to be slightly increasing.

It is interesting that the 46.5% speedup shown in Figure 7.15a is about the same as the 43.9% savings shown in Figure 7.16. That is, the stroke savings more or less translate to speed gains. This suggests that the perceptual, cognitive, and motor costs of Fisch word completion are not overly taxing, which has been a problem with some prior word prediction systems (Goodenough-Trepagnier *et al.* 1986, Soede and Foulds 1986).

The bottom line is the percentage of word completions undone. The weighted mean over all weeks is 7.7%. As an indicator of completion errors, this value is probably high, since users may undo selected completions for reasons other than errors, e.g., as a result of changing what they want to write. A regression line shows this trend to be slightly decreasing.

For comparisons, the percentage of letters undone (backspaced) is shown as the middle line. The weighted mean for undone letters is 16.5%. This value is not surprising in light of previous results indicating that backspace is the second most common keystroke in desktop text entry (MacKenzie and Soukoreff 2002b). A regression line shows this trend to be slightly decreasing.

Across all weeks, the average number of characters entered per completion was 3.11. Thus, with a simple "pulse" into one of four corners, users avoid entering over 3 more characters for every word they write.

7.3.4.6 Discussion

Trackball EdgeWrite greatly benefits in speed and accuracy from having Fisch word completion. Jim's best prior performance with character-level Trackball EdgeWrite was both slower and less accurate than his performance with the new word-level version. This study also demonstrated that Trackball EdgeWrite rivals the major commercial on-screen keyboard WiViK. The study further confirms that Trackball EdgeWrite is just as fast using word prediction and completion, and that it is less visually tedious. Although Trackball EdgeWrite is more error prone during entry, it produces error-free text in the same amount of time due to efficient error correction. Over the course of both studies, Jim went from a peak on-screen keyboard speed of 6.90 WPM without word prediction to 12.09 WPM with word-level Trackball EdgeWrite. His success with Trackball EdgeWrite even resulted in a television news story in which Jim was interviewed (Ivanhoe Broadcast News 2005). As it was for Jim, Trackball EdgeWrite could be useful to other motor-impaired users who wish to "write" with their trackballs.

Chapter 8

EdgeWrite for Mobile Phones*

8.1 Motivation

Mobile phones are enormously popular. At the close of 2003, there were 400 million mobile phone users in Europe, 150 million in the United States, and 270 million in China with an additional 5 million new users each month (GSM World 2004). Phones are becoming increasing powerful as well, capable of displaying full-color graphics, animations, and sound. These trends point toward a future in which mobile phones are unified handheld video game devices, streaming media players, pocket jukeboxes, digital cameras, portable internet terminals, and of course, person-to-person communicators.

Along with this trend of more powerful phones has come a second trend in which the phones themselves are shrinking in size. In February 2004, NEC Corporation released the first "credit card phone," measuring only 85×54×8.6 mm and weighing 70 grams (Figure 8.1a). A few months later, another tiny phone was released, the NTT DoCoMo Premini (Figure 8.1b). It measures just 90×39×19 mm and weighs only 69 grams. Its successor, the Premini-II (Figure 8.1c), measures 105×46×19.4 mm, but unlike its predecessor, it features a camera, a 320×240 pixel QVGA screen, and a music player. This trend of cramming higher functionality into smaller devices shows no signs of stopping.

* Parts of this chapter are adapted from (Chau *et al.* 2006).

Figure 8.1. Mobile phones are continually shrinking in size while increasing in computing power and capabilities.

Unfortunately, despite these advances in compactness and capability—or perhaps in part because of them—the user interfaces on most mobile phones are still unable to fully capitalize on these new technologies (Hirotaka 2003). Currently, most phones use the 12-key keypad based on a design dating to the 1960s. The 12-key keypad is unable to support continuous, fluid control and consumes precious device real-estate.

To take advantage of the continuing trend in phone compactness and capabilities, more expressive input mechanisms are necessary. Video games will require continuous fluid control. Long lists of MP3 songs may require rapid scrolling driven by something other than repeatedly pressing the same key. Basic photo cropping or panning may need a device that supports continuous positioning. For many of these tasks, an isometric joystick would be a valuable asset. Some mobile phones today *do* have joysticks, but they are really just four-way switches that do not allow for fine control. An isometric joystick, on the other hand, supports continuous fluid control without requiring the user's thumb to actually move because the joystick is controlled by pressure.

Another benefit of using an isometric joystick is that a user can *feel* his or her pressure on the stick. Unlike phone keypads, which require the thumb to move from one key to another over keys that all feel the same, an isometric joystick may better support eyes-free use. In fact, prior studies show that the sleek unraised phone keypads that are popular today (Figure 8.2) are very difficult to use without continual visual attention (Silfverberg 2003).

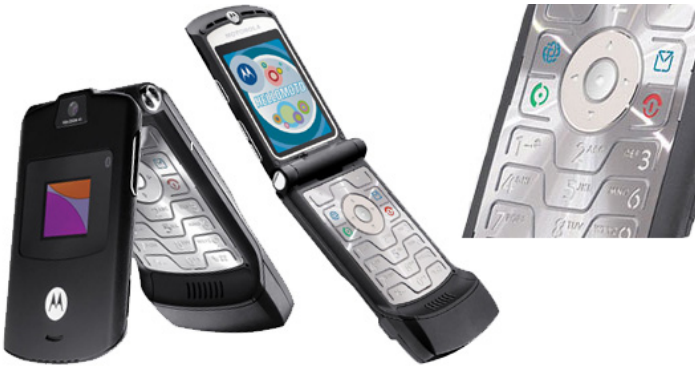


Figure 8.2. Although visually aesthetic, the popular "flat" mobile phone keypads often lack sufficient tactility to be used by feel.

Text entry may be possible using an isometric joystick to support *gestures* instead of methods that require repeatedly pressing keys. Furthermore, isometric joysticks share much in common with trackballs, since neither device has any notion of *position*. Instead, these devices report changes in rotation or pressure and the direction those changes are in (Card *et al.* 1990). These commonalities make it possible for an isometric joystick version of EdgeWrite to use the same software as Trackball EdgeWrite (§7) with adjusted parameters.

Accordingly, this chapter presents the design, implementation, and evaluation of Isometric Joystick EdgeWrite on a mobile phone. The phone prototype contains two joysticks, one on the front for use with the thumb, and one on the back for use with the index finger. The first of two evaluations compares character-level EdgeWrite to *Multitap* in a 30-session longitudinal study. The second evaluation compares word-level EdgeWrite to *T9* (http://www.tegic.com) in a 6-session study with the same users. Results confirm that in both cases, Isometric Joystick EdgeWrite is a competitive alternative to keypad-based text entry. Furthermore, EdgeWrite on an isometric joystick consumes less physical space than a 12-key keypad while offering opportunities for fluid control, something emerging phone manufacturers may find very appealing.

In addition to the two primary studies, this chapter also contains results for using Isometric Joystick EdgeWrite and Multitap without visual attention and for using the isometric joystick on the back of the phone device. Of particular interest with the back joystick is to discover the orientation of letters that subjects prefer.

8.2 Design

To the best of our knowledge, this prototype is the first to integrate isometric joysticks into a mobile phone for text entry. The hardware[13] was built out of a real mobile phone, the Red•E SC1100, which came with the Microsoft Windows Mobile 2003 Smartphone Developer Kit. IBM donated some of their TrackPoint isometric joysticks, which are commonly found in their ThinkPad laptop computers.

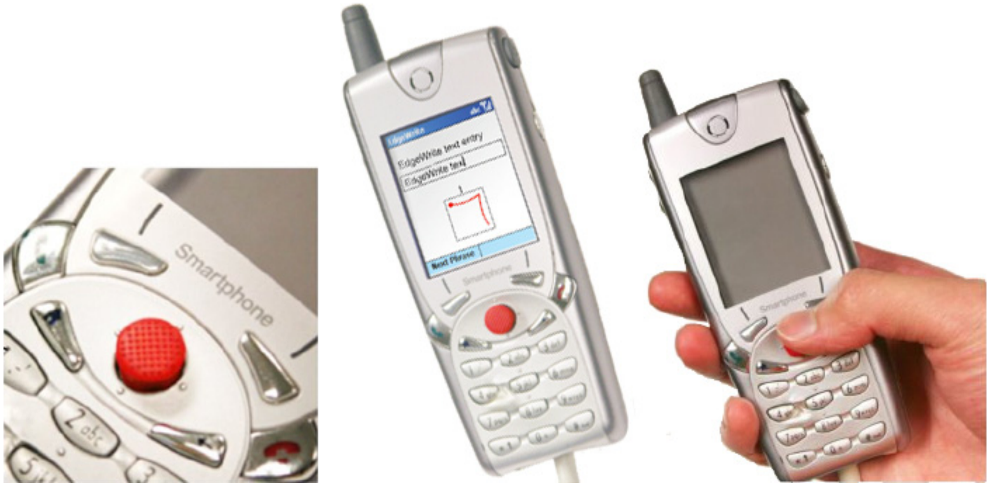


Figure 8.3. The front isometric joystick used by the thumb.

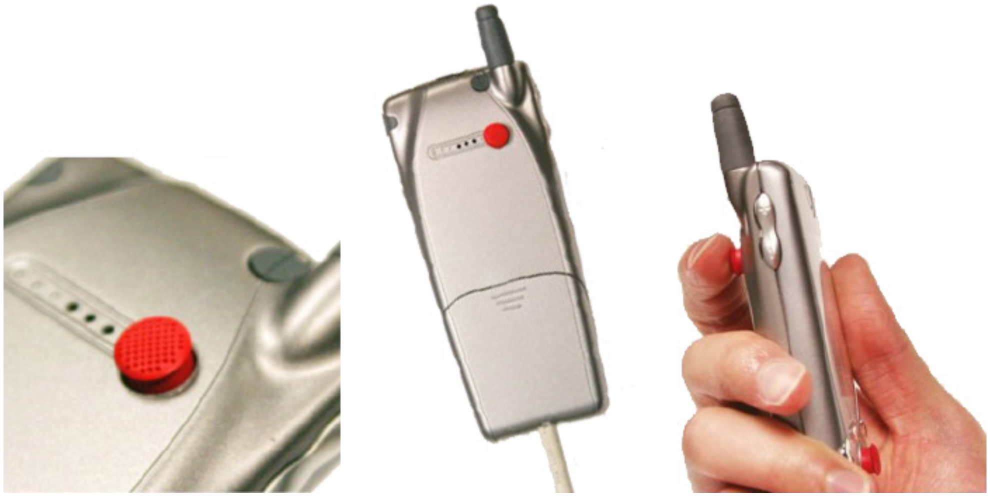


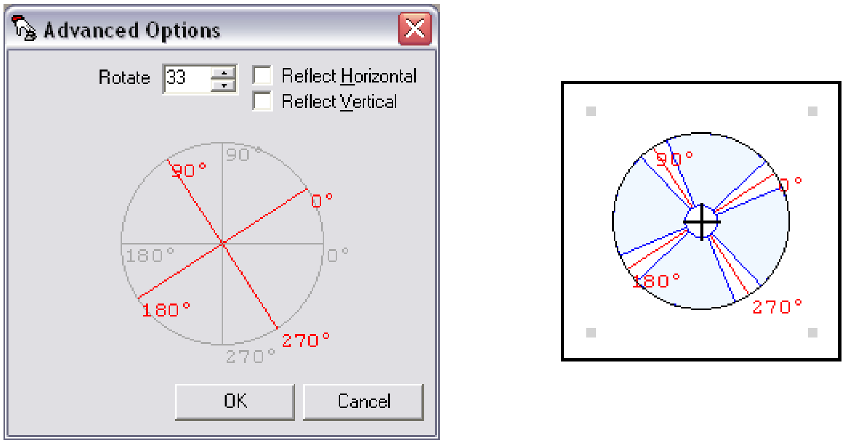
Figure 8.4. The back isometric joystick used by the index finger.

The wiring of the joystick circuits required careful planning since the original hardware was to be kept intact. The prototype preserves almost all of the phone's original

[13] Duen Horng Chau built the hardware prototype. I customized the Trackball EdgeWrite software to accompany it. Brandon Rothrock wrote the phone software for enabling the user study.

components. In particular, the LCD screen continues to work so that user studies could be run using the phone's actual screen. The keypad circuit was removed from the phone to make space for the installation of the front joystick (Figure 8.3). Unfortunately, this disables the keys. But the keys themselves were kept to maintain the familiar appearance of the phone. Integrating the back joystick was more straightforward, and only involved removing an internal speaker and drilling a hole in the casing (Figure 8.4).

Software modifications were required for the back joystick to work with EdgeWrite. In particular, the joystick's left and right directions were flipped, so an option was added to reflect the joystick's input over the vertical. Also, an index finger on the back of the device, when trying to push directly "up," usually pushed at an angle somewhere between 10 and 11 o'clock, instead of "true up" at 12 o'clock. Thus, support was added for arbitrary rotations of the EdgeWrite input plane. The result was a completely configurable input area to accommodate all types of hands (Figure 8.5). Most subjects preferred between 25°-35° of rotation, and discovered this value through trial and error until it "felt right."



Advanced Options

Rotate 33

□ Reflect Horizontal

□ Reflect Vertical

Figure 8.5. The control panel for rotating the EdgeWrite square and the rotated axes shown in the EdgeWrite writing interface.

Signals describing the joysticks' movements were transmitted to a desktop computer through a PS/2 connection via a cable at the bottom of the phone. The desktop computer ran the Trackball EdgeWrite software, translating the joystick input into character strokes, recognizing them, and then sending the recognized characters back to the phone. A special program that ran on the phone, which was also the user test software, received and displayed those characters on the phone's screen (Figure 8.6).

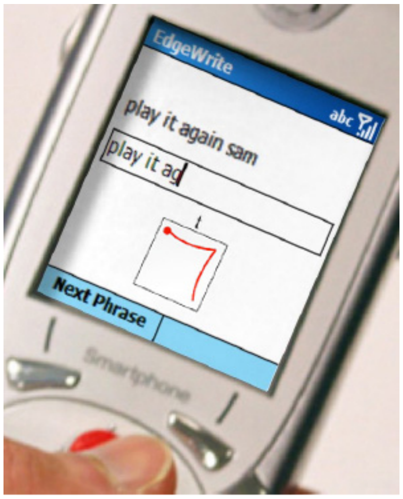


Figure 8.6. Special software on the phone "caught" the recognized characters from the desktop computer and displayed them on the phone's screen. It also drew stroke traces for Isometric Joystick EdgeWrite.

Isometric Joystick EdgeWrite has a number of additional features that makes writing with an isometric joystick easier. In particular, the sensitivity of the joystick and the size of the virtual writing space can be changed to allow strokes to be created more comfortably and accurately. Also, parameters of the underlying goal crossing technique (§7.2.2) can be adjusted for greater accuracy and control, such as the length of the crossing radius, the size of diagonal angles, and the segmentation timeout. These parameters are visible in the top-right image of Figure 7.8.

8.3 Evaluation

Isometric Joystick EdgeWrite was evaluated in two studies. In the first study, the character-level version was compared to *Multitap* over 30 sessions. As a sidebar to this study, subjects also performed a single session with both methods holding the phone beneath the edge of a table and out of sight. Subjects also entered additional phrases during sessions 21–30 using the back isometric joystick.

In the second main study, the word-level version of Isometric Joystick EdgeWrite was compared to T9 over 6 additional sessions using the same subjects.

In both studies, Isometric Joystick EdgeWrite used the EdgeWrite alphabet in place after the guessability study (§3.3.3). Also, continuous recognition feedback (§3.4.1), non-recognition retry (§3.4.2), and slip detection (§3.4.3) were all implemented.

8.3.1 Character-level EdgeWrite *vs*. Multitap

The first experiment compared the character-level version of Isometric Joystick EdgeWrite to Multitap. Multitap was chosen as the competitor because of its predominance on mobile phones today. In one survey, it was the most common choice for text entry among mobile device users (Karlson *et al.* 2006).

8.3.1.1 Subjects

Four male subjects in their early twenties took part in the study. Subjects were screened for not having been regular Multitap or T9 users. No subjects had used any version of EdgeWrite. They were paid $5 per session, with bonuses of $90 after session 10 and $40 after sessions 20 and 30.

8.3.1.2 Apparatus

Two separate phones were used in the study. One phone was the Red•E SC1100 Smartphone with two IBM TrackPoint isometric joysticks embedded in it (Figure 8.3, Figure 8.4). The other was an i-Mate Smartphone 2 (Figure 8.7), used without physical modification in the Multitap and T9 conditions. Both phones ran specialized software written in C# that allowed them to display test phrases and, in the case of EdgeWrite, draw character strokes. Stroke processing was carried out on a connected desktop PC, which was running the Trackball EdgeWrite software adjusted for the isometric joystick. The desktop also ran the TextTest user test program (Figure 2.14) and sent the test phrases and processed characters to the phone for display.

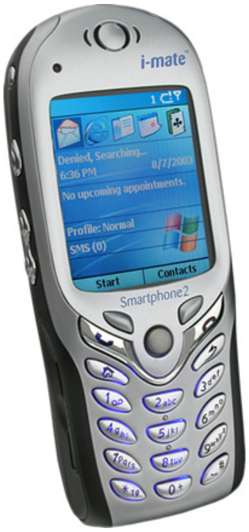


Figure 8.7. The i-Mate Smartphone 2 model used in our study in the Multitap condition. In the second study, it would be used in the T9 condition.

8.3.1.3 Trials

A single trial was one text phrase drawn randomly from a modified version of the MacKenzie and Soukoreff corpus of 500 phrases (MacKenzie and Soukoreff 2003). The phrases were modified to be shorter so that they could be displayed without wrapping on the Smartphone screen (Figure 8.6). Only letters and spaces were tested, since the mechanisms for entering more obscure characters were not substantially different from the mechanisms for entering letters. Capital letters were not included.

8.3.1.4 Procedure

The experiment was a two-factor within-subjects design with factors for *Method* (EdgeWrite, Multitap) and *Session* (1–30). Four subjects used both methods in each session entering 8 test phrases per method for 30 sessions. Thus, there were 4×2×8×30 = 1920 phrases in all, or about 48,000 characters. Method order was counterbalanced to prevent order effects by having each subject alternate his or her starting method in each session. Dependent measures for speeds and error rates are the means for the 8 trials performed by each subject with each method in each session.

At the start of the first session, subjects were introduced to both methods and shown how to perform them. They were also shown an EdgeWrite character chart and told how to read it. However, they did not have the chart available to them when entering the test phrases. Following this introduction, they entered each letter three times (“aaa”, “bbb”, …, “yyy”, “zzz”) with each method. They also entered “the quick brown fox jumps over the lazy dog” while correcting any mistakes as they went. This introduction and practice took about 30 minutes, and only occurred before session 1.

In sessions 2–30, subjects entered 2 warm-up phrases with each method, during which time they could view a character chart or ask questions if they desired. Then they entered 8 test phrases. As stated, during the 8 test phrases, no EdgeWrite character chart was made available to them. The 10 phrases took 5–10 minutes to enter.

Sessions were spaced by no less than 2 hours and no more than 2 days, with the exception of session 21, which took place about one week after session 20. No more than 2 sessions could occur on a single day for a given subject. These restrictions were similar to those used by other researchers in longitudinal text entry studies (MacKenzie and Zhang 1999).

8.3.1.5 Analysis

Data was analyzed using a mixed model analysis of variance with fixed effects for *Method* and *Session* and a random effect for *Subject* (Littell *et al.* 1996). *Session* was modeled as a continuous variable akin to time. *Method Order* was also included in the model to test for order effects. A main effects test for *Method Order* on speed is not significant ($F_{1,229}$=0.01, p=.91), and neither is a *Method Order*×*Method* interaction ($F_{1,229}$=1.42, p=.24). That both of these tests are non-significant indicates adequate counterbalancing.

8.3.1.6 Speed

Over all 30 sessions, the grand mean speed for Isometric Joystick EdgeWrite was 9.39 WPM (σ=2.34) and for Multitap was 9.64 WPM (1.65). The maximum session average for EdgeWrite was session 29 at 12.32 WPM. For Multitap, it was session 28 at 12.22 WPM.

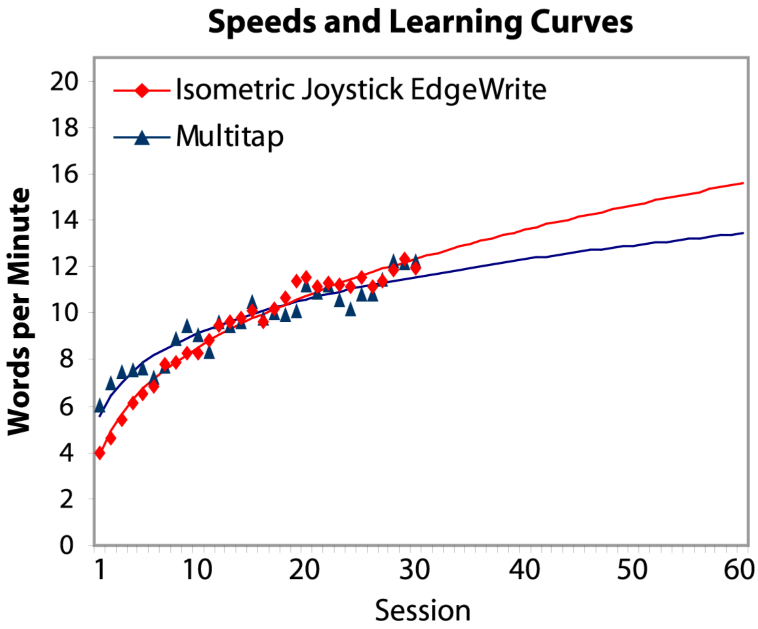


Figure 8.8. Speeds and fitted learning curves over sessions for EdgeWrite and Multitap.

A main effect of *Method* on speed is not significant ($F_{1,229}$=1.75, p=.19). This is because Multitap was faster than Isometric Joystick EdgeWrite in the early sessions, but slower toward the end. A main effect of *Session* on speed is significant ($F_{1,229}$=385.04, p<.0001), confirming that subjects did speed up over time. A *Session*×*Method* interaction is also significant ($F_{1,229}$=11.05, p<.005), indicating that subjects improved at different rates with each method. When we look at the learning curves for the two methods, we can see this difference (Figure 8.8).

The equations and R^2 fits for the learning curves in Figure 8.8 are:

• $y = 3.913x^{0.3375}$	$R^2 = 0.98$	Isometric Joystick EdgeWrite
• $y = 5.530x^{0.2169}$	$R^2 = 0.91$	Multitap

The EdgeWrite model shows a particularly high correlation, suggesting that it is well-modeled by the power law of learning (Card *et al.* 1983). The fitted learning curves themselves cross over in session 18 (~2.15 hours), and from that session through session 30, EdgeWrite is faster on average.

8.3.1.7 Uncorrected Errors

Over all 30 sessions, the grand mean uncorrected errors for EdgeWrite was 1.01% (σ=0.63) and for Multitap was 0.52% (0.38). The minimum (best) session average for EdgeWrite was session 7 at 0.28%. For Multitap, it was sessions 3, 7, and 29 at 0.00%.

Although both methods had low uncorrected error rates in general, Multitap had fewer uncorrected errors than EdgeWrite as indicated by a significant main effect of *Method* ($F_{1,229}$=17.44, p<.0001). *Session* did not have an effect on uncorrected errors ($F_{1,229}$=0.02, p=.89), which is typical in longitudinal text entry studies since expertise does not necessarily reduce uncorrected errors. Figure 8.9a shows uncorrected errors over sessions.

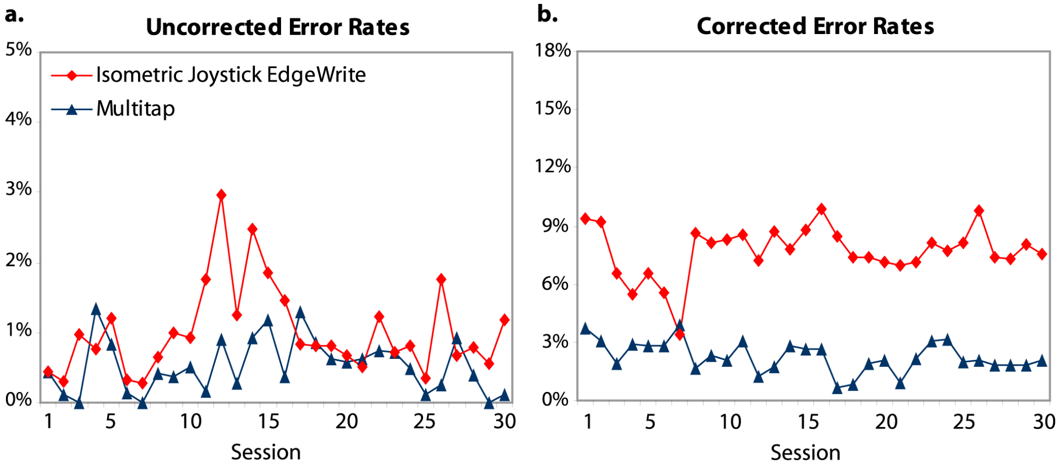


Figure 8.9. Uncorrected and corrected error rates over sessions for character-level EdgeWrite and Multitap. Lower values are better. Note the different ranges on the *y*-axes.

8.3.1.8 Corrected Errors

Over 30 sessions, the grand mean corrected errors for EdgeWrite was 7.69% (σ=1.35) and for Multitap was 2.27% (0.80). The minimum session average for EdgeWrite was session 7 at 3.37%. For Multitap, it was session 17 at 0.66%.

Multitap created fewer errors during entry than EdgeWrite as indicated by a significant main effect of *Method* ($F_{1,229}$=353.53, p<.0001). *Session* did not have an effect on corrected errors ($F_{1,229}$=0.07, p=.79). It is not atypical for a gestural method to be more error prone *during* entry than a selection-based method. Fortunately, both methods left relatively few *un*corrected errors (≤~1%), although Multitap left fewer. Figure 8.9b shows corrected errors over sessions.

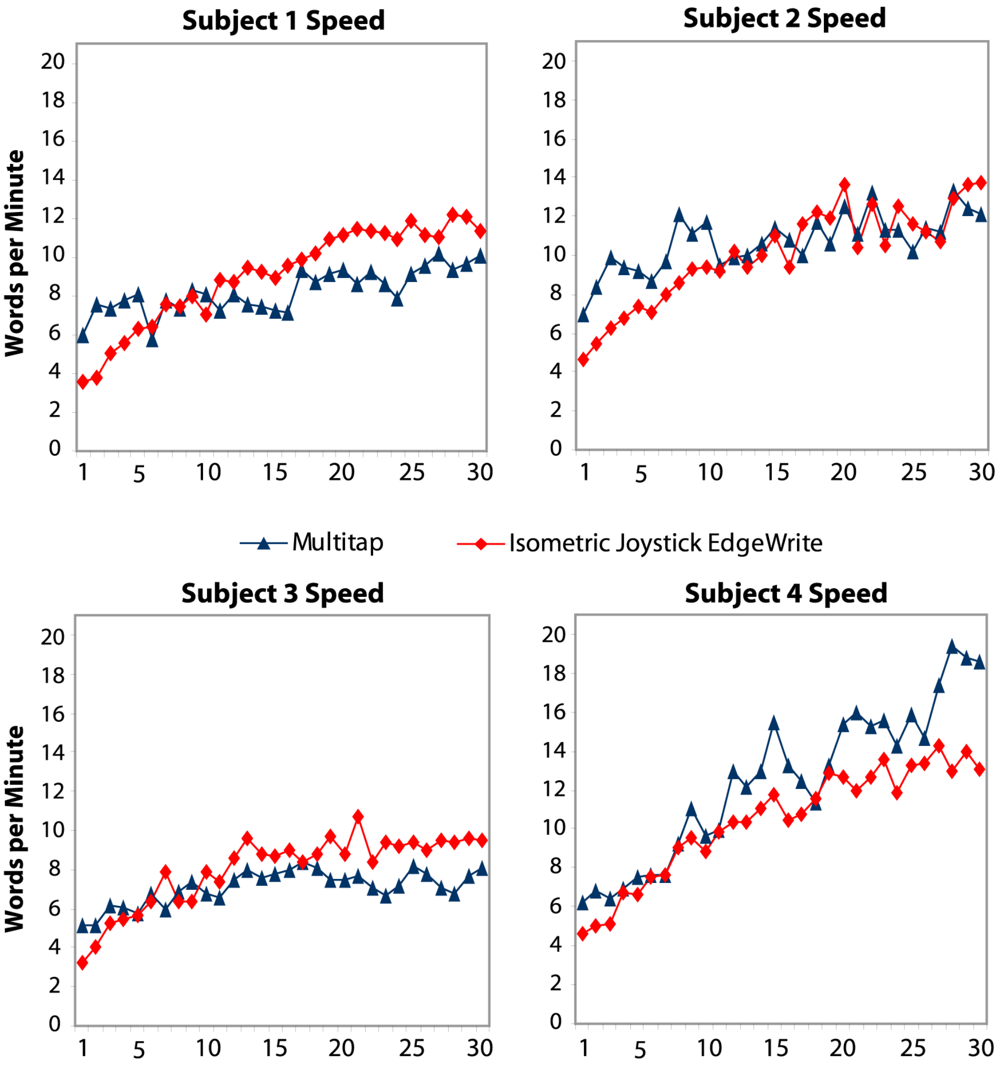


Figure 8.10. The speeds of subjects with character-level EdgeWrite and Multitap. Subjects 1 and 3 were faster with EdgeWrite. Subjects 2 and 4 were faster with Multitap.

8.3.1.9 Individual Differences

With only four subjects, it is useful to look at the performances of individual subjects. Each subject's speed is graphed in Figure 8.10.

Wilcoxon sign-rank tests for matched pairs show that subject 1 was significantly faster with EdgeWrite over sessions (9.07 *vs*. 8.25 WPM, z=112.50, p<.02). Subject 3 also was significantly faster with EdgeWrite (8.00 *vs*. 7.10 wpm, z=160.50, p<.001). On the other hand, subject 2 was significantly faster with Multitap, mainly owing to the early sessions (10.33 *vs*. 10.73 WPM, z=-106.50, p<.03). Subject 4 was the fastest subject with both methods, and was much faster with Multitap (10.44 *vs*. 12.47 WPM, z=-228.50, p<.0001). Taken together, these mixed results are inconclusive and warrant further testing with more subjects to separate these two methods.

8.3.1.10 Qualitative Results

Initial impressions of Isometric Joystick EdgeWrite and Multitap were captured by a questionnaire after subjects completed session 10. Their responses to 5-point Likert scales are shown in Figure 8.11. In summary, subjects felt that Isometric Joystick was slightly easier to use, faster, more enjoyable, more comfortable, and better liked. However, they also felt EdgeWrite was harder to learn and slightly less accurate. A repeated measures analysis of variance of subjects' average Likert scores is not significant, but the trend is in favor of EdgeWrite ($F_{1,3}$=2.29, p=.23).

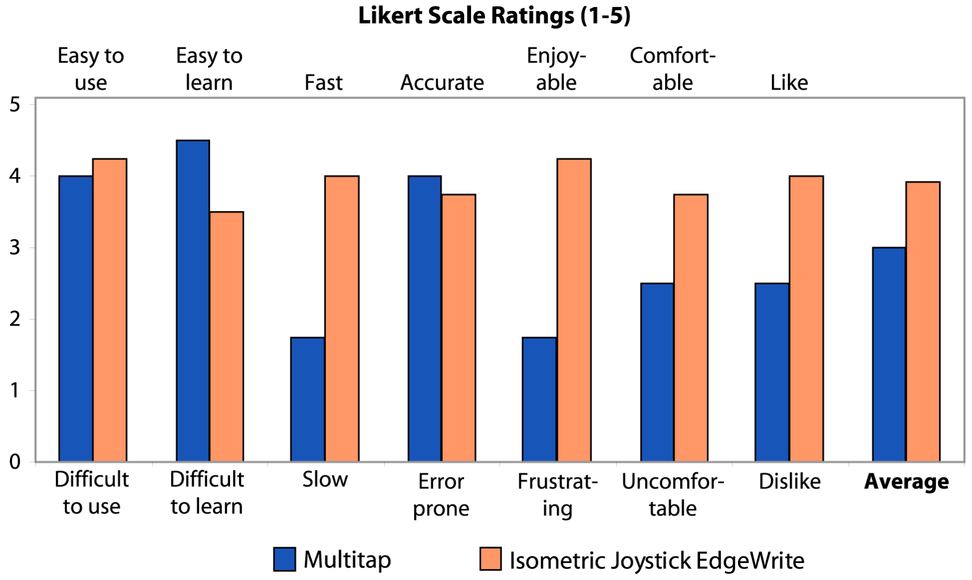


Figure 8.11. Subjective ratings for Isometric Joystick EdgeWrite and Multitap. Ratings are on a Likert scale (1–5). Higher values are better.

On the whole, subjects seemed to prefer Isometric Joystick EdgeWrite. Subjects unanimously indicated that they would choose it over Multitap to write a line of text, a few paragraphs, or even a few pages.

8.3.2 EdgeWrite and Multitap Eyes-Free

The Multitap study (§8.3.1) was conducted in ideal conditions. The subjects were seated in a chair using a mobile phone with their undivided attention devoted to the text entry task. However, this is not a realistic setting for much of mobile text entry. Therefore, we attempted to see how the two methods fared when visual attention was impossible, which is closer to real mobile use (Oulasvirta *et al.* 2005). At the end of the 30th session, subjects were asked to enter another set of phrases when holding the device under the edge of the table such that they could not see it. However, they *could* see the text output on the desktop computer screen. Thus, subjects were blind to their device ("input blind"), but not blind to the text they produced (not "output blind"). In a real-world setting, this situation might correlate to when users have an ear-bud for audio feedback or a heads-up or head-mounted display but are unable to look at their hands or their device.

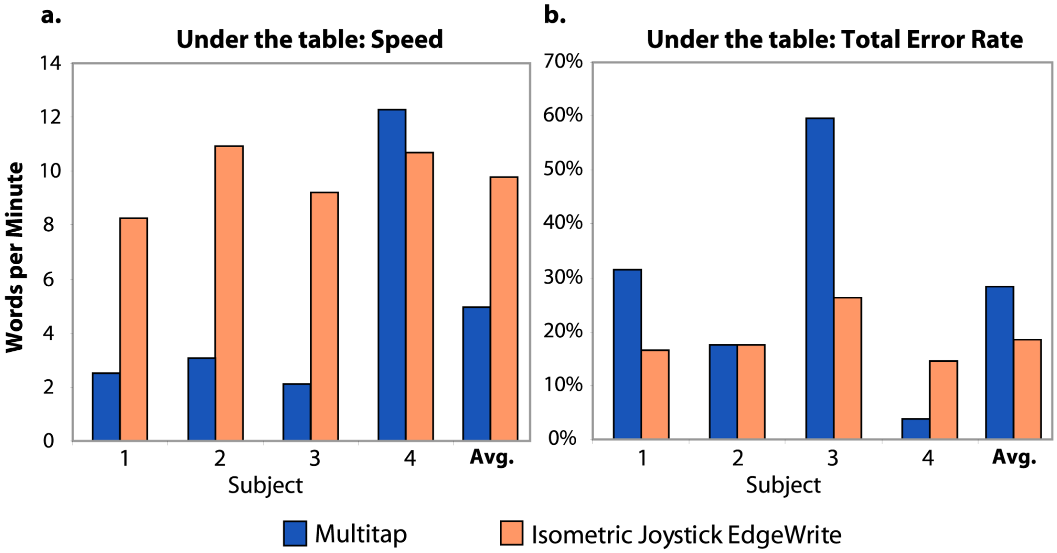


Figure 8.12. Speeds and total error rates for EdgeWrite and Multitap when subjects were unable to see the device. Higher values are better in (a); lower values are better in (b).

All subjects entered 8 phrases with Isometric Joystick EdgeWrite. However, subjects 1–3 became so frustrated with Multitap when unable to see their device that they gave up after completing 1, 5, and 2 phrases, respectively. Only subject 4 completed all 8 phrases with Multitap. Figure 8.12 shows subjects' means for the trials they completed.

EdgeWrite was faster on average than Multitap (9.78 *vs*. 5.00 WPM). However, this difference failed to be significant in a Wilcoxon sign test (*z*=4.00, p=.25) due to lack of power and subject 4's data, which runs opposite to that of subjects 1–3. EdgeWrite was also more accurate on average (18.77% *vs*. 28.16%). However, this difference was also not significant (*z*=-3.00, p=.38). Total errors are shown instead of uncorrected and corrected errors because different subjects adopted different correction strategies. Given the difficulty of correcting errors in Multitap when unable to see the device, some subjects left most of their errors uncorrected, while others spent much of their time trying to correcting errors.

Although these results are not conclusive, they do show that keypad-based methods are difficult to use without concentrated visual attention, a finding consistent with prior studies (Silfverberg 2003). They also show that gestural text entry methods may fare better in these circumstances. Further testing is required to elicit these differences, but these results are promising for Isometric Joystick EdgeWrite.

8.3.3 Isometric Joystick on the Back

Also investigated was the performance of the index finger with an isometric joystick embedded in the back of the device (BoD) (Figure 8.4). This was done in two parts. In the first part, a letter orientation study was run before any of the sessions comparing Multitap and EdgeWrite began. In the second part, subjects entered phrases using the BoD joystick in sessions 21–30. That is, after subjects finished writing with Multitap and EdgeWrite in those sessions, they entered another 10 phrases (2 practice, 8 test) with the BoD isometric joystick. Both parts of the BoD study are described below.

8.3.3.1 Letter Orientation Study

Before subjects participated in their first session—before they had any exposure to EdgeWrite or the isometric joystick phone—they traced English capitals "O", "C", "U", and "L" in four conditions. The conditions were formed from two factors: *Joystick* (front, back) and *Feedback* (visible, not visible). Subjects 1 and 3 first traced the letters using the front joystick, while subjects 2 and 4 first used the back joystick. In all cases, subjects first did the "not visible" condition followed by the "visible" condition to avoid the latter biasing the former. When feedback was visible, subjects could see their trace unfold on the desktop computer screen. Subjects were told to pretend that the desktop screen was the screen of their mobile device.

Figure 8.13 shows how a BoD letter-trace might appear to the user if the user were looking *through* the device. To illustrate the point clearly, Figure 8.13 uses a BoD touchpad and an iPAQ device, even though a BoD isometric joystick and Smartphone were used in the study. (Unlike a finger on an isometric joystick, which never actually moves, a finger on a touchpad can trace a discernable path. However, the orientation issue is equally relevant to a trace performed with an isometric joystick.)

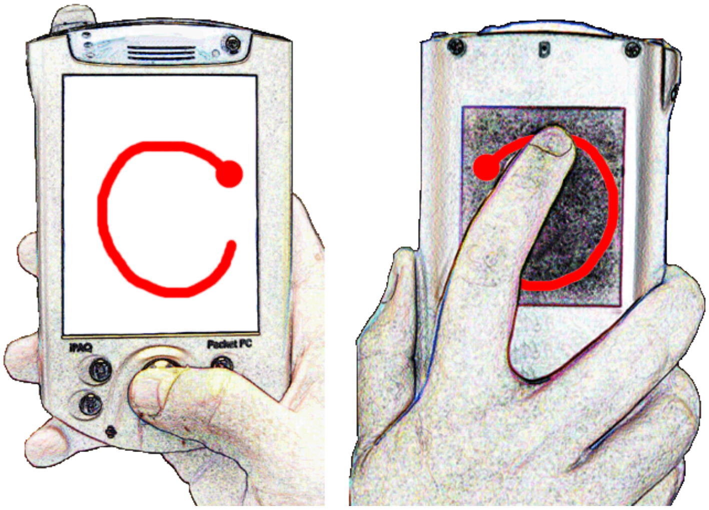


Figure 8.13. The illustration shows how a trace of an English "C" on the back of the device is visually correct if looking *through* the device but motor-reversed from the perspective of the index finger.

The issue of interest is how subjects would orient letters when using the back joystick. On the one hand, subjects may write *visually-oriented*, as if they could "see through" the mobile phone to the trace drawn out by their index finger (Figure 8.13). However, this would mean they are writing in mirror-image with respect to their index finger. Thus, the other option is to write *motor-oriented*, where the index finger moves through the same path it always does when writing. Clearly, software can produce visually-correct output for either approach, but the question is, what do subjects expect? Which is more natural?

The four letters "O", "C", "U", and "L" were chosen for their different symmetries. However, no differences were observed among letters. When using the front joystick, all subjects traced all four letters in the expected fashion, visually and motorically correct. This was true in both feedback conditions. For the back joystick without feedback, 3 of 4 subjects traced letters in a visually-oriented manner. Subject 1, however, traced letters in a motor-oriented manner. When feedback became available, however, all four subjects traced in a visually-oriented manner, indicating that they thought this was the "proper orientation." Note that the trace feedback was like that shown in Figure 8.13, where it's as if subjects can "see through" their device. Thus, it seems that although one subject may have thought differently apart from feedback, once visual feedback became available, all subjects preferred this orientation. This sentiment is consistent with prior results of letter orientation on differently-oriented surfaces of the body (Parsons and Shimojo 1987). In light of these findings, when the BoD isometric joystick was used to enter text in sessions 21–30, it was used corresponding to the visually-oriented condition.

8.3.3.2 Speed

The grand mean speed of subjects using the back isometric joystick was 7.70 WPM (σ=0.96). The maximum session average achieved on the back of the device was 8.87 WPM in the 8th session. Recall that the BoD test was over 10 sessions from sessions 21–30. By comparison, the average front-joystick speed for these sessions was 11.49 WPM (0.42). In addition, over sessions 1–10, the average front-joystick speed was 6.56 WPM (1.53). These data are shown in Figure 8.14.

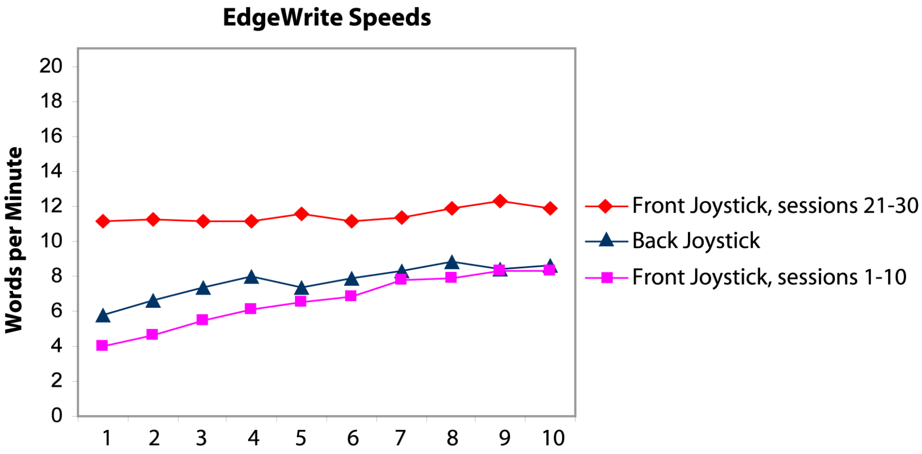


Figure 8.14. Average speeds for the front and back EdgeWrite isometric joysticks.

8.3.3.3 Uncorrected Errors

The grand mean uncorrected error rate for the back joystick was 2.86% (σ=1.46). The minimum (best) session average achieved on the back of the device was 0.57% in the 9th session. For the front joystick over sessions 21–30, the average was 0.86% (0.42). For the front joystick over sessions 1–10, the average was 0.68% (0.33). These data are depicted in Figure 8.15a.

8.3.3.4 Corrected Errors

Not surprisingly, corrected errors for the back joystick were noticeably worse than for the front joystick. The grand mean corrected error rate for the back joystick was 12.62% (σ=1.56). The minimum (best) session average achieved on the back of the device was 10.51% in the 8th session. For the front joystick over sessions 21–30, the average was 7.83% (0.81). For the front joystick over sessions 1–10, the average was 7.11% (1.95). These data are graphed in Figure 8.15b.

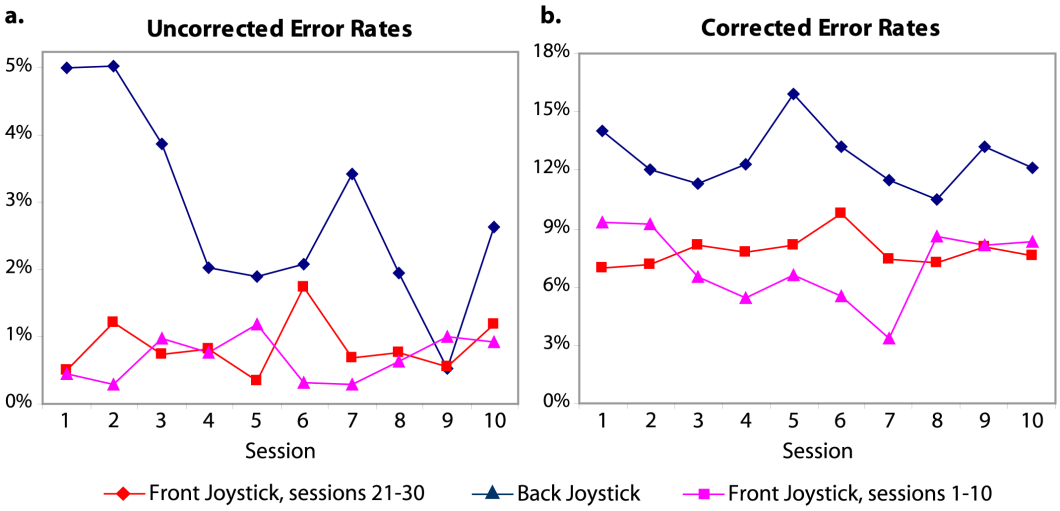


Figure 8.15. Average uncorrected and corrected error rates for the front and back isometric joysticks. Lower values are better. Note the different ranges on the *y*-axes.

The statistical tests for these data are omitted because it is unclear that a fair comparison can be made. On the one hand, subjects had 20 prior sessions of EdgeWrite practice with the front joystick before performing with the back joystick. Comparisons to data from sessions 1–10 are therefore not particularly meaningful. On the other hand, subjects were just writing with a new joystick for the first time, so comparisons to data from sessions 21–30 are speculative. It is clear from the graphs, however, that using a joystick with an index finger on the back of a device is entirely feasible, being comparable in speed but more prone to error than the front joystick. To the best of my knowledge, this work is the first explicit BoD text entry study.

8.3.4 Word-level EdgeWrite *vs.* T9

The second main experiment was conducted about a month after the first (§8.3.1) with the same subjects, this time comparing the *word-level* version of Isometric Joystick EdgeWrite to T9. This word-level version is the Fisch design for Trackball EdgeWrite

(§7.2.5). T9 was chosen as the competitor because it is one of the most popular predictive disambiguating phone keypad-based methods. About 10.5% of surveyed phone users enter text with T9 (Karlson *et al.* 2006).

8.3.4.1 Procedure

The experiment was a two-factor within-subjects design with factors for *Method* (EdgeWrite, T9) and *Session* (1–6). Four subjects used both methods in each session entering 8 test phrases per method for 6 sessions. Thus, there were 4×2×8×6 = 384 test phrases in all, or ~9600 characters. Test phrases were pulled randomly from the same published corpus as before (MacKenzie and Soukoreff 2003). Dependent measures for speed and error rates are the means for the 8 phrases entered by each subject with each method in each session.

As before, subjects entered 10 total phrases (2 warm-up, 8 test) with each method in each session, or 20 phrases per session. Before the first session, subjects were introduced to word-level EdgeWrite and T9. This introduction took about 15 minutes, and only occurred before the first session. Sessions were spaced by no less than 2 hours and by no more than 2 days. No more than 2 sessions could occur on a single day for a given subject.

8.3.4.2 Analysis

As before, the data were analyzed using a mixed model analysis of variance with fixed effects for *Method* and *Session* and a random effect for *Subject*. *Method Order* was included in the model to test for order effects. A main effects test for *Method Order* on speed is not significant ($F_{1,37}$=0.04, p=.83), and neither is a *Method Order*×*Method* interaction ($F_{1,22}$=0.00, p=.97), indicating adequate counterbalancing.

8.3.4.3 Speed

Over all 6 sessions, the grand mean speed for word-level EdgeWrite was 12.81 WPM (σ=1.37) and for T9 was 15.20 WPM (0.78). These are 36.4% and 57.7% faster than the grand means for character-level EdgeWrite (9.39 WPM) and Multitap (9.64 WPM), respectively. The maximum session average for word-level EdgeWrite was session 5 at 14.27 WPM. For T9, it was session 6 at 16.10 WPM. These speeds fall between those from a prior study of T9 for novices (9.09 WPM) and experts (20.36 WPM) (James and Reischel 2001), suggesting that subjects' T9 proficiency was good but could improve.

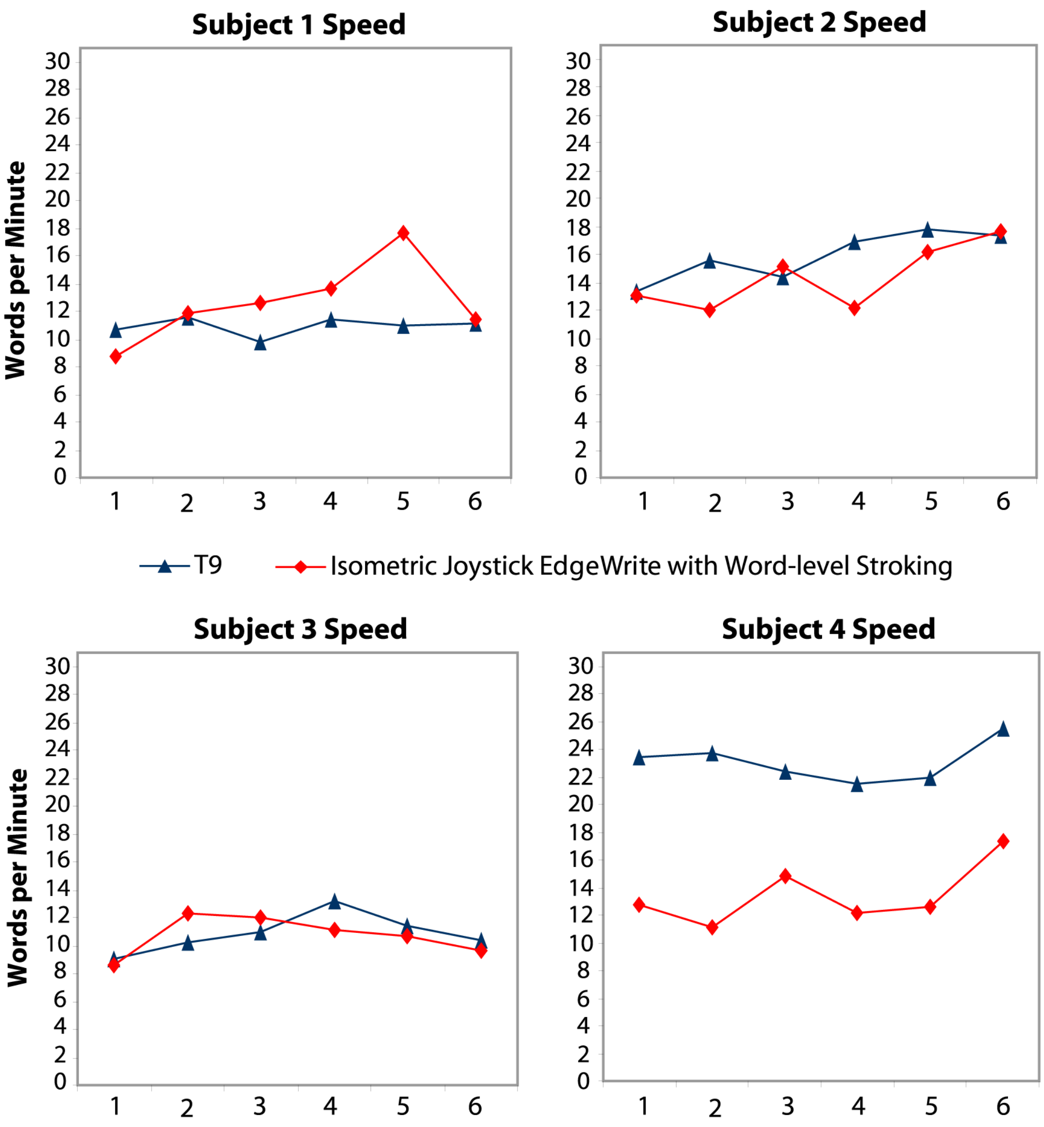


Figure 8.16. The speeds of subjects with word-level EdgeWrite and T9.

Figure 8.16 shows each subject plotted separately because of the large differences among subjects—particularly subject 4—which would be lost in a combined plot. Subject 4 was a clear outlier in T9 performance. Mean speeds without his data are 12.58 WPM (σ=1.53) for EdgeWrite and 12.56 WPM (1.05) for T9.

T9 was faster than EdgeWrite as indicated by a significant main effect of *Method* ($F_{1,37}$=8.21, p<.01). However, with only six sessions, insufficient learning took place for a significant main effect of *Session* ($F_{1,37}$=3.95, p=.06). A *Session*×*Method* interaction is also not significant ($F_{1,37}$=0.33, p=.57).

Wilcoxon sign-rank tests for matched pairs show no significant differences in EdgeWrite *vs*. T9 speed for subject 1 (12.67 *vs.* 10.91 WPM, *z*=7.5, p=.16), subject 2 (14.36 *vs.* 15.90 WPM, *z*=-6.5, p=.22), and subject 3 (10.71 *vs.* 10.87 WPM, *z*=-1.5, p=.84). However, subject 4's T9 speed is significantly faster (13.49 *vs.* 23.11 WPM, *z*=-10.5, p<.05). Thus, the significant effect of *Method* on speed can be largely attributed to subject 4's performance. Recall that subject 4 was also much faster than subjects 1–3 with Multitap and EdgeWrite in the previous study (Figure 8.10).

8.3.4.4 Uncorrected Errors

Over all 6 sessions, the grand mean uncorrected errors for EdgeWrite was 0.54% (σ=0.32) and for T9 was 0.21% (0.30). These values are 46.5% and 59.6% lower than those for character-level EdgeWrite (1.01%) and Multitap (0.52%), respectively. This is pleasing, since the uncorrected errors for character-level EdgeWrite were low to begin with. The minimum (best) session average for EdgeWrite was session 5 at 0.23%. For T9, it was sessions 3 and 6 at 0.00% (Figure 8.17a).

A main effect of *Method* on uncorrected errors is only marginally significant in T9's favor ($F_{1,37}$=4.07, p=.051). *Session* did not have an effect on uncorrected errors ($F_{1,37}$=0.52, p=.47).

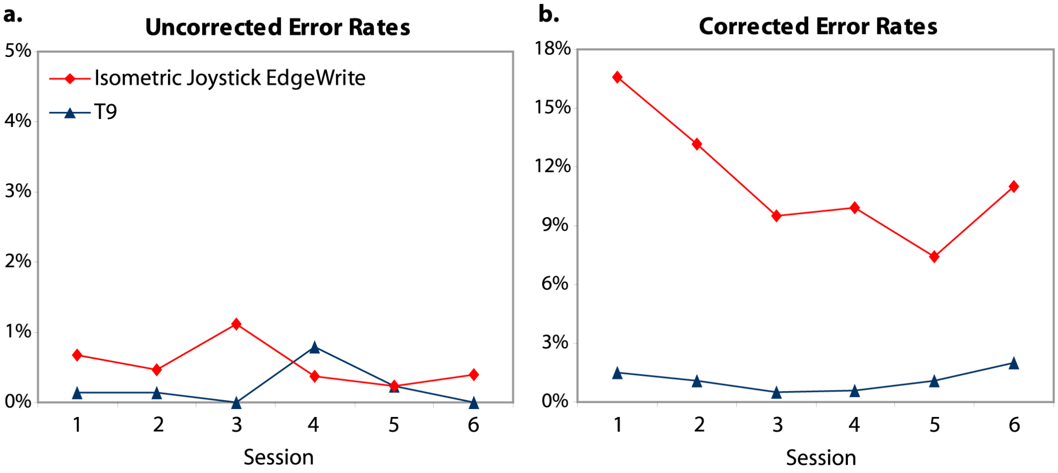


Figure 8.17. Uncorrected and corrected error rates over sessions for word-level EdgeWrite and T9. Lower values are better. Note the different ranges on the *y*-axes.

8.3.4.5 Corrected Errors

Over 6 sessions, the grand mean corrected errors for EdgeWrite was 11.26% (σ=3.21) and for T9 was 1.12% (0.58). This is the first metric that is *worse* for word-level

EdgeWrite compared to character-level EdgeWrite (7.69%)—worse by 46.4%. In contrast, T9 was 50.7% better than Multitap (2.27%). The minimum session average for EdgeWrite was session 5 at 7.42%. For T9, it was session 3 at 0.46% (Figure 8.17b).

T9 created fewer errors during entry than EdgeWrite as indicated by a significant main effect of *Method* on corrected errors ($F_{1,37}$=115.55, p<.0001). *Session* also had a significant effect ($F_{1,37}$=4.70, p<.05), as subjects reduced their errors over sessions. A significant *Session*×*Method* interaction ($F_{1,37}$=5.94, p<.02) indicates that sessions affected each method differently. Figure 8.17b shows that it was EdgeWrite's corrected errors that improved dramatically over sessions.

The fact that corrected errors were high for word-level EdgeWrite is at least in part due to the way corrected errors are counted (§2.4.3). With Fisch word-level stroking, multiple letters can be entered in a single action. Similarly, multiple letters can be erased in a single action. However, to remain consistent with other studies for the sake of comparisons, every erased letter must be counted as a *separate* corrected error. The efficiency of error correction is therefore not accounted for by this measure. This is a limitation in the aggregate corrected errors metric (Soukoreff and MacKenzie 2003) that gives it reduced applicability to word-level entry methods. See §11.1.1 for a related point about measuring character-level errors.

T9, on the other hand, did not "commit" letters into the text entry field until a subject had finished entering his or her key sequence and possibly cycled through an *n*-best list. When a subject finally pressed SPACE, his or her current word was committed all at once. This means that subjects could preview their word before it was entered, and hence, drastically reduce corrected errors. Of course, T9 would be impossible to use "under the table" for this reason.

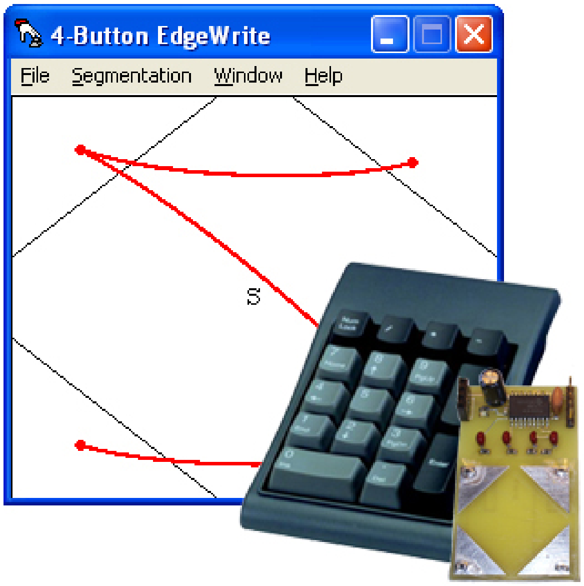
8.3.4.6 Discussion

As with the Multitap study (§8.3.1), the results of the T9 study are inconclusive and more testing is required. However, it is fair to say that word-level EdgeWrite is at least "competitive" with T9. Subject 1 was better on average *in every session* with EdgeWrite than with T9. Subjects 2 and 3 were mixed, and subject 4 was a clear outlier in performance with both Multitap and T9. A post-test questionnaire revealed that subject 4 was a trained piano and guitar player, criteria for which we did not screen. This subject's musical training may well have influenced his finger dexterity and caused him to be somewhat unrepresentative of the "typical" mobile phone user.

Technical improvements may also help distinguish EdgeWrite from Multitap and T9. There was a minor amount of lag occasionally perceptible in the experiment due to the communication between the desktop computer and the mobile phone. This lag was not present for Multitap or T9, but did appear for EdgeWrite due to the larger amounts of data being transmitted to draw strokes on the phone screen (Figure 8.6).

Another improvement is to fix the way word selections are undone in EdgeWrite. In the second study, a user could make a word-level backspace to undo a selected word completion *as long as they performed the backspace immediately after word selection*. If any other text entry events happened before the selection was undone, the undo state was lost and a word-level backspace would erase the whole previous word, not just the completed letters. This is a technical limitation that will be remedied for future studies, but the logs show it hurt the performance of word-level EdgeWrite against T9. Unfortunately, it is impossible to estimate by how much.

Still, it is very encouraging that different versions of EdgeWrite can rival two major mobile phone text entry methods in a fraction of the physical space required by a keypad after just over 2 hours of practice. Also, the results from "under the table" suggest that EdgeWrite is a much better option when visual attention is limited or impossible. This may make Isometric Joystick EdgeWrite better for users who are walking or riding. Finally, it is valuable to see that in speed and uncorrected errors, word-level EdgeWrite improved character-level EdgeWrite by 36.4% and 46.5%, respectively. These values are consistent with Jim's respective improvements of 46.5% and 36.7% from the word-level Trackball study (§7.3.4).



Chapter 9

EdgeWrite on Four Keys or Sensors[*]

This chapter presents two new forms of EdgeWrite, versions that do not use digitizing surfaces or virtual "surfaces" with the mouse cursor. Instead, these versions use four discrete "binary" sensors that are directly mapped to the four corners of the EdgeWrite square. One prototype is implemented using a standard numeric keypad. The other uses four capacitive sensors in an integrated hardware sensing unit. The majority of the chapter is devoted to the 4-key prototype and the longitudinal study that evaluates it.

9.1 Motivation

Text entry with only a few keys has been studied in mobile computing for some time (Bellman and MacKenzie 1998, MacKenzie 2002c, MacKenzie 2002b). As mobile technologies shrink while becoming more powerful and network-aware, few-key methods remain relevant for devices such as wrist-watch PDAs, GPS units, and 2-way pagers. Areas besides mobile computing may also benefit from few-key methods. For example, fabric keypads have been sewn into smart clothing (Orth *et al.* 1998), and few-key methods have been designed for people with limited ranges of motion (Evreinova *et al.* 2004).

[*] Parts of this chapter are adapted from (Wobbrock *et al.* 2006).

Prior few-key methods have mostly been based on *selection*, displaying letters on a screen and having the user pick repeatedly from among them. Although selection-based methods are easy to learn, they have serious drawbacks for mobile text entry similar to the drawbacks of on-screen keyboards for desktop text entry (§2.3.6): (1) they require a screen; (2) the selections themselves consume precious screen real-estate; (3) they require a user's visual attention and cannot be performed by feel; (4) they involve two foci-of-attention, the text being written and the selectable letters; and (5) they can be quite slow and tedious.

Gestural methods, on the other hand, depend not on an on-screen depiction of selections but on the execution of meaningful motor patterns. Therefore, they generally do not incur the aforementioned drawbacks. In the few cases where gestures have been applied to keys (Isokoski and Raisamo 2000, Jannotti 2002, Evreinova *et al.* 2004), however, they have been somewhat arbitrary and difficult to learn. However, the advantages of gestures for few-key text entry warrant the investigation of a quickly learnable gestural technique. Therefore, EdgeWrite was implemented for use on four keys.

9.2 Design

As discussed in §3.2, EdgeWrite gestures are fully defined by the sequence of corner-hits they make inside an EdgeWrite square. In the four-key version, then, each key-press represents a corner-hit (Figure 9.1). Thus, unlike MDITIM or UDLR (§2.3.8), key-presses do not represent strokes, but *endpoints* of strokes, allowing for more Roman-like gestures.

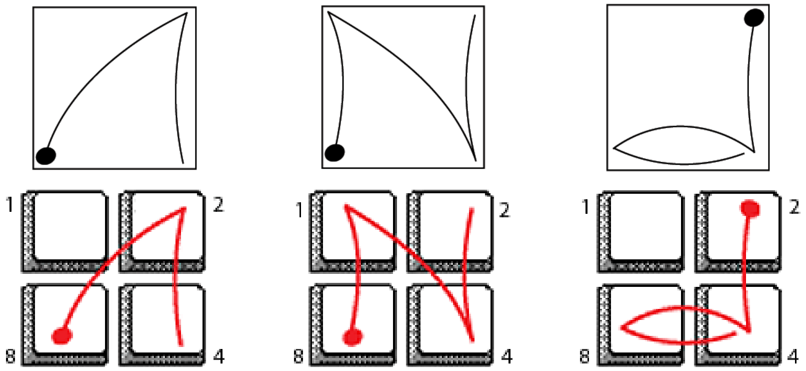


Figure 9.1. EdgeWrite letters mapped to four keys. Letters are defined by their sequence of corner-hits: "a" = 824, "n" = 8142, "d" = 2484.

With this four-key design, the KSPC of EdgeWrite's primary letter forms is 3.52. This is higher than the 5-key selection keyboard to which EdgeWrite will be compared, which has a KSPC of 3.24. But gestural methods do not require visual search like selection-based methods do. So in this case, there is a tradeoff between KSPC and visual search.

Adapting a unistroke stylus method to four keys requires a solution to the segmentation problem, since "stylus lift" is not relevant. For segmentation, Four-key EdgeWrite uses an adaptive timeout that adjusts on a per-letter basis to the speed at which a user makes a letter according to Equation 9.1.

$$T = F \cdot \frac{\sum_{i=2}^{n} tDOWN_i - tUP_{i-1}}{n-1} \tag{9.1}$$

In this equation, T is the time until segmentation, F is a multiplier preference ranging from 1.20 (expert) to 2.00 (novice), $tDOWN_i$ is the down-time of the i^{th} key-press, tUP_i is the up-time of the i^{th} key-press, and n is the total number of key-presses ($n > 1$).[14] Thus, for users who hesitate little between key-presses, segmentation occurs sooner than for users who hesitate more. After each key-down event, the timer is stopped so that segmentation cannot occur until all keys are up. When all keys are up, the timeout is computed and the timer restarted. When the timer elapses, segmentation occurs, meaning the corner sequence is recognized and reset.

9.3 Evaluation

This study compared Four-key EdgeWrite to predominant 3-key and 5-key methods. The study was conducted over 10 short sessions simulating "daily intermittent use." Only character-level EdgeWrite was tested, and continuous recognition feedback (§3.4.1) and non-recognition retry (§3.4.2) were implemented. Slip detection (§3.4.3) was not a part of this method.

[14] A special case is used for $n = 1$ in which a base timeout of 250 ms is used. This value is still multiplied by the F scaling factor.

9.3.1 Competitor Methods

Few-key text entry has predominantly been accomplished with either 3-key or 5-key selection-based methods. The latter are even in use on mainstream commercial products. These methods are reviewed in general in §2.3.8. Here, more specific details are given.

9.3.1.1 Three-Key Design

Although there are many possible layouts for 3-key text entry methods, the one in Figure 9.2 was found by a previous study to be "particularly promising" (MacKenzie 2002c). This design places SPACE to the far left, and the selector snaps there after each entry. This enables users to unhesitatingly move the selector to the right after each entry. We calculated its KSPC to be 10.53 using previously used letter frequencies (Soukoreff and MacKenzie 1995). Layouts with lower KSPC do exist, but these often increase visual search time at the expense of speed (Bellman and MacKenzie 1998). In a previous study, the design in Figure 9.2 was measured at 9.10 WPM and 2.11% total errors in a single session (MacKenzie 2002c). For this study, the design was augmented with BACKSPACE ('<') for error correction. Its placement to the left of SPACE does not increase KSPC because the selector snaps to SPACE and does not wrap.

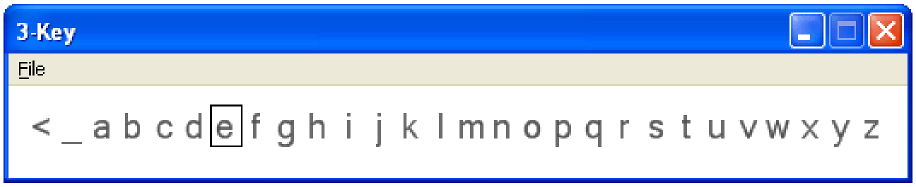
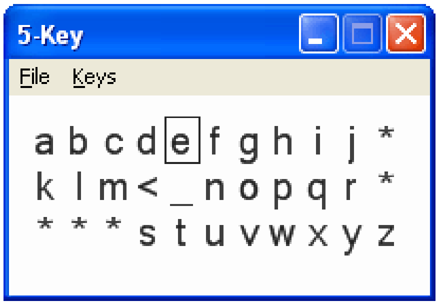


Figure 9.2. The 3-key design used in the study. The letter "e" is currently selected.

The 3-key method is greatly enhanced by key-repeat. The key-repeat times used in this study were taken from prior work (MacKenzie 2002c), set to 176 ms for the initial delay and 32.1 ms for the repeat delay. These are fast key-repeat times and enable high performance.

9.3.1.2 Five-Key Design

The 5-key method uses four keys to move over a matrix of letters and a fifth key for selection. The design in Figure 9.3 is based on the *Glenayre AccessLink II* pager (Figure 2.13) as described by MacKenzie (MacKenzie 2002b). Its alphabetic layout is optimized to put common letters "e", "s", "t", "n", "o" and "u" near SPACE. Where the Glenayre pager had punctuation marks, here we replaced them with asterisks since punctuation was not used in this study.



a b c d e f g h i j *
k l m < _ n o p q r *
* * * s t u v w x y z

Figure 9.3. The 5-key design used in the study.

The KSPC for the commercial product is 3.13. As with the 3-key method, the 5-key method is augmented with BACKSPACE ('<'). Its placement to the left of SPACE minimally increases KSPC to 3.24. This is 0.28 less than the Four-key EdgeWrite method. Like the 3-key method, the 5-key method employs snap-to-home (SPACE) and key-repeat.

Both the 3- and 5-key designs could reduce KSPC using optimization techniques. But KSPC is not the sole design factor, as visual search time is significant. Manufacturers have generally eschewed fully optimized keyboards, perhaps favoring users' first impressions over their extended performance. Certainly, the intermittent use of few-key text entry methods requires that they be quickly learnable.

9.3.2 Method

This section describes the experimental method used to evaluate the three-key, five-key, and Four-key EdgeWrite methods.

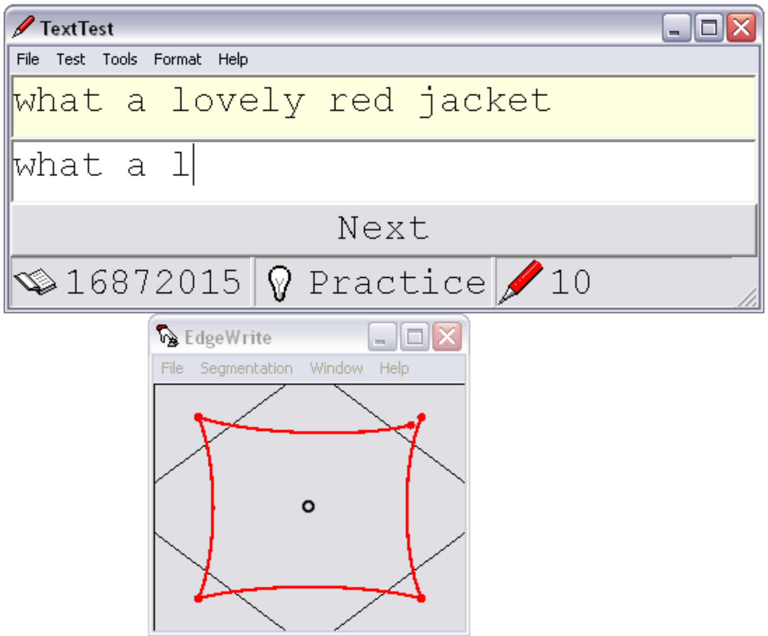
9.3.2.1 Subjects

Five subjects (2 female) ranging from 27 to 33 years old took part in the study over 10 consecutive days. Subjects were all right-handed daily computer users. Their keyboard typing rates were measured to provide context for their few-key results. Their mean typing speed was 70.79 WPM (σ=13.76) with 3.54% total errors (σ=1.22%). None of the subjects had used any of the few-key methods. They were paid $10 per session.

9.3.2.2 Apparatus

The test software shown in Figure 9.4 presented phrases from a standard corpus (MacKenzie and Soukoreff 2003). In all, the software logged 1600 test phrases, or about 50,000 characters. Backspace was supported, and subjects were not forced to remain synchronized with the presented text.

As in prior work (Bellman and MacKenzie 1998), subjects used a standard desktop numeric keypad with one hand to control the few-key methods. Subjects were told to use the key configuration they found most comfortable. They used three fingers with *3-key*, three or four fingers with *5-key*, and four fingers with *4-key EdgeWrite* (e.g., on the 1, 2, 4 and 5 keys). Note that with all these methods they used these fingers at the same time, holding them over the numeric keypad to enable a very limited form of touch-typing. Using the numeric keypad allowed for the comparison of these methods under "ideal" conditions: one-handed with familiar comfortably-sized keys. Custom devices could be tailored toward any of the techniques if the study's results warrant further design.



what a lovely red jacket

what a l

16872015 Practice 10

Figure 9.4. The TextTest text entry test software and Four-key EdgeWrite. The *4-key-noviz EdgeWrite* condition lacked the stroke visualization window shown here.

9.3.2.3 Procedure

Each session consisted of 2 warm-up phrases and 8 test phrases for each of the 4 methods. These 10 phrases took ~5 minutes to complete per method. To see how Four-key EdgeWrite fared with no stroke visualization, a condition called *4-key-noviz EdgeWrite* was included. In this condition, the stroke window in Figure 9.4 was removed. Although this resulted in two EdgeWrite methods per session (*viz* and *noviz*), the total time for EdgeWrite was still just ~10 minutes. Moreover, the two selection-based methods shared common features, e.g., key-repeat times and certain keys. The presentation order of the four methods was counterbalanced according to a Latin Square.

During the 2 warm-up phrases, an EdgeWrite character chart was displayed, but this chart was not shown during the 8 test phrases. Not surprisingly, subjects had to guess many strokes in the early sessions. Although not showing a character chart undoubtedly hurt Four-key EdgeWrite, it allowed for the assessment of learnability without aids.

9.3.3 Results

This section presents the results of the longitudinal comparison of the few-key text entry methods.

9.3.3.1 Analysis

The data were analyzed using a mixed model analysis of variance in which *Method* (3-key, 5-key, 4-key, 4-key-noviz) and *Session* (1–10) were fixed effects and *Subject* was a random effect (Littell *et al.* 1996). The model included *Method Order* (1–4), but no order effects were found on speed ($F_{3,164}$=0.58, p=0.63), indicating adequate counterbalancing.

9.3.3.2 Speed

Mean WPM over all 10 sessions were: *3-key* 9.08 (σ=1.31), *5-key* 10.62 (2.61), *4-key EdgeWrite* 12.50 (3.91), *4-key-noviz EdgeWrite* 12.94 (3.99). By session 10 these improved to: *3-key* 9.81 (1.33), *5-key* 12.86 (2.26), *4-key EdgeWrite* 15.95 (3.22), *4-key-noviz EdgeWrite* 16.86 (3.06). Figure 9.5a depicts speeds over sessions.

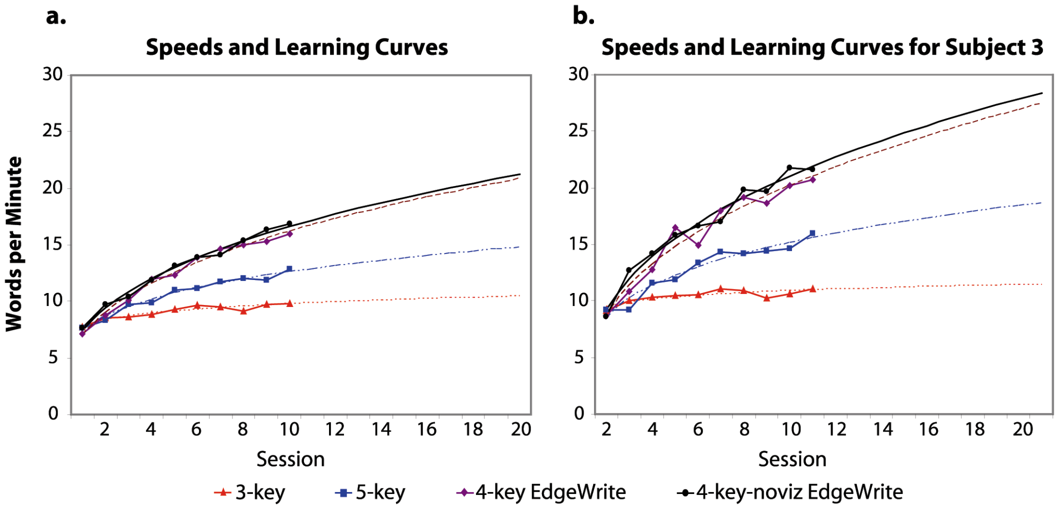


Figure 9.5. Average speeds over sessions and fitted learning curves. The right graph shows the subject who was fastest with all four methods.

A main effect for WPM is significant over 10 sessions for *Session* ($F_{1,164}$=350.63, p<.01), *Method* ($F_{3,164}$=72.61, p<.01), and *Session*×*Method* ($F_{3,164}$=27.91, p<.01). That is, subjects sped up over time and did so at different rates with each method. Contrasts show that the speeds of the EdgeWrite methods were not detectably different from each other ($F_{1,164}$=1.45, ns), but were significantly faster than *3-key* ($F_{1,164}$=202.78, p<.01) and *5-key* ($F_{1,164}$=67.21, p<.01). Also, *5-key* was faster than *3-key* ($F_{1,164}$=26.88, p<.01).

Subjects learned the EdgeWrite methods quickly, overtaking the selection-based methods by session 2. The learning curve equations fitted to the power law of learning (Card *et al.* 1983) for Figure 9.5a are:

- $y = 6.998x^{0.364}$ $R^2 = 0.99$ 4-key EdgeWrite
- $y = 7.389x^{0.352}$ $R^2 = 0.99$ 4-key-noviz EdgeWrite
- $y = 7.398x^{0.231}$ $R^2 = 0.97$ 5-key
- $y = 7.822x^{0.097}$ $R^2 = 0.90$ 3-key

Subject 3 performed the fastest with all four methods (Figure 9.5b). Over 10 sessions, his WPM peaked at: *3-key* 11.03 (1.35), *5-key* 15.99 (1.65), *4-key EdgeWrite* 20.75 (1.47), *4-key-noviz EdgeWrite* 21.73 (1.50). Subject 3's single fastest phrase occurred in session 9 with the *4-key EdgeWrite* method at 24.06 WPM and 0.0% total errors. This is remarkable for "gesture-typing" on four keys after ~90 minutes of practice.

Subject 3's learning curves for Figure 9.5b are:

- $y = 8.688x^{0.384}$ $R^2 = 0.96$ 4-key EdgeWrite
- $y = 9.237x^{0.375}$ $R^2 = 0.98$ 4-key-noviz EdgeWrite
- $y = 8.553x^{0.261}$ $R^2 = 0.94$ 5-key
- $y = 9.470x^{0.064}$ $R^2 = 0.72$ 3-key

9.3.3.3 Uncorrected Errors

Uncorrected errors are graphed in Figure 9.6a. Means over 10 sessions were: *3-key* 0.60% (0.75), *5-key* 0.96% (1.10), *4-key EdgeWrite* 1.69% (1.71), *4-key-noviz EdgeWrite* 2.00% (2.05). A main effect for uncorrected errors is significant for *Session* ($F_{1,164}$=4.00, p<.05) and *Method* ($F_{3,164}$=9.05, p<.01) but not for *Session×Method* ($F_{3,164}$=0.43, ns). Contrasts show that the EdgeWrite methods were not detectably different from each other ($F_{1,164}$=1.37, ns), but left more errors than *3-key* ($F_{1,164}$=22.71, p<.01) and *5-key* ($F_{1,164}$=10.42, p<.01). Also, the *3-key* and *5-key* methods were not significantly different from each other ($F_{1,164}$=1.73, ns).

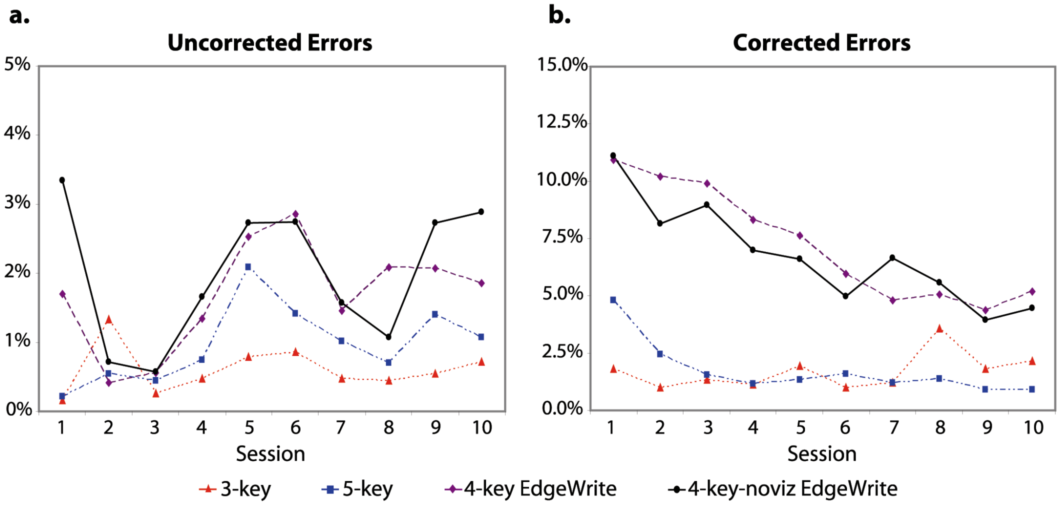


Figure 9.6. Average error rates over sessions for the few-key methods. Lower values are better. Note the different ranges on the *y*-axes.

9.3.3.4 Corrected Errors

Corrected errors are graphed in Figure 9.6b. Mean corrected error rates over 10 sessions were: *3-key* 1.69% (1.61), *5-key* 1.73% (1.49), *4-key EdgeWrite* 7.23% (4.26), *4-key-noviz EdgeWrite* 6.74% (4.30). A main effect for corrected errors is significant for *Session* ($F_{1,164}$=38.93, p<.01), *Method* ($F_{3,164}$=64.32, p<.01), and *Session*×*Method* ($F_{3,164}$=9.87, p<.01). Contrasts show the EdgeWrite methods were not detectably different from each other ($F_{1,164}$=0.75, ns), but were more error prone than *3-key* ($F_{1,164}$=130.82, p<.01) and *5-key* ($F_{1,164}$=125.69, p<.01). Also, *3-key* and *5-key* were not significantly different from each other ($F_{1,164}$=0.02, ns). Although more error prone overall, EdgeWrite errors dropped significantly over sessions ($F_{1,80}$=47.43, p<.01) and did not appear to level off, so they may have continued to drop over future sessions.

9.3.3.5 Discussion

Despite being gestural, the EdgeWrite methods were quickly learned, demonstrating the benefits of mnemonic gestures. It is clear from the learning curves that more sessions, or sessions offering more practice, are needed to find the asymptotic speeds of the EdgeWrite methods. The study's results also indicate that removing a visualization of the letter stroke being made is not detrimental to EdgeWrite, which may have positive implications for wearable contexts.

It is not surprising that the gestural methods were less accurate, although uncorrected error rates were quite low for all four methods (≤2%). It may be possible to improve accuracy by intelligently handling premature segmentations or by preventing backspaces for a brief period after non-recognitions, since subjects would sometimes "feel" a mistake and hastily make a backspace, only to find their mistake was a non-recognition and their backspace removed a previously correct character.

9.4 Four Capacitive Sensors

Four-key EdgeWrite used a numeric keypad to quickly prototype a new version of EdgeWrite using only four sensors. Another way of implementing EdgeWrite is with four *capacitive sensors* (Figure 9.7). Four-sensor EdgeWrite is somewhat similar to Touchpad EdgeWrite (§6.2.2), except that the sensing surface is not continuous but discrete. Each capacitive-sensing corner is only capable of reporting whether or not it is being touched; thus, it might be called a "binary" sensor. Segmentation cannot therefore be achieved with "finger lift" as it can on a touchpad, since whenever the finger moves between corners, it will appear to be lifted. In Four-sensor EdgeWrite, an adaptive timeout is used for segmentation according to Equation 9.1.

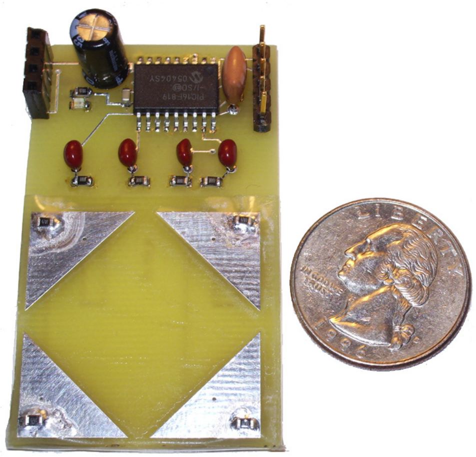


Figure 9.7. Four-sensor EdgeWrite uses four charge-pump capacitive sensors (the metal-foil triangles). Four surface mount resistors are used as tactile bumps.

9.4.1 Implementation

Capacitive sensors can be implemented a variety of ways. Four-sensor EdgeWrite uses four "charge-pump" capacitive sensors[15]. This type of capacitive sensor is fairly robust and easy to implement.

[15] This circuit was described by Johnny C. Lee and Scott E. Hudson in 05-897 *Electronics prototyping for HCI*, Carnegie Mellon University, Spring 2006.

The sensor pad can be most anything that will hold a charge, such as a mass of tinfoil, an insulated copper wire, a metal plate, or thin foil-like pads like those used for the model in Figure 9.7. The metal does not need to be exposed, but can be covered by a thin layer of tape, plastic, or rubber.

A schematic for a single charge-pump sensor is shown in Figure 9.8. Each sensor pad is coupled with a dedicated capacitor C_0 measuring 0.1µF. This capacitance is large relative to the small capacitance C_1 of the sensor pad. Two microprocessor pins are used in the circuit as well, one for charging the sensing pad (PAD), and one for pulling charge from the sensor pad into the large capacitor (DRAIN). A resistor *R* (~50Ω) is also included as a precaution so that a short cannot occur between the sensor pad and the PAD pin.

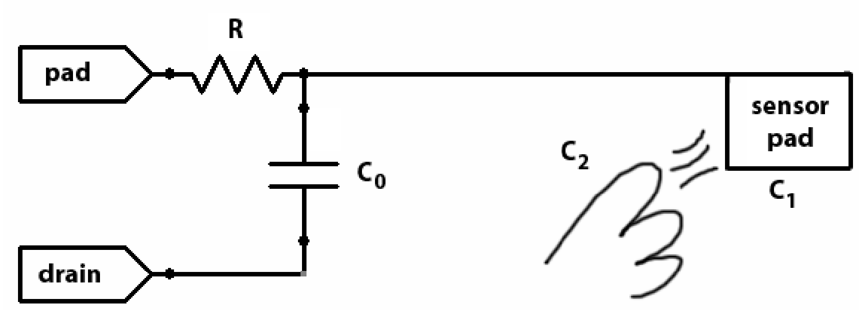


Figure 9.8. The schematic for one charge-pump capacitive sensor.

The sensor works by counting the number of cycles it takes to charge the capacitor C_0 such that PAD will read "high." The capacitor C_0 is charged by "pumping" little bits of charge from the sensor pad C_1 into C_0 over time. When a finger is absent, the number of pumps will be some value *A*. When a finger is present, it brings with it a much larger capacitance C_2, and the number of necessary pumps *P* to charge C_0 will be much less than *A*. When this occurs, a finger is detected. The algorithm for a single sensor pad is:

1. Set both PAD and DRAIN to low to discharge the circuit. Set COUNT to zero.
2. Set DRAIN to high impedance to effectively disconnect it.
3. Set PAD to high to send charge into C_1, and C_2 if it is present.
4. Set PAD to high impedance to effectively disconnect it.
5. Set DRAIN to low to pull the charge from C_1 (and C_2) into C_0.
6. Read the value of PAD:

```
        if 0: increment COUNT and goto step 2.
        if 1: a finger is present if COUNT ≤ THRESHOLD; goto step 1.
```

For the actual circuit, four complete sensors like the one shown in Figure 9.8 were connected to eight pins on a *Microchip Technology PIC16F819* microprocessor (http://www.microchip.com). Because of some pernicious side effects that arose when calling functions to rapidly change the states of the pins for four simultaneous sensors, direct memory access had to be used instead, where function calls were circumvented by directly setting 1's and 0's at memory offsets on the processor. The result was a fast sensing loop capable of supporting writing with a finger.

The circuit schematic is shown in Figure 9.9. The schematic was developed by adapting a previous circuit designed by Johnny C. Lee that supports in-circuit programming using a *Melabs serial programmer* (http://www.melabs.com) and communication with the PC over a *Pololu USB-to-serial adapter* (http://www.pololu.com). The extended circuit includes four capacitive sensors and works with a desktop application written in C# that receives serial events and translates them into EdgeWrite character strokes, capable of entering text into any application.

9.4.2 Evaluation

Four-sensor EdgeWrite has not been formally evaluated. It was intended to demonstrate the feasibility of EdgeWrite text entry on cheap sensing technology with only four capacitive sensors. More complex sensing arrays like those used in touchpads or PDA screens are thereby demonstrated to be unnecessary for EdgeWrite text entry.

In an effort to establish ballpark figures for performance, I tested character-level Four-sensor EdgeWrite myself by entering 15 phrases presented by the TextTest program. My overall speed for these phrases was 11.38 WPM (σ=0.93). My fastest speed for a single phrase was 13.09 WPM, while my slowest speed was 9.95 WPM.

Uncorrected and corrected error rates were 1.49% (2.32) and 2.87% (2.82), respectively. These numbers are comparable with many of the "full fledged" versions of EdgeWrite described in this dissertation, which is noteworthy given the minimal sensing technology used to implement Four-sensor EdgeWrite.

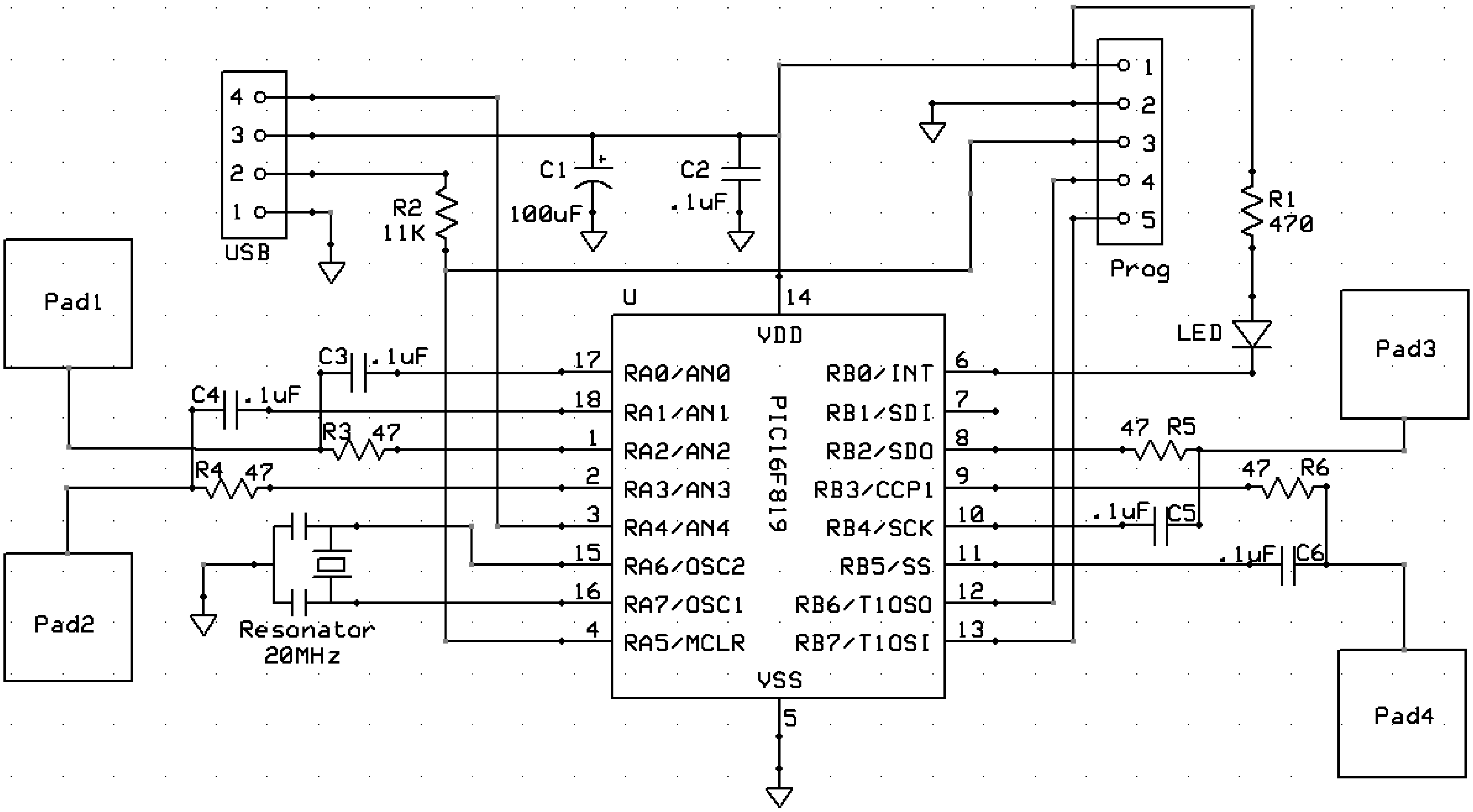


Figure 9.9. The schematic for Four-sensor EdgeWrite. The four pads correspond to the four metallic triangles in Figure 9.7. Each one is equivalent to the “sensor pad” in Figure 9.8.

Chapter 10

EdgeWrite by Others

Other researchers have created versions of EdgeWrite using the publicly available XML character set (Appendix A) or the EdgeWrite DLL (§3.6.1). This chapter briefly reports on these projects as a way of further demonstrating the versatility of EdgeWrite.

10.1 Steering Wheel EdgeWrite (González)

For his undergraduate Bachelor's honors thesis in computer science at Carnegie Mellon University, Iván E. González implemented a number of input techniques for steering wheels intending to provide better access to car navigation systems.

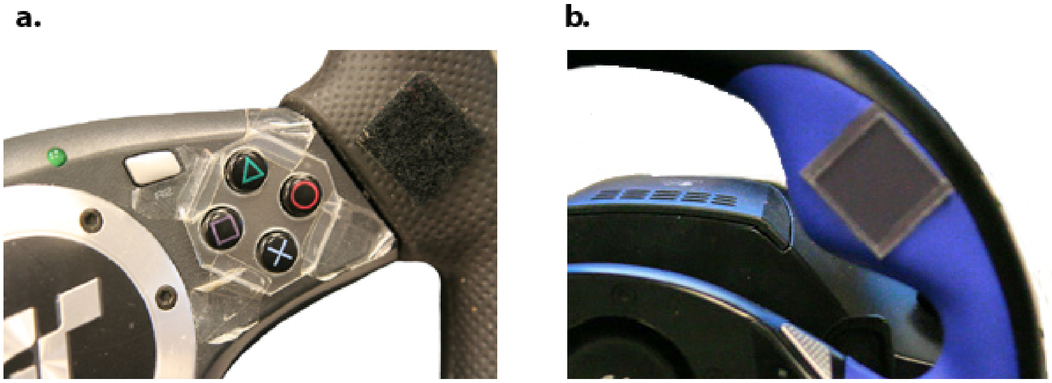


Figure 10.1. Two EdgeWrite prototypes for use with steering wheels. The left one uses small buttons. The right one uses a small touchpad. Both are controlled by the thumb.

Among his techniques were two versions of EdgeWrite. One version used the four built-in buttons on a Logitech steering wheel used for gaming on desktop PCs (Figure 10.1a). An additional plastic piece made it easier for a user's thumb to slide over these buttons. The other version used a small touchpad, a Synaptics *StampPad*, embedded in the surface of a different computer game steering wheel (Figure 10.1b). The StampPad worked automatically with the Touchpad EdgeWrite software (§6.2.2). González also implemented a number of selection-based methods for the StampPad, such as *repeated stroking*, *rate-controlled scrolling*, and *radial dialing* (e.g. like on an Apple iPod) to select from a list of street names in a mock heads-up navigation system.

After preliminary testing, González chose the StampPad version to use in a study of street name selection while stationary and while driving. González conducted the study with the STISIM driving simulator (http://www.systemstech.com) (Figure 10.2).

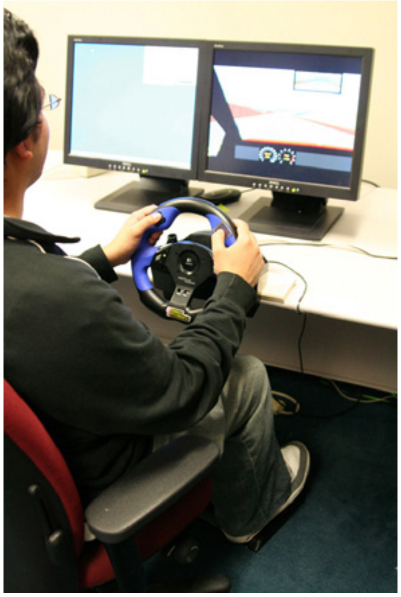


Figure 10.2. The experimental setup with the STISIM driving simulator. The list of street names and input techniques are shown on the left monitor as peripheral heads-up display.

The same four subjects who participated in the prior study using the mobile phone isometric joystick were used (§8.3). Besides EdgeWrite, they also used a number of selection-based techniques. For stationary street name entry using the steering wheel, González found that EdgeWrite beat all five on-screen keyboard techniques when selecting from a set of 228 street names. However, it was faster to directly scroll through the list instead of spell out the name. When the best of these techniques were tested while doing simulated driving, however, EdgeWrite ranked *first* in avoiding center-line crossings, speeding, and crashes, and was a close second in street name selection time. The only measure in which EdgeWrite fared worse than the list-scrolling techniques was in touching the shoulder of the road. More testing is needed for definitive results, but this preliminary study shows that EdgeWrite on a steering wheel may be competitive with the off-wheel navigation systems using on-screen keyboards that are used in cars today.

10.2 WatchPad EdgeWrite (Blaskó and Feiner)

Gabor Blaskó and Steven Feiner of Columbia University implemented a finger-controlled version of EdgeWrite for the IBM/Citizen WatchPad 1.5 (Blaskó and Feiner 2004). This wrist watch runs the Linux operating system for which Blaskó wrote a version of EdgeWrite (Figure 10.3).

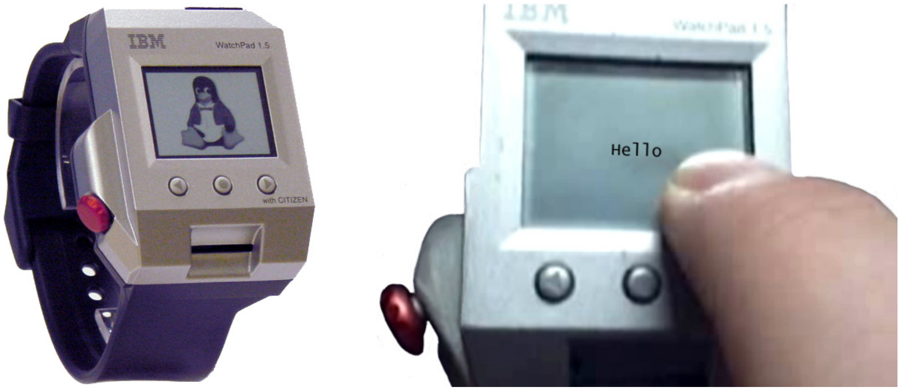


Figure 10.3. The IBM/Citizen WatchPad 1.5 being used for EdgeWrite text entry.

EdgeWrite is particularly appropriate here because of the watch's raised square bezel. Also, Blaskó and Feiner already had developed interaction techniques for the watch based on the four corners of the watch's screen (Figure 10.4). Thus, the conceptual step to using EdgeWrite was an obvious one.

The researchers did not evaluate EdgeWrite text entry on the watch, but their implemented prototype was capable of transmitting entered characters to a desktop terminal via a wireless connection, making a future evaluation easy to conduct.

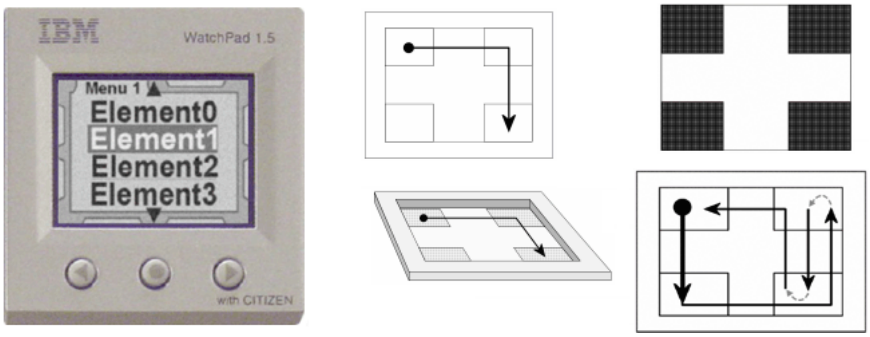


Figure 10.4. Blaskó's and Feiner's interaction techniques for the WatchPad already used corners and edges. Images taken from (Blaskó and Feiner 2004). Used with permission.

10.3 Edgeless EdgeWrite (Andersen and Zhai)

Tue Haste Andersen and Shumin Zhai implemented an EdgeWrite variant called *Edgeless EdgeWrite*, or *ELEW* (Andersen and Zhai 2004). ELEW removes the need for strokes to hit absolute corner locations by using a proportional shape matching gesture recognizer. This allows ELEW to remove the plastic edges of a physical square and to redefine some of the EdgeWrite letters accordingly (Figure 10.5).

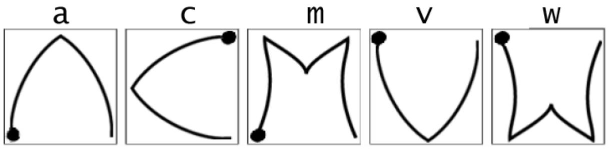


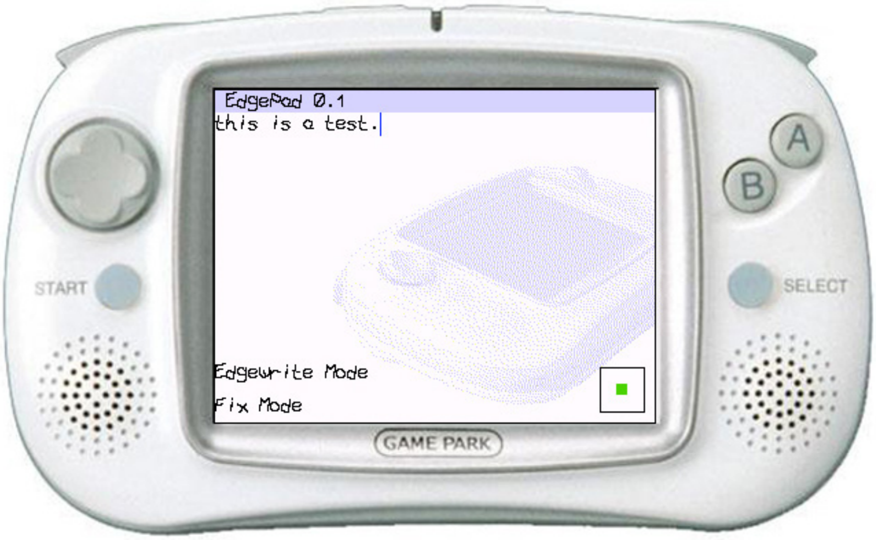
Figure 10.5. Edgeless EdgeWrite has some different letters than EdgeWrite. Image adapted from (Andersen and Zhai 2004). Used with permission.

ELEW was not intended as a version for people with motor impairments. Instead, ELEW was used in an experiment to test whether audio feedback, including musical feedback, improves people's experience of writing strokes. Musical feedback was synthesized in real-time according to the eight possible directions one might move in an ELEW character. Thus, the straight-line properties of the EdgeWrite alphabet were a useful asset to this experiment.

After running 16 subjects, Andersen and Zhai did not find any significant differences among their audio conditions in terms of speed or accuracy of entry. However, they did note subjects' significant preference for the most "musical" of the feedback conditions, wherein the synthesized music responded in real-time to the speed of the writer. The study also found a comparable speed and accuracy to regular EdgeWrite with somewhat less practice, perhaps due to the redesigned alphabet.

10.4 EdgePad for the GP32 (Ward)

Ian Ward implemented a version of EdgeWrite called *EdgePad* for the four-way directional pad on a GP32 game device (http://gp32.sector808.org/edgewrite.php). The GP32 game device is an open source handheld platform that does not have a touch screen or stylus (Figure 10.6). In fact, it only has a directional pad and a few buttons. Not surprisingly, the device's existing text entry methods are lacking, which prompted Ward's efforts. In a personal communication, Ward stated that he implemented EdgePad because the GP32 is a small device with limited screen space, and he was frustrated with the slow speeds of its on-screen keyboards. He said that with EdgePad, he uses a fixed timeout to segment between letters when the D-pad returns-to-center. He reported that he can use EdgePad "accurately and quickly without looking at the screen." His website says he can enter text at 10 WPM and "hardly make any mistakes."



this is a test.

Figure 10.6. The GP32 device and the EdgePad text entry utility. Used with permission.

Ward, who goes by "Woogal," was interviewed by *Hooka*, a GP32 fan site containing interviews of top GP32 developers. What follows is an excerpt:[16]

Hooka: What kind of stuff do you read/write with lazy reader and EdgePad?

Woogal: I read a lot of old sci-fi, particularly H.G.Wells.

Hooka: You started EdgePad before you had a chatboard, I remember trying to use it and not getting the hang of it. Why did you decide to do an EdgeWrite style writing for GP32?

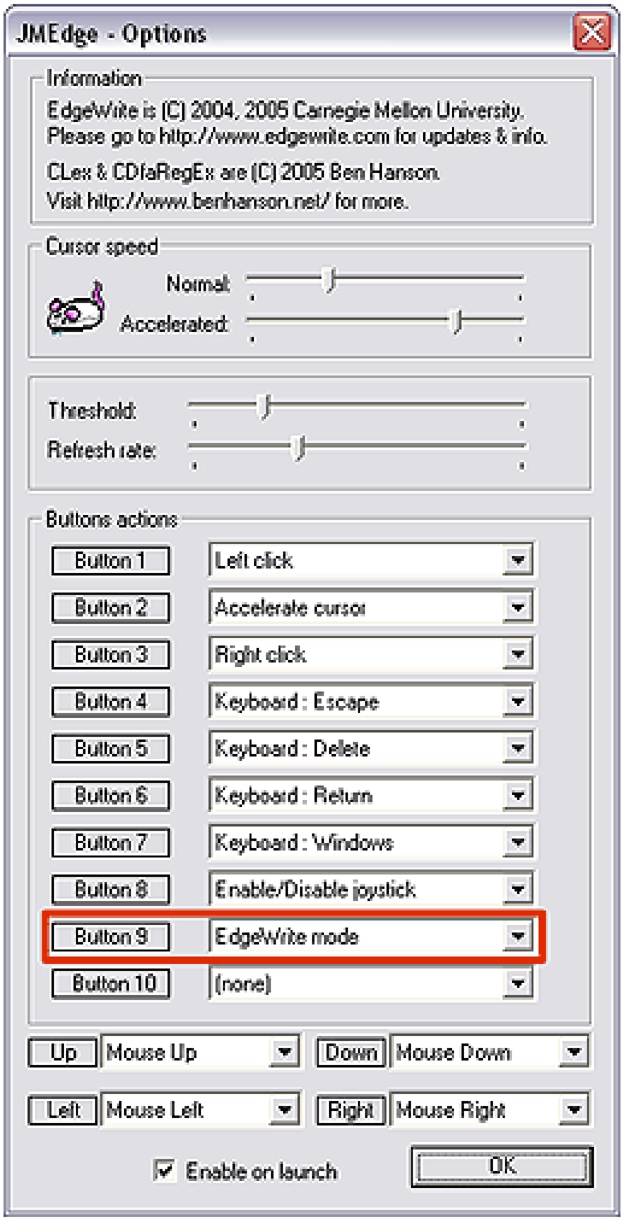
[16] The complete interview can be found at http://www3.telus.net/public/hooka/woogal.html.

> Woogal: I had tried using the existing text entry systems on the GP32 but wasn't really happy or comfortable with any of them. I started to do a little research into alternatives and came across a paper on using EdgeWrite for joystick input. The idea intrigued me and fitted all my ideal criteria (able to use it without looking at the screen, and not have most of the very limited screen space taken up with an on screen keyboard), so I thought I'd give it a go.

10.5 JMEdge (Richez)

Xavier Richez wrote *JMEdge* (http://perso.orange.fr/xavier.richez/jmedge-en.html), a Windows application that runs in the System Tray and translates joystick events into text input. JMEdge is similar to Joystick EdgeWrite (§5), except that it lacks any visualization of strokes being made, and it requires a button-press to segment between letters—return to center is not used. This makes it slower than Joystick EdgeWrite and seemingly more error prone, although a formal evaluation has not been conducted.

JMEdge is part of the *JoyMouse*++ system that enables a joystick to emulate the mouse on a desktop computer. The control panel for JoyMouse++ and JMEdge is shown in Figure 10.7.



JMEdge - Options

Information

EdgeWrite is (C) 2004, 2005 Carnegie Mellon University.
Please go to http://www.edgewrite.com for updates & info.

CLex & CDfaRegEx are (C) 2005 Ben Hanson.
Visit http://www.benhanson.net/ for more.

Cursor speed

Normal:
Accelerated:

Threshold:
Refresh rate:

Buttons actions

Button 1	Left click
Button 2	Accelerate cursor
Button 3	Right click
Button 4	Keyboard : Escape
Button 5	Keyboard : Delete
Button 6	Keyboard : Return
Button 7	Keyboard : Windows
Button 8	Enable/Disable joystick
Button 9	EdgeWrite mode
Button 10	(none)

Up	Mouse Up	Down	Mouse Down
Left	Mouse Left	Right	Mouse Right

☑ Enable on launch

Figure 10.7. The JoyMouse++ and JMEdge control panel for enabling EdgeWrite text entry using a PC-compatible joystick. Button 9 currently switches to EdgeWrite mode.

Chapter 11

A New Character-level Error Analysis*

Over the course of this work, numerous text entry evaluations were conducted and hundreds of thousands of text entry events were analyzed. The *TextTest* and *StreamAnalyzer* programs (§2.4) were built to facilitate many of these studies and analyses. These tools support the running of studies in the *unconstrained text entry paradigm* (Soukoreff and MacKenzie 2001, Soukoreff and MacKenzie 2003). Although this paradigm has many benefits, it lacks a character-level analysis of the *input stream*—those events that occur during text entry but that do not appear in the final transcribed string. This chapter presents the algorithmic work for such an analysis. As such, it constitutes a contribution to the literature on text entry evaluation.

11.1 Motivation

Measuring accuracy in unconstrained text entry experiments (§2.4.3) is not entirely straightforward. For example, in Figure 11.1, *P* is a string presented to a subject and *T* is the subject's transcription. How many errors are there?

```
P: the quick brown
T: the quicxk brown
              ^^^^^^^^
```

Figure 11.1. An example presented (*P*) and transcribed (*T*) string.

* Parts of this chapter are adapted from (Wobbrock and Myers 2007).

In Figure 11.1, a simple pair-wise comparison suggests that everything after the "c" in *T* is in error. But this is probably not the case. More likely, there was one insertion error, the "x" in *T*. Automatically detecting such errors between *P* and *T* has been the subject of recent work (Soukoreff and MacKenzie 2001, Soukoreff and MacKenzie 2003).

What if we consider not just the final transcribed string but the entire *input stream*, the record of all input events produced by the subject? If errors were made and corrected, the input stream would hold more error information than the transcribed string. This information could be useful to designers and evaluators for improving techniques. For example, the transcription in Figure 11.1 could have been produced from the input stream (*IS*) in Figure 11.2:

IS: f←**t**n←**he** p←**qu**l←**i**k←**cxk** **b**fo←←**rown**

Figure 11.2. An example input stream (*IS*) resulting in *T* from Figure 11.1. Transcribed letters are in bold and backspaces are represented by "←".

In the input stream, "←" symbols indicate backspaces and bold letters compose the final transcribed phrase *T*. Clearly, there were more errors in transcribing this text than *T* alone reveals. Thus, an analysis of *T* without *IS* paints an impoverished picture of errors. We *could* try to analyze *IS* for errors, but input streams are messy and ambiguous, and determining errors within them is difficult. The core challenge in determining input stream errors is assessing *intention*. What is the subject trying to enter at every character position? Because of the difficulty in answering this question, many previous text entry evaluation strategies unnaturally constrained text entry experiments so as to make the determination of errors, particularly character-level ones, trivial. But this comes at the cost of so totally altering the natural transcription process that results from such studies are cast into doubt.

For example, one artificially constrained experimental paradigm disallows erroneous characters completely. As the subject transcribes text on a line beneath the presented text, any attempted character that does not match the character directly above it is not displayed. Often in this paradigm the entry of an erroneous character results in an audible "beep." Subjects may incur many successive beeps without ever seeing characters appear because their entries do not match the presented character at their position. What's worse, backspace is rendered irrelevant, even though backspace is the second most common keystroke in real desktop text entry after space (MacKenzie and Soukoreff 2002b). But inferring intention in this paradigm is trivial, because one assumes a subject is always trying to enter the next character in the presented text, even though this is not always true. The experience for subjects entering text in these evaluations is highly unnatural and potentially frustrating, since each error creates a "road block" that stops them abruptly, often for many characters. Nevertheless, this paradigm's ease of use has caused many to employ it (Venolia and Neiberg 1994, Isokoski and Kaki 2002, Evreinova *et al.* 2004, Ingmarsson *et al.* 2004).

Another way to unnaturally constrain text entry experiments is to allow errors but to prevent error correction, usually by disabling backspace. Two examples are in the evaluations of the OPTI keyboard (MacKenzie and Zhang 1999) and the Half-Qwerty (Matias *et al.* 1996). In this paradigm, subjects are free to enter any letter, but once entered, letters can not be backspaced. Characters are deemed erroneous if they do not agree with the presented letter at the current position. Erroneous characters are often accompanied by an audible "beep" and accuracy is calculated as a pair-wise comparison between the letters of *P* and *T*. Thus, when subjects make errors, they are forced to mentally "catch up" to the presented text by entering the next letter at the current position, rather than by trying the missed letter again. Maintaining synchronicity is therefore of utmost importance, and single insertions often result in multiple successive out-of-synchronization errors, also known as "error chunks" (Matias *et al.* 1996). Given the importance and prevalence of error correction during real text entry, this paradigm, like the previous one, is very artificial and often frustrating for users.

A third approach has been simply to ignore errors altogether, for example by using paper mockups of text entry methods. In this paradigm, errors are not recorded, analyzed, or reported. Surprisingly, some published studies fall into this category (Lewis *et al.* 1999b, MacKenzie *et al.* 1999, Rodriguez *et al.* 2005). The obvious drawback here is that speed and accuracy are tradeoffs, and analyzing one without the other encourages unreliable and flimsy comparisons.

Unfortunately, all three of these contrived paradigms are unnatural when compared to text entry outside the lab. This fact has motivated recent developments in automated error rate calculation (Soukoreff and MacKenzie 2001, Soukoreff and MacKenzie 2003) and the advent of the *unconstrained* text entry evaluation paradigm. In unconstrained text entry experiments, subjects are presented with phrases and are told to transcribe them "quickly and accurately." Subjects are neither forced to maintain synchronicity with the presented text nor are they required to fix errors—but they may and usually do. In essence, they are free to enter text nearer to how they would do it in "the real world." Importantly, a subject's subjective experience more closely matches their customary text entry experience, where error beeps, stalled text cursors, and error chunks are not the norm. The data acquired in unconstrained evaluations are automatically analyzed by algorithms designed to accommodate them. This chapter contributes to this set of algorithms.

11.1.1 Aggregate and Character-level Errors

Aggregate error rates give an indication of a text entry method's accuracy over all entered characters. For example, the keystrokes per character (KSPC) metric (§2.4.2), which was used in the evaluation of Stylus EdgeWrite (§4.3.1), is an aggregate metric. As stated, KSPC is a ratio of all entered characters, including backspaces, to final characters appearing in the transcribed string. Although the calculation of KSPC requires the input stream, it only requires a *count* of characters entered, not the discovery of *what those characters were intended to be*. This is also true of a widely-used definition of "corrected errors" (§2.4.3), which treats any backspaced character as an error, even if the erased character was, in fact, correct. A later paper acknowledged this limitation (Soukoreff and MacKenzie 2004), separating backspaced letters into two categories: *corrected-but-right* and *corrected-and-wrong*. However, that paper did not offer any algorithms for performing this separation, which is something the current work provides.

Many evaluators have reported KSPC error rates based on the number of backspaces made during entry (Sears and Zha 2003, Wobbrock *et al.* 2003b, Gong and Tarasewich 2005), but these are aggregate error rates, not character-level ones. While aggregate error rates provide a baseline for comparison, they are not sufficiently fine-grained to aid designers in targeting problematic characters. Designers and evaluators need more than aggregate measures; they need an indication of what is happening *at the level of individual characters*. For example, a character-level error analysis can tell us the probability of entering a "y" when attempting a "g". It can also tell us which characters are prone to errors of insertion, omission, or substitution, or which characters are particularly slow or fast to produce.

Until now, however, the unconstrained text entry paradigm has lacked a formal character-level error analysis that handles input streams. While the unconstrained experimental paradigm is a significant advance over the more artificial paradigms mentioned in the previous section, the lack of a character-level error analysis for input streams is a serious drawback. Thus far, character-level error analyses for the unconstrained paradigm have focused on presented and transcribed strings (MacKenzie and Soukoreff 2002a), not input streams. As mentioned, the difficulty in analyzing input streams is that they are fraught with ambiguity, making it hard to discern the subject's intention at each character position. This chapter addresses this ambiguity by using a small set of reasonable assumptions. It argues that over the course of a text entry study, the number of times the assumptions are invalid will be vastly outweighed by the number of times they are valid, a claim supported by empirical results. These carefully formulated assumptions enable the unlocking of the rich character-level information of input streams.

11.1.2 Advantages of Using Input Streams

Analyzing input streams from unconstrained text entry evaluations has a number of practical benefits for designers and evaluators of text entry methods. Among these benefits are:

- The input stream usually yields more data per trial than the transcribed string, since by definition $|IS| \geq |T|$. Therefore, depending on the research questions being asked, analyzing the input stream for character-level errors may allow us to run fewer trials and save time and money on evaluations, which are often time consuming and expensive (Jeffries *et al.* 1991). This will be the case if error data is what we are after. However, if we are interested in learning rates as measured by speeds over sessions, analyzing *IS* will not reduce the number of required sessions.

- When instructed to "enter the text quickly and accurately" (Soukoreff and MacKenzie 2003), subjects tend to fix most, if not all, of their errors in text entry trials. For example, in the study accompanying a character-level error analysis from the prior work (MacKenzie and Soukoreff 2002a), subjects left only 2.23% errors in *T*. Other studies show even fewer uncorrected errors: 0.79% (Soukoreff and MacKenzie 2003), 0.53% (§5.3.2), and 0.36% (§4.3.1). In the extreme case, if subjects correct all errors, *P* and *T* will be identical and no character-level error information will be available. Such a contingency does not reduce the value of *IS*, however, since *corrected errors*—the errors subjects made but fixed—are still captured.

- Speed and uncorrected errors are tradeoffs in text entry (MacKenzie and Soukoreff 2002b). Therefore, to equitably compare speeds, some experiments (Lewis 1999) have required *perfect transcription*, where leaving errors in *T* is not permitted. But character-level error analyses of perfect transcription studies are useless using only *P* and *T* since they will always be identical. Analyzing *IS*, on the other hand, allows for the extraction of character-level results, even in perfect transcription studies.

- For stroke-based or handwritten entry such as EdgeWrite and Graffiti (Blickenstorfer 1995), one possible outcome of an attempted character is a *non-recognition*. By definition, *T* cannot contain non-recognitions, but *IS* can. Therefore, looking at *IS* can be valuable to designers trying to identify characters that are difficult to recognize in stroke-based entry methods.

11.1.3 Limitations of this Analysis

Like most automated analyses, the current work has limitations. It is designed to analyze data from unconstrained text entry experiments in which subjects transcribe text rather than generate it. Although generating text is more natural, it has numerous problems when used in experiments, such as introducing thinking time, complicating the identification of errors, and abdicating control of letter and word distributions (MacKenzie and Soukoreff 2002b, MacKenzie and Soukoreff 2003). For these reasons, text entry evaluations nearly always involve text transcription.

In the unconstrained paradigm, text is assumed to flow serially forward with the entry of new characters and backward with the correction operation (i.e. backspace). Therefore, the current analysis does not accommodate "random access" editing using cursor keys, multi-character selection, or the mouse cursor. Indeed, a comprehensive analysis of text entry performance is a current goal of researchers, but even the metrics for such an analysis are not yet understood, let alone algorithms for automated measurement.

The current analysis does not accommodate method-specific tokens within the input stream, such as the individual key-presses in *Multitap* (e.g., 222-NEXT-2-8 = "cat"). Rather, the current analysis examines input streams containing all entered characters and backspaces in a method-agnostic fashion that makes it suitable for any character-level text entry technique. Method-specific analyses still have to be implemented in a method-specific fashion, and are viewed as supplements to this work.

The current work functions well for any type of character-level method. These methods include typing, unistrokes, stylus keyboards, eye-tracking keyboards, thumbwheels, character recognizers, and so on—any technique that produces one character at a time. The results are less informative, however, for word-level methods that produce multi-character chunks. Examples of such methods are word-producing *sokgraphs* (Zhai and Kristensson 2003), word prediction systems (Wobbrock and Myers 2006b), and speech recognition systems. Although the algorithms will work for such methods, character-level errors are less relevant since the "character" is not the unit of production. The algorithms *could* be adapted to accommodate word-level entry by treating multi-character chunks as individual symbols. Such an extension is not a theoretical complication but is beyond the scope of this work.

The remainder of this chapter first describes the character-level error analysis for the unconstrained text entry paradigm from prior work (MacKenzie and Soukoreff 2002a) on which this work directly builds. The prior character-level analysis examines only *P* and *T*. Then this chapter describes the extensions to this work that make it suitable for input streams. A taxonomy of input stream errors types is presented, including the assumptions underlying each, and algorithms for their detection are given. Finally, two analyses of real experimental data are conducted, one with the prior analysis of just *P* and *T*, and one with the current analysis using *P*, *T*, and *IS*. With the current analysis and the software that implements it, designers and evaluators stand to benefit from richer character-level error information which, in turn, will result in more refined text entry techniques from rigorous evaluations.

11.2 A Prior Analysis of *P* and *T*

The current work builds on a prior character-level analysis of just *P* and *T* (MacKenzie and Soukoreff 2002a). This technique must first be understood before describing the current work. Thus, a brief overview of that technique is presented. Readers are directed to the prior work for more details.

The prior analysis began by asking, "How many errors are in the following transcription?"

```
P: quickly
T: qucehkly
```

Figure 11.3. The example used in the original analysis (MacKenzie and Soukoreff 2002a). This example is used for continuity.

A pair-wise comparison of the letters in Figure 11.3 suggests that all letters after the "qu" are in error. But intuition tells us that the "kly" at the end seems correct. How can we determine the correct number of errors?

The answer lies in using the MSD statistic (§2.4.2). Recall that this statistic tells us the shortest "distance" between two strings, which can be characterized as the minimum number of errors between them. This is also equivalent to the minimum number of simple editing operations, so-called "Morgan's operations" (Morgan 1970), required to turn one string into the other.

In Figure 11.3, MSD = 3. This means that there are a minimum of 3 errors in *T* relative to *P*. It also means that in no less than 3 simple editing operations can we make "quickly" and "qucehkly" into equivalent strings. The error types are:

- *Insertion* — occurs when a letter appears in *T* but not in *P*. For example, if *P* is "cat" and *T* is "cart", we have an insertion for "r". We can represent an insertion by placing a hyphen "-" in *P* where the insertion appears in *T*. For example, we would write "cat" as "ca-t".

- *Omission* — occurs when a letter appears in *P* but is omitted from *T*. For example, if *P* is "cat" and *T* is "ct", we have an omission of "a". We can represent an omission by placing a hyphen in *T* where the omitted letter appears in *P*. For example, we would write "ct" as "c-t". The prior analysis termed these errors "deletions" (MacKenzie and Soukoreff 2002a), but "omission" will be used here as in earlier papers (Gentner *et al.* 1984).

- *Substitution* — occurs when corresponding letters in *P* and *T* do not agree. For example, if *P* is "cat" and *T* is "kat", there is a substitution of "k" for "c".

As stated, MSD = 3 for our example. But although we know *that* 3 errors exist between *P* and *T*, and therefore 3 operations are necessary to equate them, we do not know *which* 3 errors occurred, or *which* 3 operations reflect the subject's intentions. For example, did the subject substitute the "c" for the "i" in Figure 11.3 or did he or she omit the "i" and correctly enter the "c"?

To handle this ambiguity, we first generate all possible operation sets of cardinality 3 that make *P* and *T* equivalent. These are called the *optimal alignments* of *P* and *T* (MacKenzie and Soukoreff 2002a). There are 4 optimal alignments for our example in Figure 11.3:

```
P1: qu-ickly
T1: qucehkly

P2: qui-ckly
T2: qucehkly

P3: quic-kly
T3: qucehkly

P4: quic--kly
T4: qu-cehkly
```

Figure 11.4. The optimal alignments of "quickly" and "qucehkly" (MacKenzie and Soukoreff 2002a).

To detect errors after identifying the optimal alignments, we simply move through the (P_n, T_n) pairs comparing letters in a pair-wise fashion at each position. If two letters agree, a *no-error* is the result. If two letters disagree, a *substitution* occurred. If a dash appears in *P*, an *insertion* occurred. If a dash appears in *T*, an *omission* occurred.

Ambiguity is handled by weighting each error by the inverse of the number of alignments. For example, in the 4 alignments above, each substitution for "i" is tallied as 1 × 0.25. Summed together, we get 0.75 as the substitution error rate for "i". In other words, we say there is a 75% chance that the user committed a substitution for "i" while entering "qucehkly". Accordingly, there is a 25% chance that the user omitted "i", as seen in the fourth alignment.

The main limitation of this prior analysis is that it ignores all erased characters. In most unconstrained text entry experiments, corrected errors greatly outnumber uncorrected errors. This means that much richer error data is available to us if we include corrected errors in our analyses. This is particularly true for character-level errors, since uncorrected character-level errors may be rare for any given character. Including corrected character-level errors allows us to see which characters really are error-prone during entry.

11.3 Making Sense of Input Streams

The analysis of input streams yields new types of errors. It also entails new complexities due to ambiguity. This section describes both.

11.3.1 Input Stream Error Types

When we move from an analysis of just *P* and *T* to an analysis involving *IS*, new types of errors arise. These new errors provide more detail about the text entry process and give us a more powerful scope under which to view errors. This section presents a taxonomy of input stream error types with relevant examples. The assumptions used in this section are made explicit in the section that follows (§11.3.2).

11.3.1.1 Uncorrected No-Errors, Substitutions, Insertions, Omissions

Hereafter, Gentner *et al.*'s errors (Gentner *et al.* 1984) are prefaced with the term "uncorrected" to indicate that these errors remain in the transcribed string. We therefore have *uncorrected no-errors*, *uncorrected substitutions*, *uncorrected insertions*, and *uncorrected omissions*. The definitions of uncorrected errors remain unchanged from the prior work (Gentner *et al.* 1984, MacKenzie and Soukoreff 2002a).

11.3.1.2 Corrected No-Errors

Correct characters are often erased in the text entry process, particularly when touch-typing (Soukoreff and MacKenzie 2004). These correct-but-erased characters are called *corrected no-errors*. When combined with uncorrected no-errors, they compose the set of correctly entered characters.

11.3.1.3 Corrected Substitutions

Consider the following input stream, where "←" is a backspace.

```
 P: quickly
IS: qv←w←uickly
```

Figure 11.5. An input stream showing difficulty entering “u”.

The input stream in Figure 11.5 shows a correctly entered “q” but apparent trouble producing the following “u”. The subject first entered, and then backspaced, a “v” and a “w” before correctly entering the “u”. This might be because “u”, “v”, and “w” are keys too close together on a mobile device’s keyboard, or because a user easily confused strokes for “u”, “v”, and “w” in a stroke-based method. For example, “u”-“v” confusion is common for novices when writing Graffiti.

In Figure 11.5, we term “v” and “w” *corrected substitutions* for “u”. That’s because if either “v” or “w” (but not both) were left in lieu of “u”, we would have had an *un*corrected substitution for “u”. Note that the term “corrected” refers to the fact that “v” and “w” were backspaced, not to the fact that “u” correctly ended up in *T*. For example, “v” and “w” are corrected substitutions for “u” in Figure 11.6 despite an “x” persisting as an *un*corrected substitution for “u”:

```
 P: quickly
IS: qv←w←xickly
```

Figure 11.6. The “v” and “w” are corrected substitutions for “u”, while the “x” is an uncorrected substitution for “u”.

Subjects in text entry experiments often go a few letters past erroneous entries before noticing their errors and backspacing to fix them (Soukoreff and MacKenzie 2004). This can result in an input stream like the following:

```
 P: quickly
IS: qvlck←←←←uickly
```

Figure 11.7. The “vl” is erroneous but the subject did not correct it until after correctly entering the first “ck”.

In Figure 11.7, the subject may not have spotted the erroneous “vl” until after the first “ck”. The “v” should be classified as a corrected substitution for “u”, the “l” as a corrected substitution for “i”, and the erased “ck” as corrected no-errors.

11.3.1.4 Non-recognition Substitutions

A *non-recognition substitution* occurs when an attempt to produce a character yields no result. Although non-recognitions are more applicable to stroke-based entry than to keys or buttons, virtual keyboards may regard clicks or taps that land in the "dead space" between keys or along their margins as non-recognitions, since these are also futile attempts to produce characters. Thus, non-recognitions are applicable to methods beyond those that use recognizers. Indeed, any method in which an attempt to produce a character can produce nothing is relevant.

Non-recognitions can be represented in the input stream as "ø". Consider the input in Figure 11.8.

```
 P: quickly
IS: qøuickly
```

Figure 11.8. The input stream contains a non-recognition "ø".

In this example, the first attempt at "u" produced no actual character ("ø"). The second attempt produced a "u", and the user proceeded correctly thereafter. Note that no backspace is required to remove a non-recognition since it does not represent a printed character.

There are various ways to add non-recognitions to input streams (*IS*). One way is for a text entry technique to send a special non-printable character code to be trapped by the user test software and logged directly as a non-recognition. This approach may be called "explicit non-recognition handling." A second approach is for a log file analyzer to infer a non-recognition when it sees that a stroke (or other effort) was begun and ended but did not produce a character. This approach can be called "implicit non-recognition handling." As described below (§11.4.3), the test software supports both methods.

11.3.1.5 Corrected Insertions

Insertions are extra characters in *T* or *IS* lacking a corresponding character in *P*. Consider the input in Figure 11.9.

```
 P: quickly
IS: qxui←←←uickly
```

Figure 11.9. The "x" in the input stream is a corrected insertion.

Here, it seems the subject inserted an “x” before the first “u”, but later noticed the “x” and erased it. If the “x” were not erased, it would have resulted in an *un*corrected insertion. As it is, it is a *corrected insertion.* Note that this classification is independent of the fact that the second “u” is ultimately transcribed in *T*. It is *not* independent, however, of the fact that a “u” immediately follows the inserted “x”. The analysis relies on this fact to determine that the “x” was inserted and not an attempted “u”, and the “u” an attempted “i”, which is possible but not likely.

Note that in Figure 11.9, the first “u” and “i” are treated as errors by aggregate measures that only count backspaces in the input stream (Soukoreff and MacKenzie 2003). But clearly, the first “u” and “i” are correct, and the current analysis treats them that way, classifying them as corrected no-errors.

A second type of corrected insertion is when characters are entered beyond the length of *P*. Figure 11.10 shows an example of this:

 P: quickly

IS: **quicklx**a←

Figure 11.10. The input stream shows an “a” inserted beyond the corresponding length of the presented string.

In this example, it seems the “a” was inserted at the end of *IS* and then erased. Since all letters in *P* and *IS* are already paired, the “a” is deemed a corrected insertion.

A third type of corrected insertion is when we have two identical letters in a row, and the first one is correct but the second one is incorrect. Consider the following:

 P: speech

IS: **spee**e←**ch**

Figure 11.11. The third “e” in the input stream could be the result of an accidental doubling of the correct “e” before it.

In this example, it seems likely that the third “e” is not an attempt at “c” but an accidental double-entry of the previous “e”. This type of corrected insertion is probably more common to keypad or keyboard entry than stroke-based entry, since double-entries can occur when physical buttons are pressed too firmly, held down too long, or because the key-repeat rate is too fast. Double-entries sometimes occur with a stylus on a virtual keyboard for subjects with tremor (Wobbrock *et al.* 2003b). Input techniques can be "debounced" to help protect against these kinds of insertion errors.

One requirement for this type of corrected insertion is that the character prior to the potential double-entry is correct (the second "e" in Figure 11.11). This increases our confidence that the character under consideration (the third "e") is indeed a double-entry. Consider:

```
 P: speech
IS: spedd←←ech
```

Figure 11.12. The "d"'s are both deemed corrected substitutions. The second "d" is not a corrected insertion because the "d" before it is an error.

The input in Figure 11.12 should probably not be treated as having a corrected insertion for the second "d" because the first "d" is itself erroneous as a corrected substitution for "e". In this case, the second 'd" is treated as a corrected substitution for "c".

11.3.1.6 Non-recognition Insertions

Only the second of the three types of corrected insertion, those caused by entering letters beyond the length of *P* (Figure 11.10), applies to *non-recognition insertions*. For example:

```
 P: cat
IS: catøø
```

Figure 11.13. The input stream shows two non-recognition insertions.

The input stream in Figure 11.13 shows two non-recognition insertions, both of which occur after each letter in *P* is already paired with a letter in *IS*.

11.3.1.7 Corrected Omissions

Omissions occur when characters in *P* are skipped in *T* or *IS*. *Corrected omissions* occur when a character in *P* is initially skipped but then later replaced, thus remedying the omission. Figure 11.14 shows an example.

```
 P: quickly
IS: quikl←←ckly
```

Figure 11.14. The "c" is initially omitted, resulting in a corrected omission.

It seems the "c" is initially skipped but later replaced, resulting in a corrected omission. (Corrected no-errors must be tallied for the first "kl".) Note that the classification of "c" as a corrected omission does not depend on a "c" ultimately being transcribed. For example:

```
 P: quickly
IS: quikl←←xkly
```

Figure 11.15. The "c" is initially omitted, resulting in a corrected omission, even though an "x" takes its place as an uncorrected substitution for "c".

The input in Figure 11.15 seems to contain a corrected omission for "c", because "c" was initially omitted. This classification depends on the correctly entered "k" immediately following the "i". If the "k" were another letter, we would have no reason to believe that the letter was not an attempted "c"—and therefore a corrected substitution. Note that the transcribed "x" results in an uncorrected substitution for "c".

11.3.2 Assumptions for Resolving Ambiguity

Section 11.3.1 made some implicit assumptions to resolve ambiguity. This section makes those assumptions explicit, arguing for their reasonableness and necessity if value is to be extracted from input streams at the level of individual characters.

11.3.2.1 Subjects Proceed Sequentially Through P

The first assumption is that subjects proceed sequentially through *P* as they enter *IS*. This assumption is the bedrock on which this analysis is built. Consider the following:

```
 P: cats
IS: cuf←←ats
```

Figure 11.16. Our first assumption says that subjects proceed sequentially through the presented string, so "uf" in *IS* is matched with "at" in *P*.

Our first assumption pairs "uf" with "at" in Figure 11.16. Without this assumption, we would have to allow that any letter in *IS* could be paired with any letter in *P*, or none in *P* at all! This assumption is reasonable given the nature of text entry experiments in which subjects are instructed to sequentially transcribe presented strings.

Now consider the following:

P: cat
IS: **c**x←**at**

Figure 11.17. Was the "x" an attempted "a" or merely a corrected insertion? Our first assumption favors the former.

In this example, it seems the subject entered an "x" while attempting an "a". Are we sure? What if the "x" was not an attempted "a" but an insertion that was promptly corrected? In other words, do we have a corrected substitution of "x" for "a" or a corrected insertion of "x"? Our first assumption favors the former: "x" is paired with "a". Prior character-level evidence shows that substitutions are usually much more common than insertions (MacKenzie and Soukoreff 2002a).

However, this assumption does not prevent us from detecting corrected insertions and corrected omissions. Consider:

P_1: cat
IS_1: **c**xa←←**at**

P_2: cat
IS_2: **c**t←**at**

Figure 11.18. These two examples show a corrected insertion and a corrected omission, respectively.

In the first pair, was "x" an attempted "a" and "a" an attempted "t"? Or was the "x" simply inserted? Because the first "a" in IS_1 matches the "a" in P_1, we favor the latter and report a corrected insertion for "x" and a corrected no-error for "a".

In the second pair, was "t" an attempted "a" or was "a" omitted and then promptly added? Since the first "t" in IS_2 matches the letter after the "a" in P_2, namely the "t", we favor the latter and report a corrected omission for "a" and a corrected no-error for the first "t" in IS_2.

11.3.2.2 Subjects Insert or Omit Only One Character in a Row

The second assumption states that subjects insert or omit only one character at a time. Consider these variations on the input streams from Figure 11.18:

P_1: cats
IS_1: **c**xfa←←←**ats**

P_2: cats
IS_2: **c**s←**ats**

Figure 11.19. The second assumption allows for only one corrected insertion or omission in a row. Thus, these examples contain only corrected substitutions.

In the first pair, we could regard both "x" and "f" in IS_1 as corrected insertions because they are followed by an "a". But then, at how many consecutive insertions do we draw the line? For tractability, we chose to draw it at one and deem "xfa" as an attempted "ats" and report them all as corrected substitutions.

In the second pair, we might regard both "a" and "t' in P_2 as corrected omissions. But again, at how many consecutive omissions do we draw the line? We chose to allow one sequential omission, since subjects are instructed to progress sequentially through *P* and skipping multiple letters is discouraged and uncommon in practice.

The assumptions so far allow us to avoid considering unlikely possibilities, such as:

P: cats
IS: **c**xfs←←←**ats**

Figure 11.20. The second assumption keeps us from treating the "at" in *P* as initially omitted and the "xf" in *IS* as initially inserted.

The limitation of one insertion or omission in a row prevents us from treating "at" in *P* as omissions and "xf" in *IS* as insertions. Instead, we simply have corrected substitutions of "xf" for "at", which are much more likely and straightforward.

11.3.2.3 Backspaces Are Made Accurately and Intentionally

The third assumption is that backspaces are made both accurately and intentionally. Backspace ("←") is fundamental to text entry. For this reason, keyboards usually make backspace larger than other keys and unistroke alphabets like Graffiti and EdgeWrite assign simple straight line strokes to backspace. Designers are motivated to make backspace quick and accurate due to its frequency and importance. As previously stated, backspace has been shown to be the second most commonly typed desktop key after the spacebar (MacKenzie and Soukoreff 2002b).

The current analysis assumes that backspaces are made *accurately*; that is, an attempted "←" results in a "←". Since we do not have any backspaces in *P* to compare to those in *IS*, there is no implied intention on which we can rely. Accommodating the possibility of erroneous backspaces greatly complicates the analysis and is an ambitious topic for future work.

Clearly, not all backspaces are going to be made accurately in a large text entry experiment. Consider these cases:

P : cat

IS_1: **c**xø←**at**

IS_2: **c**xz←←**at**

Figure 11.21. The third assumption asserts that backspaces are made accurately, but "ø←" and "←←" patterns may indicate exceptions.

In *IS_1*, we regard the non-recognition ("ø") as an attempted "t", after which the subject notices the erroneous "x" and erases it. But what if the non-recognition was a failed first attempt at backspace? It is difficult to say, since subjects often notice errors only after they've gone past them (Soukoreff and MacKenzie 2004), and "ø" could have been an attempted letter.

Similarly, in *IS_2* we regard the "z" as an attempted "t", after which the subject notices the erroneous "x" and erases "xz". But what if the "z" was an attempted backspace to begin with? Again, there is no way to know, but the "←←" pattern may indicate a failed attempt at backspace. But then again, subjects often move a few letters past an error before noticing the error and backspacing to fix it, so perhaps the "z" was an attempted "t" after all. There is simply no way to know, so we rely on our assumption.

The third assumption also says that backspaces are made *intentionally*; that is, an attempt at something other than "←" does not result in a "←". Consider the following:

P: cat

IS: **ca**a←**at**

Figure 11.22. The third assumption asserts that backspaces are made intentionally, but "*x*←…←*x*" patterns may indicate exceptions.

Perhaps the subject initially thought the "a" was in error. Or perhaps the backspace was an attempted "t", which erased the already correct "a", and therefore the "a" had to be replaced. Under this assumption, we treat the backspace as intentional. Corrected no-errors like the first "a" in Figure 11.22 can occur when subjects enter text quickly, since they will occasionally anticipate an error and enter a backspace even when their entry turns out to have been correct.

To assess the feasibility of these assumptions about backspace for a stroke-based method, the logs from the study of Stylus EdgeWrite in §4.3.1 were manually examined. If backspaces were error-prone, we should see many "ø←" and "←←" patterns in the input streams. If backspaces were unintentional, we should see many "*x*←…←*x*" patterns, where *x* is a correct entry and "…" are optional intervening characters.

Together, subjects in the study attempted 4932 characters. Of these, 4660 produced characters and 272 were non-recognitions. Of the 4660 entered characters, 4207 were alphanumeric and 453 were backspaces. The sequence "ø←" occurred 55 times, and "←←" occurred 38 times. In scrutinizing the strokes by hand, however, it was clear that only 8 "ø←" were due to unrecognized attempts at backspace. Similarly, only 2 "←←" were due to misrecognized backspaces. Thus, only 10/453, or 2.21%, of attempted backspaces appeared to violate the first part of our assumption.

Because recognition-based methods are often less accurate than selection-based methods during entry (Költringer and Grechenig 2004), the first part of our third assumption is likely to hold even more reliably for stylus keyboards, on-screen keyboards, mini-QWERTY keyboards, and others, since backspace is selected and is not the result of a potentially sloppy stroke.

The sequence "*x*←…←*x*", where *x* is a correct entry, was observed 6 times. This means only $\frac{6}{4207-2+6}$, or 0.14%, of attempted alphanumeric characters were unintentional backspaces,[17] supporting the reasonableness of the second part of the third assumption.

11.3.2.4 Omissions in T Are Also Omitted in IS

The fourth assumption says that letters skipped in *T* are also skipped in *IS*. Consider the following:

[17] The denominator is the total number of attempted alphanumerics: the total number of produced alphanumerics (4207) minus those that were attempted backspaces (2) plus the backspaces that were meant to be alphanumerics (6).

```
 P: cats
IS: cax←y←z
 T: caz
```

Figure 11.23. This input is ambiguous as to whether the "z" should be aligned with "t" or "s". Both possibilities are represented in the optimal alignment set (Figure 11.4) and are weighted accordingly.

The transcription *T* for this input aligns with *P* as both "caz-" and "ca-z". Two alignments are needed because we cannot be sure whether "z" was an attempted "t" or an attempted "s". It may be that the subject was attempting "t" all along and forgot the "s" at the end. Or it may be that the subject did not see the "t" and was attempting the "s". As discussed above (§11.2), weighting by the number of alignments accommodates this uncertainty. Thus, in the first alignment ("caz-"), "x" and "y" are treated as attempts at "t"; in the second ("ca-z"), they are treated as attempts at "s".

On the other hand, what if we had an "s" in the final transcription instead of a "z"? This is shown in the following example:

```
 P: cats
IS: cax←y←s
 T: cas
```

Figure 11.24. Our fourth assumption treats the "x" and "y" as attempts at "s".

In this case, there is only one optimal alignment of *T* with *P*, "ca-s", which contains an uncorrected omission for "t". Because the "t" was skipped, we assume the "x" and "y" were not attempts at "t", but attempts at "s". After all, it was with the entry of an "s" that the subject was satisfied to leave a character in that position. To assume otherwise is to allow that "x" was perhaps first an attempt at "t", and that "y" was another attempt at "t"—or perhaps suddenly an attempt at "s". We would have to assume that subjects left nothing to show for their attempts at "t" and became misaligned as a result.

In our experience over many text entry studies, subjects are rarely this capricious. Numerous studies have shown that regardless of the text entry method being used, subjects try hard to stay aligned with *P* while transcribing *T*. If they fail to obtain a correct letter after many tries, they often leave their final incorrect attempt and proceed, thereby remaining aligned with *P*, rather than erasing their final attempt and becoming misaligned thereafter. In other words, if the "y" in Figure 11.24 was indeed an attempt at "t", it is unlikely that subjects would first erase it before attempting the "s", because this throws them out of alignment and their “s” would appear below the “t”. If indeed “y” was a failed attempt at “t”, subjects will usually leave it and simply try the “s”. In short, subjects eschew gaps and embrace alignment.

One complication with this assumption, however, is when subjects *overshoot* with backspace and neglect to replace letters. For example:

```
 P: cats
IS: caf←←ts
 T: cts
```

Figure 11.25. The subject may have overshot with backspace past the erroneous “f” and through the correct “a”, which he then neglected to replace.

Figure 11.25 shows the erasure of an erroneous “f” and a correct “a”. Nothing is replaced for the “a” that was accidentally erased, so the aligned transcription is “c-ts”, which contains an uncorrected omission for “a”. Since the subject omitted “a” in *T*, it is assumed they omitted it in *IS*, but that is not true here. This complication arises because over backspacing without replacement injects ambiguity into discerning intention. Fortunately, over backspacing without replacement is relatively uncommon, and is discouraged by well designed user test software that uses fixed-width fonts for *P* and *T* and displays *P* above *T* in close proximity. Manual inspection of data from the Stylus EdgeWrite study (§4.3.1) revealed that only 2/453, or 0.44%, of backspaces were overshoots. And in both cases, the letter erased by overshooting *was* replaced, meaning this assumption held for all 4207 characters. Still, relaxing this assumption is a topic for future work.

11.4 Error Detection and Classification

The following algorithm automates the detection and classification of the 10 error types described above (§11.3.1). Automatically-generated error reports can aid designers and evaluators in improving text entry techniques by revealing troublesome characters.

11.4.1 Algorithm Walkthrough Step by Step

The algorithm from prior work compared *P* and *T* (MacKenzie and Soukoreff 2002a). The current algorithm builds on this work and compares *P* and *IS*. The pseudocode enables technique designers and evaluators to incorporate this analysis into their own work. In the code, both space and letters are referred to as just “letters” and backspace is the only error correction operation. In **for** loops, the **to** keyword includes the upper-bound. For readability, string indices are not bounds-checked, and the comparison of $A[i]$ to $B[j]$ is taken to be **false** if either index i or j is beyond the end of its respective string A or B.[18] The notation $|S|$ means the length of string S or the cardinality of set S. The "⇐" symbol means "assign" while the symbol "$\overset{+}{\Leftarrow}$" means "append to string" or "add to set." The style of this pseudocode is based on that of a popular algorithms book (Cormen *et al.* 1990).

11.4.1.1 Flag the Input Stream

In the first step, we flag the transcribed letters in the input stream, i.e., we flag the letters in *IS* that compose *T*. We do this with a backward pass over *IS*, incrementing a counter after passing "←", decrementing it (but not below zero) after passing a letter, and leaving it unchanged after passing a non-recognition ("ø"). We flag letters for which the counter is zero *before* decrementing. Figure 11.26 shows a hypothetical input stream for *T* = "qucehkly" from Figure 11.3. Flagged letters are bold:

count: 12100010001000121000010

IS: pv←←**qu**c←**c**øk←**eh**ly←←**kly**z←

Figure 11.26. The first step is to flag the letters in the input stream that compose the transcribed string using a backward pass.

The pseudocode for this step is given in Figure 11.27.

```
FLAG-STREAM(IS)
    $count \Leftarrow 0$
    for $i \Leftarrow |IS| - 1$ to 0 do
      if $IS[i]$ = '←' then
        $count \Leftarrow count + 1$
      else if IS-LETTER($IS[i]$) then
        if $count = 0$ then FLAG($IS[i]$)
        else $count \Leftarrow count - 1$
    return IS
```

Figure 11.27. Algorithm for "flagging" the characters in *IS* that compose *T*.

11.4.1.2 Compute the MSD Matrix

The second step is to compute the minimum string distance (MSD) matrix. The MSD algorithm fills a matrix of integers as it determines the minimum number of operations

[18] Where necessary, one can add checks so that tests fail if an index is out of bounds. Thus, **if** ($A[i] = B[j]$) expands to **if** ($i < |A|$ **and** $j < |B|$ **and** $A[i] = B[j]$).

required to equate two strings. A prior version of the algorithm (Soukoreff and MacKenzie 2001) is adjusted to return not just the string distance but also the filled MSD matrix, which we will use in the next step. Readers wishing further details on this algorithm are directed to that paper or earlier ones (Levenshtein 1965, Wagner and Fischer 1974).

```
MSD-MATRIX($P$, $T$)
    $D \Leftarrow$ new matrix of dimensions $|P| + 1, |T| + 1$
    for $i \Leftarrow 0$ to $|P|$ do
      $D[i, 0] \Leftarrow i$
    for $j \Leftarrow 0$ to $|T|$ do
      $D[0, j] \Leftarrow j$
    for $i \Leftarrow 1$ to $|P|$ do
      for $j \Leftarrow 1$ to $|T|$ do
        $D[i, j] \Leftarrow$ MIN($D[i - 1, j] + 1$,
                         $D[i, j - 1] + 1$,
                         $D[i - 1, j - 1] + P[i - 1] \neq T[j - 1]$)
    return $D[|P|, |T|]$ and $D$
```

Figure 11.28. Algorithm for computing the minimum string distance. In this case, the ≠ comparison returns integer ‘1’ if **true** and integer ‘0’ if **false**.

11.4.1.3 Compute the Set of Optimal Alignments

The third step is to compute the set of optimal alignments of *P* and *T* (see Figure 11.4). In Figure 11.29, *D* is the MSD matrix, and *x* and *y* are initialized with the lengths of *P* and *T*, respectively. *P’* and *T’* are initially empty strings and the “+” operation on them is string concatenation. For more information, readers are directed to prior work (MacKenzie and Soukoreff 2002a).

```
ALIGN($P$, $T$, $D$, $x$, $y$, $P'$, $T'$, ref alignments)
    if $x = 0$ and $y = 0$ then
      alignments $\Leftarrow^{+}$ ($P'$, $T'$)  // add a new aligned pair
      return
    if $x > 0$ and $y > 0$ then
      if $D[x, y] = D[x - 1, y - 1]$ and $P[x - 1] = T[y - 1]$ then
        ALIGN($P$, $T$, $D$, $x - 1$, $y - 1$, $P[x - 1] + P'$, $T[y - 1] + T'$)
      if $D[x, y] = D[x - 1, y - 1] + 1$ then
        ALIGN($P$, $T$, $D$, $x - 1$, $y - 1$, $P[x - 1] + P'$, $T[y - 1] + T'$)
    if $x > 0$ and $D[x, y] = D[x - 1, y] + 1$ then
      ALIGN($P$, $T$, $D$, $x - 1$, $y$, $P[x - 1] + P'$, “-” + $T'$)
    if $y > 0$ and $D[x, y] = D[x, y - 1] + 1$ then
      ALIGN($P$, $T$, $D$, $x$, $y - 1$, “-” + $P'$, $T[y - 1] + T'$)
```

Figure 11.29. Algorithm for computing the optimal alignments of *P* and *T*. Reproduced with permission (MacKenzie and Soukoreff 2002a).

11.4.1.4 Stream-Align IS with P and T

In the fourth step, we add a copy of *IS*, each still "flagged," to each optimal alignment pair computed by ALIGN, thereby forming optimal alignment triplets. We then *stream-align* the triplets so that *P*, *T*, and *IS* are all aligned. We do this by aligning the flagged letters in *IS* with their correspondents in *P* and *T*. The last alignment from Figure 11.4 looks like this when stream-aligned:

```
 P4: ____qui__c___--____kly__
 T4: ____qu-__c___eh____kly__
IS4: pv←←qu_c←cøk←ehly←←klyz←
```

Figure 11.30. A stream-aligned triplet of (*P*, *T*, *IS*). This is the fourth of the optimal alignments from Figure 11.4.

The triplet in Figure 11.30 uses underscore spacers ("_") in *P* and *T* where flagged letters in *IS* are absent, and in *IS* where uncorrected omissions are in *T* (i.e., where "-" symbols appear in *T*). Figure 11.31 gives the pseudocode for stream-aligning *P*, *T*, and *IS*.

```
STREAM-ALIGN(IS, alignments)
foreach aligned pair <P', T'> in alignments do
  IS' ⇐ COPY(IS)
  for i ⇐ 0 to MAX(|T'|, |IS'|) – 1 do
    if T'[i] = '-' then
      INSERT('_', IS'[i])
    else if not IS-FLAGGED(IS'[i]) then
      INSERT('_', P'[i])
      INSERT('_', T'[i])
  triplets ⇐+ (P', T', IS')  // add a new aligned triplet
return triplets
```

Figure 11.31. Algorithm for aligning *P*, *T*, and *IS*.

11.4.1.5 Assign Position Values to Characters in the Input Stream

The fifth step is to assign "position values" to each character in *IS*. Position values help determine the intended letter in *P* for each letter in *IS*. Flagged letters always receive a position value of zero. (Recall that flagged letters are those in *IS* that compose *T*. Unflagged characters are erased letters, backspaces, non-recognitions, and underscore spacers.) Within each unflagged substring between two flags in *IS*, a letter's position value is the substring index it *would* have had if that substring were transcribed to that point. A backspace's position value, by contrast, is the position value of the letter that the backspace erases. The position values for our example are shown in Figure 11.32.

```
      011000000000000011000000
IS₄: pv←←qu_c←cøk←ehly←←klyz←
```

Figure 11.32. Position values are shown atop characters in the input stream. They are assigned using a forward pass.

The algorithm for assigning position values is shown in Figure 11.33. The function receives the set of *triplets* from STREAM-ALIGN.

```
ASSIGN-POSITION-VALUES(ref triplets)
foreach aligned triplet <P, T, IS> in triplets do
  pos ⇐ 0
  for i ⇐ 0 to |IS| – 1 do
    if IS-FLAGGED(IS[i]) then
      SET-POSITION-VALUE(IS[i], 0)
      pos ⇐ 0
    else
      if IS[i] = '←' and pos > 0 then
        pos ⇐ pos – 1
      SET-POSITION-VALUE(IS[i], pos)
      if IS-LETTER(IS[i]) then
        pos ⇐ pos + 1
```

Figure 11.33. Algorithm for assigning position values to characters in *IS*. Position values help determine the intended letter in *P* for each letter in *IS*.

11.4.1.6 Proceed Through IS to Detect and Classify Errors

The sixth and final step of the algorithm proceeds through each *IS* in each stream-aligned triplet and classifies each input stream character. The procedure takes successive substrings between flagged letters in *IS*, where the substrings are bound by a flagged character on their right but not on their left. Thus, the first three substrings of IS_4 from Figure 11.30 are "pv←←**q**", "**u**", and "_c←**c**". Letters within a substring are then compared to corresponding letters in *P*. These "corresponding letters" are determined using the position values just assigned in Figure 11.33. The comparisons performed for the fourth stream-alignment (Figure 11.30) are illustrated in Figure 11.34. Note that this processing is applied to each triplet returned by STREAM-ALIGN, not just this one.

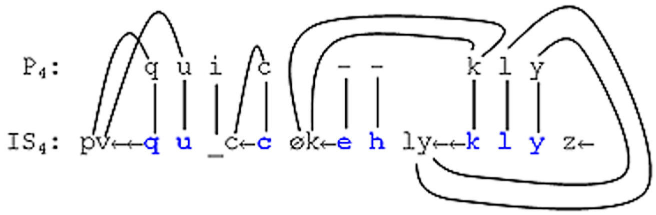


Figure 11.34. Comparisons made for the fourth stream-alignment (Figure 11.30).

```
DETERMINE-ERRORS(triplets)
foreach aligned triplet <P, T, IS> in triplets do
  $a \Leftarrow 0$
  for $b \Leftarrow 0$ to $|IS| - 1$ do
    if $T[b]$ = '-' then
      UNCORRECTED-OMISSION($P[b]$)
    else if IS-FLAGGED($IS[b]$) or $b = |IS| - 1$ then
      $M \Leftarrow$ new integer set  // corrected omissions
      $I \Leftarrow$ new integer set  // corrected insertions
      for $i \Leftarrow a$ to $b - 1$ do  // iterate over a substring between flags
        $v \Leftarrow$ GET-POSITION-VALUE($IS[i]$)
        if $IS[i]$ = '←' then
          if CONTAINS($M$, $v$) then REMOVE($M$, $v$)
          if CONTAINS($I$, $v$) then REMOVE($I$, $v$)
        else if $IS[i] \neq$ '_' then
          $target \Leftarrow$ LOOK-AHEAD($P$, $b$, $v + |M| - |I|$, IS-LETTER)
          if $IS[i]$ = 'ø' then
            if $target \geq |P|$ then NONREC-INSERTION('ø')
            else NONREC-SUBSTITUTION($P[target]$, 'ø')
          else  // $IS[i]$ is a letter
            $next_P \Leftarrow$ LOOK-AHEAD($P$, $target$, 1, IS-LETTER)
            $prev_P \Leftarrow$ LOOK-BEHIND($P$, $target$, 1, IS-LETTER)
            $next_{IS} \Leftarrow$ LOOK-AHEAD($IS$, $i$, 1, IS-NOT('ø', '_'))
            $prev_{IS} \Leftarrow$ LOOK-BEHIND($IS$, $i$, 1, IS-NOT('_'))
            if $IS[i] = P[target]$ then
              CORRECTED-NOERROR($IS[i]$)
            else if $target \geq |P|$ or $IS[next_{IS}] = P[target]$
                  or ($IS[prev_{IS}] = IS[i]$ and $IS[prev_{IS}] = P[prev_P]$) then
              CORRECTED-INSERTION($IS[i]$)
              $I \Leftarrow^{+} v$  // track this corrected insertion
            else if $IS[i] = P[next_P]$ and IS-LETTER($T[target]$) then
              CORRECTED-OMISSION($P[target]$)
              CORRECTED-NOERROR($IS[i]$)
              $M \Leftarrow^{+} v$  // track this corrected omission
            else CORRECTED-SUBSTITUTION($P[target]$, $IS[i]$)
      end for // $i$ from $a...b - 1$
      if $P[b]$ = '-' then UNCORRECTED-INSERTION($T[b]$)
      else if $P[b] \neq T[b]$ then UNCORRECTED-SUBSTITUTION($P[b]$, $T[b]$)
      else if $P[b] \neq$ '_' then UNCORRECTED-NOERROR($T[b]$)
      else if $IS[b]$ = 'ø' then NONREC-INSERTION('ø')
      $a \Leftarrow b + 1$  // $a$ starts the next substring
```

Figure 11.35. Algorithm for classifying errors in the input stream. This algorithm finds all errors from previous work (MacKenzie and Soukoreff 2002a) and the new input stream error types described in §11.3.1.

```
Look-Ahead($S$, $start$, $count$, Condition-Fn)
    $index \Leftarrow start$
    while $0 \le index < |S|$ and not Condition-Fn($S[index]$) do
        $index \Leftarrow index + 1$  // proceed until the condition is met
    while $count > 0$ and $index < |S|$ do
        $index \Leftarrow index + 1$
        if $index = |S|$ then break
        else if Condition-Fn($S[index]$) then
            $count \Leftarrow count - 1$
    return $index$
```

Figure 11.36. Procedure used by Determine-Errors (Figure 11.35). This procedure looks forward in string *S* from a zero-based index *start* until a *count* number of Condition-Fns have been satisfied, returning the index of the *count*th successful test. An example is Look-Ahead(*P*, 0, 2, Is-Letter), which would find two letters *ahead* of the first letter in *P* (i.e. the 3rd letter).

```
Look-Behind($S$, $start$, $count$, Condition-Fn)
    $index \Leftarrow start$
    while $0 \le index < |S|$ and not Condition-Fn($S[index]$) do
        $index \Leftarrow index - 1$  // go back until the condition is met
    while $count > 0$ and $index \ge 0$ do
        $index \Leftarrow index - 1$
        if $index < 0$ then break
        else if Condition-Fn($S[index]$) then
            $count \Leftarrow count - 1$
    return $index$
```

Figure 11.37. Procedure used by Determine-Errors (Figure 11.35). This function is analogous to Look-Ahead but operates in the reverse direction.

The intended letter in *P* (i.e. *P*[*target*]) is determined by using the position value at *IS*[*i*]. This value, which in Determine-Errors is stored in the variable *v*, is added to $|M| - |I|$ to determine the *target* in *P*. The set *M* stores the position values of letters that precipitate corrected omissions, and the set *I* stores position values of letters that are corrected insertions. When a backspace ('←') is processed, the sets *M* and *I* are inspected to see if they contain the position value of the backspace; that is, to see if a corrected omission or corrected insertion was just "undone." If the value *v* is found, it is removed from *M* or *I* accordingly. This, in turn, appropriately affects the determination of the *target* index, since *target* is calculated using $v + |M| - |I|$.

Figure 11.38 gives the output of Determine-Errors for the example triplet from Figure 11.30 and Figure 11.34. The output shows 18 results. This compares to only 9 results available from an analysis of just *P* and *T*. Thus, using the input stream has doubled the character-level error data available for analysis. The triplet is reproduced here for convenience.

```
 P4: ____qui__c___--____kly__
 T4: ____qu-__c___eh____kly__
IS4: pv←←qu_c←cøk←ehly←←klyz←

corrected substitution (q, p)
corrected substitution (u, v)
uncorrected no-error (q, q)
uncorrected no-error (u, u)
uncorrected omission (i, -)
corrected no-error (c, c)
uncorrected no-error (c, c)
non-recognition substitution (k, ø)
corrected no-error(k, k)
uncorrected insertion (-, e)
uncorrected insertion (-, h)
corrected omission (k, -)
corrected no-error (l, l)
corrected no-error (y, y)
uncorrected no-error(k, k)
uncorrected no-error (l, l)
uncorrected no-error (y, y)
corrected insertion (-, z)
```

Figure 11.38. Classification output for the triplet from Figure 11.30. Each line can be read as `classification`(*intended character, produced character*).

11.4.2 Character-level Metrics

After processing data from an experiment, we have error tallies for *each letter*. We also have counts: how many times was each letter presented? transcribed? entered? intended? correct? unrecognized? Note that only the first two of these are available from an analysis of just *P* and *T*.

11.4.2.1 Three Error Rates

Each character has three separate error rates: uncorrected, corrected, and total. These are defined as follows for a given character *i*:

$$Uncorrected\ Error\ Rate_i = 1 - \frac{Uncorrected\ NoErrors_i}{Transcribed_i} \tag{11.1}$$

$$Corrected\ Error\ Rate_i = 1 - \frac{Corrected\ NoErrors_i}{Entered_i - Transcribed_i} \tag{11.2}$$

$$Total\ Error\ Rate_i = 1 - \frac{Uncorrected\ NoErrors_i + Corrected\ NoErrors_i}{Entered_i} \tag{11.3}$$

Equation 11.1 answers the question, “of the *i*’s remaining in the transcription, what percent were erroneous?” Equation 11.2 answers, “of the erased *i*’s, what percent were erroneous?” Equation 11.3 answers, “of all entered *i*’s, what percent were erroneous?” Note that a high error rate for Equation 11.2 means that most of the backspaced *i*’s were in fact errors and *should* have been corrected. Conversely, a low rate for Equation 11.2 means that subjects were erasing already-correct *i*’s, which means they were too hasty to backspace.

The total error rate for a letter (Equation 11.3) refers only to actual entries of that letter. It says, “given that an *i* was entered, what are the chances that *i* was correct?” Omissions of *i* are therefore not captured by Equation 11.3. Instead, omissions are handled by their own error rate, described below.

11.4.2.2 Substitutions vs. Intentions

The character-level data allow us to answer the question, “what is the probability of getting *i* when trying for *i*?” For this we take the ratio of substitutions to intentions. The number of times a letter was “intended” is obtained by adding its substitutions and no-errors. These five error types (uncorrected, corrected, and non-recognition substitutions; and uncorrected and corrected no-errors) have intended *target* letters and actual *produced* letters (or non-recognitions).

$$\textit{Uncorrected Substitution Rate}_i = \frac{\textit{Uncorrected Substitutions}_i}{\textit{Intended}_i} \tag{11.4}$$

$$\textit{Corrected Substitution Rate}_i = \frac{\textit{Corrected Substitutions}_i}{\textit{Intended}_i} \tag{11.5}$$

$$\textit{Nonrecognition Substitution Rate}_i = \frac{\textit{Nonrecognition Substitutions}_i}{\textit{Intended}_i} \tag{11.6}$$

$$\textit{Total Substitution Rate}_i = \frac{\begin{array}{l}\textit{Uncorrected Substitutions}_i + \\ \textit{Corrected Substitutions}_i + \\ \textit{Nonrecognition Substitutions}_i\end{array}}{\textit{Intended}_i} \tag{11.7}$$

Equation 11.4 answers the question, “when trying for *i*, what is the probability we produce an uncorrected substitution for *i*?” Similar questions can be asked of corrected substitutions (Equation 11.5) and non-recognition substitutions (Equation 11.6). Equation 11.7 answers, “when trying for *i*, what is the probability that we don’t get *i*?”

11.4.2.3 Omissions vs. Presentations

We can also determine whether or not some letters are prone to omission. For this we take the ratio of omissions to presentations, i.e., the number of times a letter was omitted compared to the number of times it was presented to the subject for transcription.

$$Uncorrected\ Omission\ Rate_i = \frac{Uncorrected\ Omissions_i}{Presented_i} \tag{11.8}$$

$$Corrected\ Omission\ Rate_i = \frac{Corrected\ Omissions_i}{Presented_i} \tag{11.9}$$

$$Total\ Omission\ Rate_i = \frac{Uncorrected\ Omissions_i + Corrected\ Omissions_i}{Presented_i} \tag{11.10}$$

Equations 11.9 and 11.10 are special in that they can be over 100% if a single presented letter is omitted repeatedly. This is a theoretical possibility but subjects are unlikely to exhibit this behavior.

11.4.2.4 Insertions vs. Entries

We can also discover whether or not a letter *i* is prone to insertion. For this we take the ratio of *i*'s insertions to *i*'s entries.

$$Uncorrected\ Insertion\ Rate_i = \frac{Uncorrected\ Insertions_i}{Entered_i} \tag{11.11}$$

$$Corrected\ Insertion\ Rate_i = \frac{Corrected\ Insertions_i}{Entered_i} \tag{11.12}$$

$$Total\ Insertion\ Rate_i = \frac{Uncorrected\ Insertions_i + Corrected\ Insertions_i}{Entered_i} \tag{11.13}$$

11.4.3 TextTest and StreamAnalyzer

To facilitate the automation of the analyses described in this paper, two complementary applications were built. The first is *TextTest* (shown in Figure 2.14), a program designed for conducting text entry evaluations. It works on Microsoft Windows systems with any text entry method, provided the method sends characters through the low-level keyboard input stream. For example, the `SendInput()` and `keybd_event()` Win32 functions or the `SendKeys` methods in C# or VB .NET do this on Windows systems. TextTest can also send or receive characters over TCP for conducting studies on mobile devices.

Recognition-based text entry methods can send special non-printing character codes to TextTest for explicitly logging character-starts, character-ends, and non-recognitions. TextTest randomly presents phrases from a published corpus of 500 (MacKenzie and Soukoreff 2003), or from any custom phrase set, and writes XML log files for easy parsing. TextTest is implemented in C#.

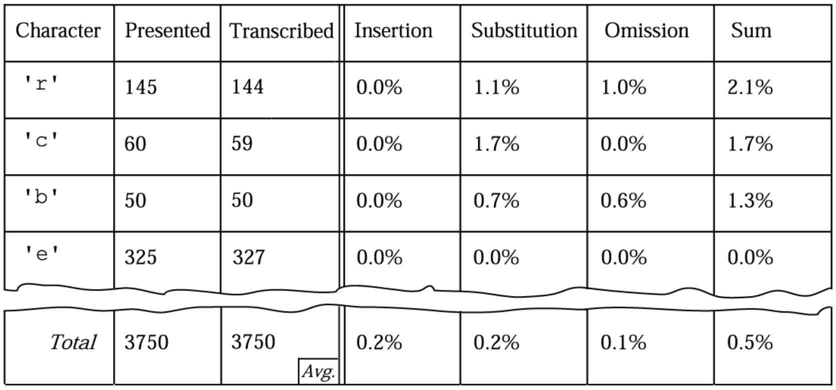
Accompanying TextTest is a log file analyzer named *StreamAnalyzer* that parses TextTest's XML logs and computes various measures of text entry performance. Alternatively, when run with the "-d" switch, StreamAnalyzer can be used to analyze any *P* and *IS* given directly from the console. StreamAnalyzer performs all of the analyses described in this paper and in papers on which this work builds. It writes its output to a space-delimited text file, which can easily be pasted into a spreadsheet for statistical analysis. Like TextTest, the analyzer is written in C#. Both programs have been available for download at http://www.edgewrite.com/dev.html since 2004.

11.5 Comparison of Analyses

To illustrate the advantages of analyzing input streams over just analyzing transcribed strings, both analyses were used on text entry data from the study of Stylus EdgeWrite (§4.3.1) in which 5 first-time novices entered a total of 75 sentences for 3750 presented characters. They made 4932 attempts for 4660 produced characters and 272 non-recognitions. Of the characters, 4207 were "letters" (alphanumerics or spaces) and 453 were backspaces.

11.5.1 Error Rate Tables

Character-level probability tables show error rates for each letter (MacKenzie and Soukoreff 2002a). MacKenzie and Soukoreff's analysis enables the production of tables that list error rates for uncorrected insertions, substitutions, and omissions. Space precludes a full table here, so Table 11.1 shows the letters with the 3 highest error rates and the letter "e" for comparisons.

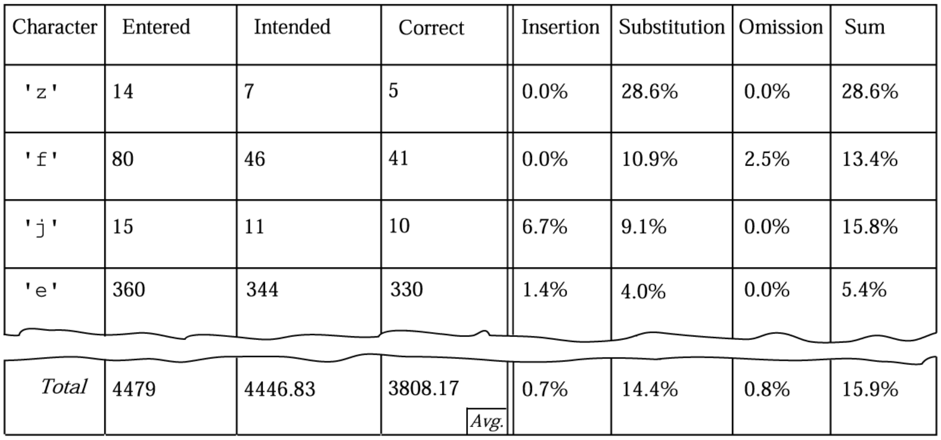


Character	Presented	Transcribed	Insertion	Substitution	Omission	Sum
'r'	145	144	0.0%	1.1%	1.0%	2.1%
'c'	60	59	0.0%	1.7%	0.0%	1.7%
'b'	50	50	0.0%	0.7%	0.6%	1.3%
'e'	325	327	0.0%	0.0%	0.0%	0.0%
Total	3750	3750 *Avg.*	0.2%	0.2%	0.1%	0.5%

Table 11.1. An excerpt of character-level results using MacKenzie and Soukoreff's analysis. The bottom row is for the whole table, not just for the excerpt shown here.

Note the relatively few total uncorrected errors examinable by this analysis (0.5%). In fact, only 8 of the 37 characters (*a–z*, 0–9, space) in the full table have non-zero error rates. Perhaps subjects were careful in correcting errors. Or perhaps they did not make many errors to begin with. With this analysis, there is no way to know.

A much bigger table results from analyses of input streams. The table has a row for each letter and a column for each count and character-level measure. This table, which is automatically generated by StreamAnalyzer, reports that "e" was intended 344 times, entered 360 times, correct 330 times, and unrecognized 8 times. The chances of getting another character when intending an "e" were 4.0%. There were no omissions of "e" and 5 insertions of "e", or 1.4% of all entered "e"'s. These are the types of results available from our character-level analysis of input streams.



Character	Entered	Intended	Correct	Insertion	Substitution	Omission	Sum
'z'	14	7	5	0.0%	28.6%	0.0%	28.6%
'f'	80	46	41	0.0%	10.9%	2.5%	13.4%
'j'	15	11	10	6.7%	9.1%	0.0%	15.8%
'e'	360	344	330	1.4%	4.0%	0.0%	5.4%
Total	4479	4446.83	3808.17 *Avg.*	0.7%	14.4%	0.8%	15.9%

Table 11.2. An excerpt of character-level results using the current analysis. The bottom row is for the whole table, not just for the excerpt shown here.

Table 11.2 is an excerpt from the full table, containing only a subset of the full table's rows and columns. It shows the letters with the 3 highest error rates and the letter "e" for comparisons.

The results in Table 11.2 differ substantially from those in Table 11.1 because of the inclusion of *corrected* errors. Overall, an attempt to make a letter had a 14.4% chance of being a substitution, not just a 0.2% chance as reported in Table 11.1. This rate is high because the test subjects were first-time users with minimal practice.

11.5.2 Confusion Matrices

The error rate tables tell us which characters are prone to different errors, among them substitutions. But the tables do not tell us what those substitution errors are. For this we use a confusion matrix (Grudin 1984, MacKenzie and Soukoreff 2002a). This is similar, but not identical to, the confusion matrices used in handwriting recognition. Those confusion matrices indicate where a gesture recognizer has trouble differentiating between written characters or words. By contrast, the confusion matrices here show where the *user* has intended some letter but produced another. This may indeed be due to a poor recognizer, or it may be due to user confusion, forgetting how to make a letter, or guessing incorrectly. For example, in Graffiti, "k" and "x" are mirror images. Although the Graffiti recognizer has no trouble distinguishing between them, novices often mistake one for the other (MacKenzie and Zhang 1997).

In the matrix, the *x*-axis is the intended character and the *y*-axis is the produced character. For a character, "total attempts" is the sum of its matrix column and "total entries" (not including insertions) is the sum of its matrix row.

Values along the diagonal—where the intended and produced characters match—represent correct entries and dwarf the values off the diagonal. Values along the diagonal are therefore omitted so that the substitution errors are more visible in Figure 11.39.

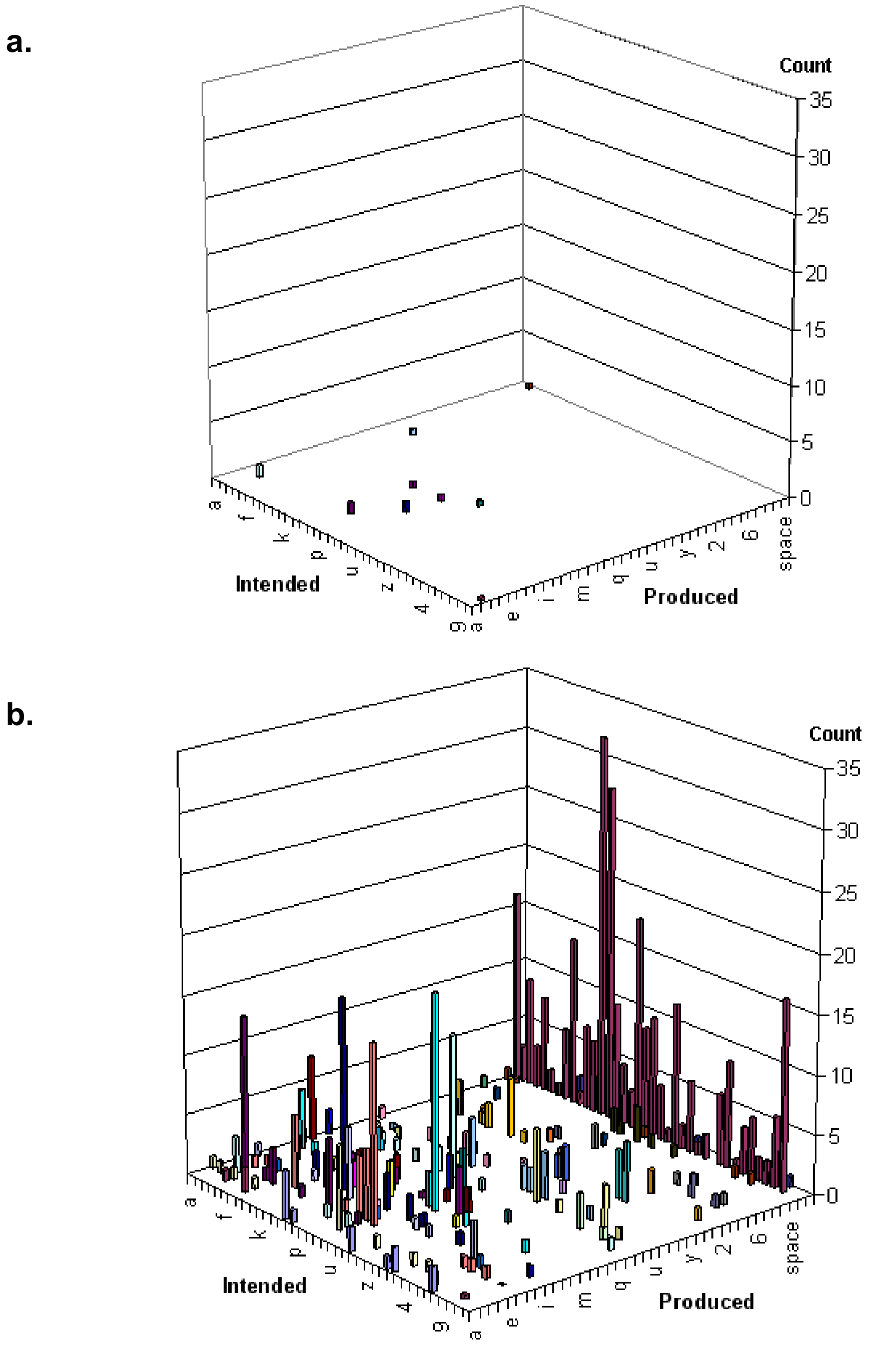


Figure 11.39. (a) Confusion matrices from the previous analysis of *P* and *T* and (b) the current analysis of *P*, *T*, and *IS* for the same empirical data. These matrices are automatically produced by StreamAnalyzer. The large discrepancies in the two graphs are due to corrected substitutions, since these appear in *IS* but not in *T*. The high values against the back wall in the bottom graph are non-recognition substitutions.

Figure 11.39a has a maximum value of just 1 off of its diagonal. Figure 11.39b has a maximum value of 32 for non-recognition substitutions (“m”, ø) and 18 for alphabetic substitutions (“u”, “l”).

One unexpected finding from Figure 11.39b not available in Figure 11.39a is the high number of “l”’s produced when trying for “u”’s, since (“u”, “l”) = 18. In EdgeWrite, if subjects lift their stylus prematurely while trying for a “u”, an “l” can result, and apparently this happened often.

Another unexpected finding was the high non-recognition rate for “n”, since (“n”, ø) = 28. The reason for this appears to be that in making the diagonal portion of “n”, subjects sometimes caught the bottom-left corner accidentally, which resulted in a non-recognition since that corner sequence is not defined in alphanumeric mode. A subsequent redesign of the corner regions changed them from rectangles to triangles and remedied this problem (§4.2.2).

Another interesting character-level finding was the high substitution rate of “i*”* for “l”, since (“l”, “i”) = 16. This tells us that when subjects were trying to produce an “l”, they often conceived of a lowercase “l”-shape, i.e., a line straight down. Such a stroke is an “i” in EdgeWrite, whereas an “l” is a capital “L”-shape down and then across.

Yet another finding of interest was the confusion of “f” for “t”, since (“t”, “f”) = 15. The strokes for “t” and “f” are mirror images of each other, i.e., reflections over the vertical. Apparently this was confusing to subjects when intending to write “t”. However, the reverse was not the case, since (“f”, “t”) = 1, meaning that when trying for “f”, subjects were not confusing it with the shape of “t”. Incorporating linguistic information in the form of letter digraph probabilities, for example, might be one way to remedy this confusion by allowing the “f” stroke to enter “t” if the previous character indicates that “t” is much more likely than “f”.

11.6 Summary

The unconstrained text entry evaluation paradigm has changed the way rigorous text entry experiments can be conducted. However, with the advent of this paradigm comes a major challenge in capturing character-level errors, particularly in the input stream where such errors are most prevalent. Although they were artificial and often frustrating for users, the former constrained text entry evaluation paradigms made it trivial to assess character-level errors, making the use of the new unconstrained paradigm a tradeoff rather than an outright win over older, more artificial paradigms.

This chapter has presented a technique and tools for the analysis of aggregate and character-level errors in the input stream, independent of the character-level entry method under investigation. It has shown the value in performing character-level error analyses on input streams rather than just on transcribed strings. Despite the inherent ambiguity in input streams, we are able to extract error information using four testable assumptions. The taxonomy of input stream error types provides a means of describing character-level errors in a well-defined fashion at a fine-grained level. The current measures, the algorithms that automate them, and the software that implements them can aid designers and evaluators of text entry methods by providing them with a richer picture of the text entry process.

Chapter 12

Conclusion

12.1 Discussion

This dissertation presented a versatile design for text entry and control called *EdgeWrite*. As part of this presentation, new generally applicable text entry concepts have been developed such as continuous recognition feedback, non-recognition retry, slip detection, and word-level stroking. Many of these techniques capitalize on the fact that strokes are fully defined by a sequence of corners—a simple but useful concept that enables new interaction techniques not previously realized in any text entry method. Beyond these new concepts, numerous specific EdgeWrite versions have been designed, built, and evaluated. A wide range of devices, technologies, and segmentation schemes have been used, some targeting motor-impaired users and others aimed at the able-bodied mainstream. Although results for different versions vary, all versions are robust, usable implementations suitable for text entry in the real world.

12.1.1 Design Space

One way of visualizing the various versions of EdgeWrite is by constructing a *design space* (Card *et al.* 1990). The space, shown in Table 12.1, classifies techniques according to segmentation scheme, use of edges, input area size, positioning control, and sensing. It also groups similar techniques and gives their chapter numbers for reference.

The EdgeWrite Design Space

		"Edges"			Input Area Size			Positioning		Sensing	
		Physical	Virtual	None	Tiny	Small	Medium	Absolute	Relative	Planar	Corners
Segmentation	Lift	Stylus Touchpad StampPad WatchPad		ELEW	Stylus ELEW	StampPad WatchPad	Touchpad	Stylus Touchpad StampPad WatchPad ELEW		Stylus Touchpad StampPad WatchPad ELEW	
	Snap-to-Center	Joystick PW-Joystick				Joystick	PW-Joystick	Joystick PW-Joystick		Joystick PW-Joystick	
	Adaptive Timeout			Four-key Four-sensor Four-button EdgePad		Four-sensor Four-button EdgePad	Four-key	Four-key Four-sensor Four-button EdgePad			Four-key Four-sensor Four-button EdgePad
	Pressure Ceases		Trackball Isometric		Isometric		Trackball		Trackball Isometric	Trackball Isometric	
	Button Press	JMEdge				JMEdge		JMEdge		JMEdge	

Stylus or Finger
Stylus EdgeWrite (4)
Touchpad EdgeWrite (6)
StampPad EdgeWrite (10)
WatchPad EdgeWrite (10)

ELEW (10)

Isotonic Joysticks
Joystick EdgeWrite (5)
PW-Joystick EdgeWrite (6)

JMEdge (10)

Relative Control Devices
Trackball EdgeWrite (7)
Isometric Joystick EdgeWrite (8)

Discrete Corners
Four-key EdgeWrite (9)
Four-sensor EdgeWrite (9)
Four-button EdgeWrite (10)

EdgePad (10)

Table 12.1. A design space of EdgeWrite input techniques. Rows group methods by segmentation scheme. Colored column areas represent different properties. Each of the 14 techniques is listed once within each colored area. Numbers in parentheses are chapters.

The design space shows all 14 of the EdgeWrite techniques described in this dissertation, including those developed by others (§10). Although there is similarity among some of the techniques, there is also great diversity when we consider the rather large differences that exist between trackballs and displacement joysticks, styli and four-keys. That one unified design for text entry can work with a mnemonic Roman-like alphabet on such a breadth of devices is an important accomplishment of the current work.

The design space also reveals new opportunities to explore. Empty cells indicate untapped regions that may hold new possibilities for design. For example, it may be fruitful to investigate a stylus version that uses pressure to segment instead of lift, thereby allowing a blind person to maintain contact with the writing surface at all times. Similarly, no "tiny" displacement joystick methods were explored; could snap-to-center segmentation be used for a mobile phone-sized displacement joystick? How would that compare to an isometric joystick (§8)? What about an eye-tracking version that uses an adaptive timeout to segment, since no concept of "lift" is available? These are just some of the possibilities the design space reveals.

12.1.2 Major Results

Beyond EdgeWrite devices and techniques, this dissertation has presented empirical studies of EdgeWrite methods, often comparing them against commercially-available competitor methods. In summary, the significant empirical results that highlight EdgeWrite's strengths are:

- *Stylus EdgeWrite* had 15.4% lower KSPC than Graffiti (1.21 *vs.* 1.43) for able-bodied novices during entry with no significant difference in speed. Furthermore, Stylus EdgeWrite was 37.3% more accurate than Graffiti for five motor-impaired novices (98.4% *vs*. 71.7% accuracy). Able-bodied experts are able to write at 24.0 WPM with 2.8% total errors with the character-level version. In addition, Fisch in-stroke word completion supports "record speeds" of over 60 WPM for certain phrases.

- *Joystick EdgeWrite* left 62.5% fewer errors than selection keyboard (0.27% *vs.* 0.72% uncorrected errors) and was 3.7% faster (6.40 *vs.* 6.17 WPM) for able-bodied novices. In addition, it left 56.5% fewer errors (0.27% *vs.* 0.62%

uncorrected errors) and was 44.5% faster (6.40 *vs*. 4.43 WPM) than date stamp. On a 5-point Likert scale, subjects rated EdgeWrite 39.1% better than selection keyboard and 33.0% better than date stamp. Experts range from 10.4–14.7 WPM with 6.3–8.8% total errors.

- *Power Wheelchair Joystick EdgeWrite* was initially slower (0.77 *vs.* 0.84 WPM) and less accurate than joystick WiViK for motor-impaired novices. However, it improved significantly over sessions, and its learning curve shows a crossover point with joystick WiViK at session 8 (~3 hours). Its peak session speed was 0.98 WPM in session 7. Able-bodied experts can write at 12.9 WPM with 8.4% total errors.

- *Touchpad EdgeWrite* was 19.0% faster (1.00 *vs.* 0.84 WPM) but made significantly more errors than joystick WiViK for motor-impaired novices. Nevertheless, subjects categorically preferred Touchpad EdgeWrite to joystick WiViK. Touchpad EdgeWrite improved significantly over sessions, and its learning curve shows a crossover point with touchpad WiViK at session 5.5 (~2 hours). Its peak speed was 1.56 WPM in session 8. Able-bodied experts can write at 19.1 WPM with 4.7% total errors.

- *Trackball EdgeWrite* was 15.4% faster (5.61 *vs*. 4.86 WPM) and left 44.8% fewer errors (1.12% *vs*. 2.03% uncorrected errors) than an on-screen keyboard on average over nine semi-weekly sessions for a subject with a spinal cord injury. Its peak character-level speed was 8.25 WPM compared to 6.90 WPM for the on-screen keyboard. In a second study, word-level Trackball EdgeWrite entered text at 12.09 WPM, which was 46.5% faster than character-level Trackball EdgeWrite (8.25 WPM) and 75.2% faster than the on-screen keyboard without word prediction (6.90 WPM). Also, the word-level version provides a 43.9% stroke savings over long-term use. Furthermore, the subject indicated a strong preference for EdgeWrite because of its reduced visual attention and fatigue. He has given up on-screen keyboards in favor of Trackball EdgeWrite for everyday use. Able-bodied experts can write from 6.75–12.75 WPM with <4% total errors with the character-level version.

- *Isometric Joystick EdgeWrite* was just as fast as Multitap, each at about 9.5 WPM. Multitap had fewer errors, but both methods had low uncorrected errors (≤1%). Learning curves show EdgeWrite overtaking Multitap at session 18 (~4 hours). Two of 4 subjects were significantly faster with EdgeWrite, and all 4 subjects preferred EdgeWrite to Multitap. The word-level version was just as fast as T9 for 3 subjects (~12.8 WPM), but for one subject it was significantly slower than T9. However, it was 36.4% faster and left 46.5% fewer errors than the character-level version of Isometric Joystick EdgeWrite.

  Isometric Joystick EdgeWrite outperformed Multitap when used "under the table" such that the device was held out of sight. EdgeWrite was faster than Multitap (9.78 *vs.* 5.00 WPM) and more accurate (18.77% *vs.* 28.16% total errors).

  Isometric Joystick EdgeWrite showed promise on the back of a mobile phone using an index finger. Subjects could write at 7.70 WPM with 2.86% uncorrected errors. The peak session speed for the back joystick was 8.87 WPM.

- *Four-key EdgeWrite* was tested with and without its stroke visualization. With these versions combined, after 10 sessions it was 67.2% faster than a 3-key method (16.41 *vs.* 9.81 WPM) and 27.6% faster than a 5-key method (16.41 *vs.* 12.86 WPM). However, it also left more errors, although uncorrected error rates were low for all methods (≤2%). The fastest subject entered his quickest phrase at 24.06 WPM with Four-key EdgeWrite. In addition, not seeing the stroke visualization seemed to have no effect on performance with Four-key EdgeWrite.

- *Four-sensor EdgeWrite* supports writing using four capacitive sensors. Able-bodied expert speeds are 11.38 WPM with 4.36% total errors.

12.1.3 Reflections and Insights

This dissertation has presented multiple versions of EdgeWrite. But what can be said about EdgeWrite in general? Although each version uses the same alphabet, most versions differ in substantial ways, making it difficult to derive lessons from EdgeWrite as a whole. However, there are some reflections and insights that the process of designing, building, and evaluating EdgeWrite has generated. These are shared in this section.

In many ways, EdgeWrite's corners are more central than its edges. In fact, the importance of EdgeWrite's corners makes *CornerWrite* a reasonable alternative. Although some versions of EdgeWrite have physical edges, others do not—but *all* versions employ some notion of corners as targets for strokes. Corners define EdgeWrite's characters. They constitute the "bare necessities" of the technique, as demonstrated by Four-key (§9.2) and Four-sensor EdgeWrite (§9.4), which have nothing *but* corners. Certainly physical edges are helpful to the stylus, touchpad, and joystick versions, but corners play a critical role in all versions of the technique.

EdgeWrite's Roman-like letter-strokes are one of its major strengths. Having Roman-like letter-forms brings a great number of benefits, including higher guessability, shortened learning and practice times, better memorability, and lessened frustration for users who may balk at having to learn seemingly arbitrary gestures. The major tradeoff here is that the EdgeWrite gestures are not optimized for minimal movement, but the difficulty of learning Unistrokes (Goldberg and Richardson 1993), the need for reduced practice sessions for people who fatigue rapidly, and the intermittent use of mobile devices all point toward the need for a Roman-like alphabet.

Another benefit of EdgeWrite's corner-defined Roman-like alphabet is that it can be learned on one device but immediately used on another. This cross-device transfer of skill was useful to Jim (§7.3.3), who initially learned EdgeWrite on a trackball but could later transfer his knowledge to the Palm PDA. Similarly, the subjects who used the steering wheel version (§10.1) had previously learned the alphabet with the isometric joystick on the mobile phone (§8.3). Transferring their knowledge of the EdgeWrite alphabet from the phone to the steering wheel was very easy for these subjects.

It is clear from this work that the process of designing technology can benefit from incorporating both theoretical models and real-world participants. The use of a theoretical model helped correct an erroneous intuition in the design of Trackball EdgeWrite (§7.2.3), namely that removing the need to move to the first corner would result in a faster technique. Although this intuition seemed straightforward, it turned out to be wrong both in theory and in practice.

Similarly, incorporating real participants in the design process helped improve the designer-made EdgeWrite alphabet (§3.3.3), even though participants were not aware that they were "designing" at all. The variety of responses made available by participants is a key benefit. The challenge is to reduce the participants' task to something concrete while still being valuable to the designers.

In empirical studies, it was interesting to note that subjects often discovered character strokes other than those appearing on the primary chart (Figure 3.9). These discoveries support the case for high *redundancy* in the EdgeWrite alphabet. Notably, EdgeWrite has more alternate strokes for letters than Unistrokes, Graffiti, or even Jot. Also interesting was that when subjects found an alternate character form, they usually stuck with it, preferring to use something they had discovered than to continue guessing possible alternatives.

It was also clear from studies that subjects' motor perception is much faster than their visual perception. For example, subjects would often "feel" that they had made a mistake while making a stroke and (sometimes prematurely, before segmentation) enter a backspace. This phenomenon may have been heightened for the versions with physical edges, since physical edges provide a sort of *passive haptic* feedback to the user. A feature that takes advantage of this fast motor-perceptual loop is non-recognition retry (§3.4.2), since it allows subjects to fluidly restart letters in which they feel they have erred without requiring them to wait and verify that an error has been produced.

A number of the studies in this work compared EdgeWrite, a gestural technique, to various *selection-based* methods (on-screen keyboards, date stamps, few-key designs, etc.). The same tradeoffs in these studies were always present. Selection-based methods have almost no learning curve but offer limited room for improvement. Gestural methods, like EdgeWrite, are initially more challenging to learn, but offer higher performance in the end. Accordingly, longitudinal studies consisting of multiple sessions are often necessary to compare methods from these two genres. Such studies help solve the problem of how much initial practice to give to each method. They also support the discovery of a crossover point (MacKenzie and Zhang 1999), where a gestural method (often) overtakes a selection-based one. A challenge for future research is to create high-performance selection-based text entry methods that combine the power of gestural methods with the "obviousness" of selection-based methods.

In many ways, this work was made more challenging (and rewarding) by its focus on people with motor impairments. One such challenge was recruiting people with disabilities, since they have a limited ability to travel, frequently interrupted schedules, and a greater likelihood of illness or injury (Coyne 2005, Feng *et al.* 2005). They also fatigue more rapidly, which complicates training at the start of a study. Furthermore, people in special populations are much more varied in their abilities, making design improvements more difficult to make for all users. This illustrates the need for high end-user customizability, such as the settable lift delays in Stylus (§4.2.4) and Touchpad (§6.2.2) EdgeWrite, or the ability to define the EdgeWrite square in most versions.

A final lesson that became clear with each additional EdgeWrite version was that seemingly small improvements can amount to big differences in the ultimate success of a method. The software and hardware for each EdgeWrite version required fine-tuning through much informal and formal testing. For example, the design of the dynamic corners was very different for the joystick version (§5.2) than for the stylus version (§4.2.2). The length of the power wheelchair joystick and the strength of its spring were also noted as having a large impact on the user experience of that version (§6.3.2). Similarly, the same software was used for Trackball EdgeWrite (§7.2) and Isometric Joystick EdgeWrite (§8.2), but not without supporting new features for the latter that included rotating the input space, shrinking the crossing radius, adjusting the mouse sensitivity, and so on. Other input device research has shown similar sensitivity to the fine adjustment of many parameters for ultimate success (Ehrlich 1997).

12.2 Contributions

The contributions of the current work fall into four categories: techniques, artifacts, methods, and tools. Techniques are general concepts that may be applied to other text entry methods or interaction designs. Artifacts are built prototypes useful to some population. Methods are new empirical approaches to evaluation that inform design. Tools are software products useful to other researchers.

TECHNIQUES

- Developed a versatile design for text entry capable of being instantiated on a range of hardware and software. This design, called *EdgeWrite*, was shown to be effective, learnable, and competitive with rival methods.

- Introduced new text entry concepts such as continuous recognition feedback, non-recognition retry, and slip detection. These innovations substantially improve the user experience of EdgeWrite, and may be used in other methods.

- Designed Fisch in-stroke word completion for scaling character-level text entry methods to word-level ones. Importantly, Fisch does not alter or impair regular unistroke text entry. Fisch may be applied to any unistroke method, and was adapted Fisch to Stylus EdgeWrite, Trackball EdgeWrite, and Isometric Joystick EdgeWrite.

- Demonstrated how physical edges and goal crossing can be used to facilitate writing with devices that did not formerly support gestural text entry.

ARTIFACTS

- Built and deployed eight robust implementations of EdgeWrite on devices ranging from styli to joysticks, trackballs to four-keys. In particular, Stylus EdgeWrite and Trackball EdgeWrite are in continued use by real-world users with motor impairments. Other versions, like Touchpad EdgeWrite and Joystick EdgeWrite, are currently being considered by both end-users and companies.

- Designed the highly guessable and learnable EdgeWrite alphabet and its XML definition. This alphabet or a variant may be useful to others who require a well-defined or highly constrained Roman-like character set.

- Built and deployed the EdgeWrite library for easy integration into others' software projects. This library, a DLL, can be used to add EdgeWrite text entry to any .NET application in as few as 10 lines of code.

- Designed, built, and evaluated the first study of back-of-device (BoD) text entry using an isometric joystick on a mobile phone.

METHODS

- Developed a new method for maximizing and evaluating the guessability of symbol sets. This method can aid symbol designers who deal with alphabets, icons, text commands, etc. Operationalized guessability and provided formulae for its measurement.

- Devised a character-level error analysis for the input stream in unconstrained text entry experiments. This error analysis and its accompanying algorithms may help text entry method designers to isolate problematic characters and improve their techniques. It also gives us a fuller picture of the text entry process itself.

TOOLS

- Developed the *TextTest* tool for conducting text entry experiments. TextTest has been used by other text entry researchers (Költringer and Grechenig 2004, Wilson and Agrawala 2006) who have built a technique and needed software to rigorously evaluate it.

- Developed the *StreamAnalyzer* tool for analyzing data collected in a text entry study. StreamAnalyzer produces both aggregate and character-level output, including error rate tables and confusion matrices. The output can be easily pasted into any statistics package or Excel.

12.3 Future Work

Although the current versions of EdgeWrite are robust deployable prototypes, there is still room to improve them. This section details some of those improvements. It also highlights some of the more interesting future projects.

The EdgeWrite DLL (§3.6.1) could be augmented to support *total* replacement of the PC keyboard. Mode-strokes would have to be defined for CONTROL, ALT, SHIFT, and the function keys. In addition, strokes for INSERT, DELETE, ESCAPE, PRINT SCREEN, and so forth would have to be defined. Another improvement would be for the DLL to support end-users defining their own "stroke macros" and then assigning those strokes to actions like pressing a sequence of keyboard keys, launching Microsoft Word, opening a web browser, or outputting one's email signature. For defining stroke macros, only one "training" stroke would be necessary since EdgeWrite's strokes are unambiguously defined by their corner sequences.

For word-level stroking, the DLL could be augmented to support end-user parsing and aggregation of text corpora. Word prediction and completion may improve when users are able to scan their own emails, instant messages, academic papers, and other documents, and build vocabulary lists based on their own writing styles.

Thus far, Stylus EdgeWrite has focused on people with motor impairments. But let us remember Lewis Carroll, who recognized the potential of his Nyctograph design for the blind (§2.2.1). EdgeWrite may pose similar value. The challenge of writing completely eyes-free with Stylus EdgeWrite is that when one lifts the stylus to segment, one often loses track of the EdgeWrite square, making it difficult to return the stylus to the square without looking. An alternative may be to devise a *continuous stroking version* where the stylus is never lifted, and therefore users can always feel where they are. Such a system would require a means of segmentation other than "lift." Viable options may be pressure, a pigtail loop (Hinckley *et al.* 2005), a fixed number of corners per letter (Evreinova *et al.* 2004), or unambiguous prefix codes (Isokoski and Raisamo 2000). Whatever the choice, a non-lifting stylus version may be of value to those without sight, or to sighted users in need of a completely eyes-free solution.

Trackball EdgeWrite was modeled using the Steering Law for crossing tasks (Accot and Zhai 1997). Such a model is elusive for Stylus EdgeWrite, however, because hitting physical edges is unlike both pointing and crossing. Acquiring targets in the presence of physical edges is somewhat like crossing but without the "follow-through" that occurs once the goal has been reached. Future work may include studies to elicit a law of acquiring targets along physical edges.

The displacement joystick versions still require accuracy improvements. Although they are capable of producing transcriptions more accurately and faster than previous methods designed for displacement joysticks, their corrected errors should be reduced. Segmentation errors were generally not a problem. Instead, the difficulty was in hitting some unintended corners while moving quickly. Incorporating a path-based recognizer like the one used in ELEW (Andersen and Zhai 2004) may help remedy this problem. For the power wheelchair joystick, improvements could be made by tailoring the joystick's parameters such as spring strength, stick length, and range of motion. As it was, the joystick that was used had a stiff spring and a short stick, which was quite unlike most of its users' own wheelchair joysticks.

Trackball EdgeWrite yielded some of the most promising results of this work (§7.3). However, due to the iterative and participatory nature of its design, it has only been in regular use by one person. There has been recent interest by others in Pittsburgh, Missouri, and Sweden. Thus, future work includes further testing and dissemination.

The software for Trackball EdgeWrite, which is also used in Isometric Joystick EdgeWrite (§8.2), may be useful in an eye-tracking (Isokoski 2000) or nose-pointing version (Austen 2004). Eye-tracking and nose-pointing can be used to move a mouse cursor, and Trackball EdgeWrite would allow goal crossing to replace area pointing while writing. This may make it possible to write gesturally with one's eyes or nose.

newAbilities, Inc. makes a mouth retainer called the *TongueTouch Keypad* to support tongue-typing by people with quadriplegia (http://www.newabilities.com). The 9-button keypad fits in the roof of one's mouth and sends sensor-presses to the desktop PC via a transmitter placed at the back of the head. Currently, one must type with the tongue on tactile bumps in a Multitap-fashion to enter text. Perhaps an EdgeWrite version for the tongue using only four corner sensors would be effective. Furthermore, tongue-based cursor control has been explored using an isometric joystick (Salem and Zhai 1997). Perhaps Salem and Zhai's device would be suitable for Isometric Joystick EdgeWrite.

Isometric Joystick EdgeWrite could also be useful when embedded in other tiny devices. A key fob with an isometric joystick would enable short-length password entry for accessing one's automobile, thereby preventing a lost key from being used to "fish" for a vehicle in a parking lot. A digital pen with an isometric joystick in its tip could double as a desktop or mobile pointing device. The pen might also be outfitted with accelerometers to support EdgeWriting in the air, since articulated "corners" would be marked by extreme changes in acceleration.

Four-sensor EdgeWrite could be integrated with mobile phones by using a touch-sensitive phone keypad. With such a keypad, certain buttons could correspond to the EdgeWrite corners, such as the 1, 3, 7 and 9 keys. Using the 5-key in the center to enable corner re-entries (§4.2.5), Fisch word-level stroking would be possible. Touch-sensitive phone keypads already exist (§2.3.5), and gestural text entry upon them may be successful using Four-sensor EdgeWrite.

New evaluations are also important to future work. In particular, evaluations during actual walking, riding, or simulated driving would better isolate performance under situational impairments. Similarly, further studies of methods when being used eyes-free or in low light are in order. Isometric Joystick EdgeWrite is already promising in this regard (§8.3.2).

The EdgeWrite character set could be modified for entering letters other than those used in English. Although it is already possible to enter most letters used in Western languages, one end-user modified the character set to optimize EdgeWrite for Swedish input, where certain characters with diacritical marks (e.g. å) are common. Going further, one could conceivably define strokes for other non-Roman alphabets such as Hebrew or Cyrillic. Asian input poses a different challenge, however, since most Asian ideograms are multi-stroke and involve a number of small embellishments. A multi-stroke version of EdgeWrite that combines unistrokes into single ideograms may be possible, although the experience of writing would be unavoidably altered from that of making such characters on paper. This would fundamentally alter the nature of EdgeWrite, which thus far has been designed for high guessability and learnability by leveraging similarity to hand-printed Roman letters.

12.4 Final Remarks

This dissertation has attempted to demonstrate the following thesis:

> *A versatile design for text entry and control called "EdgeWrite," which uses physical edges, goal crossing, and a minimized need for sensing, is effective on handhelds and desktops for people with motor and situational impairments.*

The EdgeWrite design has indeed proven versatile, capable of entering text on a variety of devices for different types of users. All versions have been accessible to their intended audience, whether motor-impaired or situationally-impaired using a small device. Physical edges and goal crossing provided the bedrock writing mechanisms on which the EdgeWrite techniques were built. User studies have demonstrated that this thesis indeed holds.

Going forward, it is important that text entry researchers prize more than just speed and accuracy, although these will remain prerequisites for any good technique. Numbered among the qualities of any excellent technique should also be *accessible*. This quality will become continually more important as devices shrink and proliferate through our aging information-rich society.

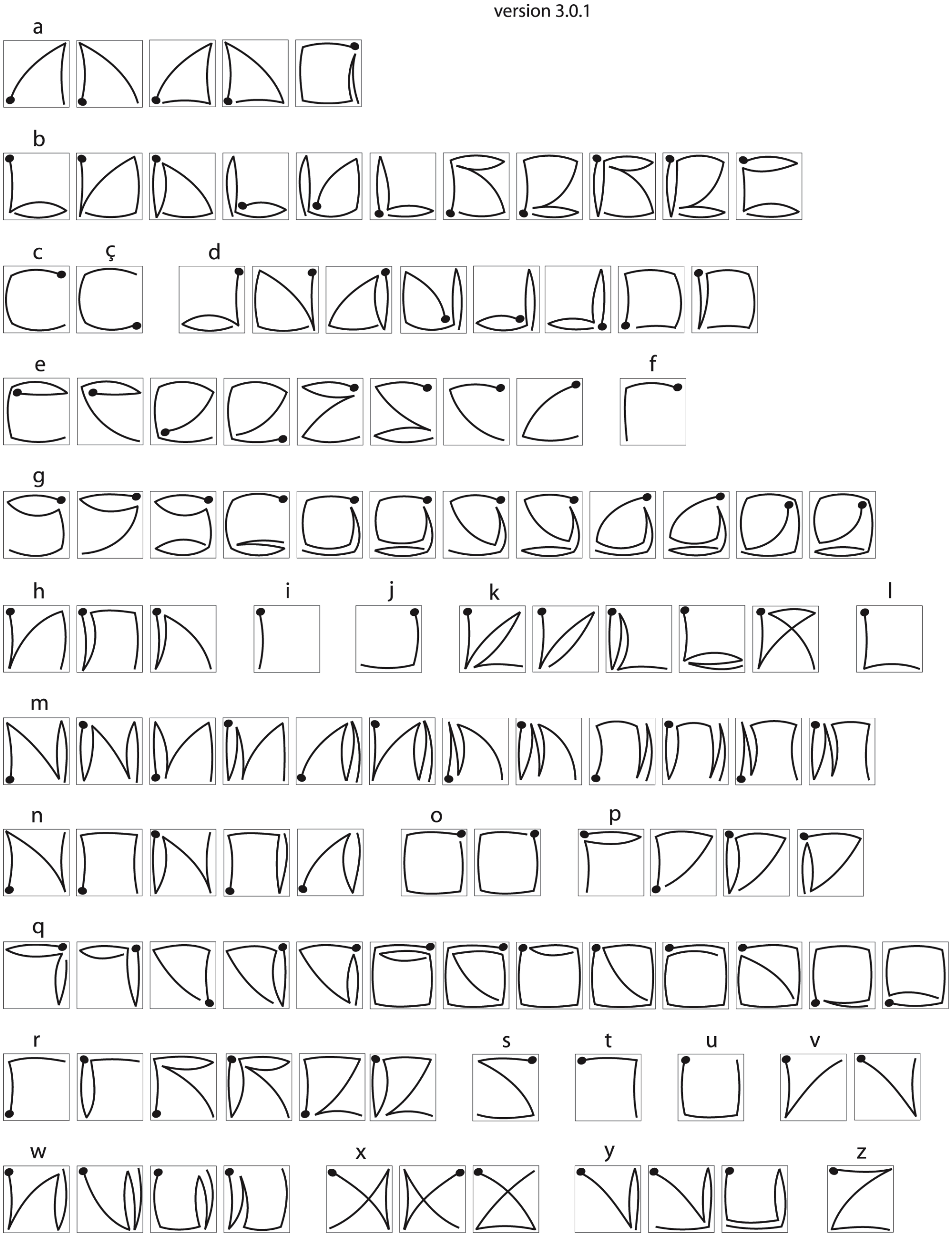
Appendix A

EdgeWrite Character Set

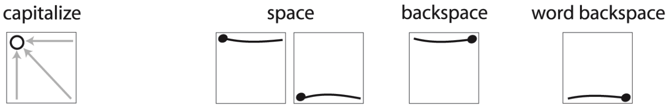
This appendix shows the full EdgeWrite character chart, version 3.0.1, beginning on the next page. After the chart, the XML definition is also given.

Alphanumeric Mode

version 3.0.1

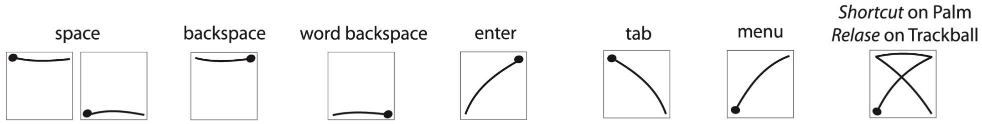


To capitalize, make the letter as usual. Then before lifting, move to the top-left corner and lift.

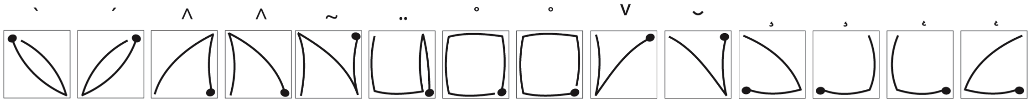


Alphanumeric Mode

version 3.0.1

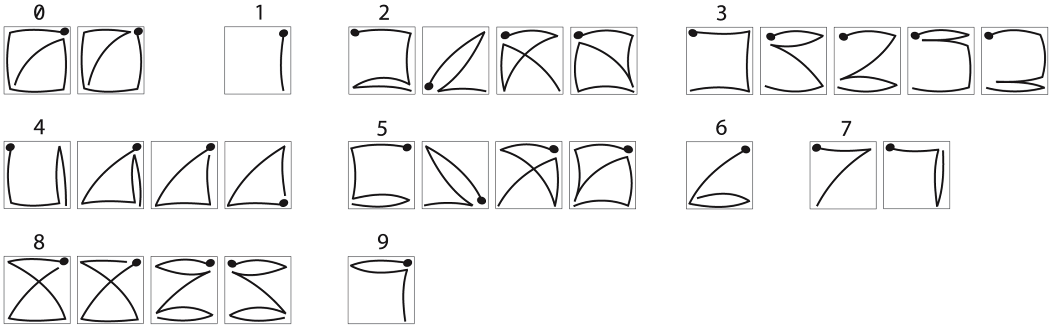


Accents



Accents are made as separate strokes immediately following the letters they modify. Not all letters can be accented.

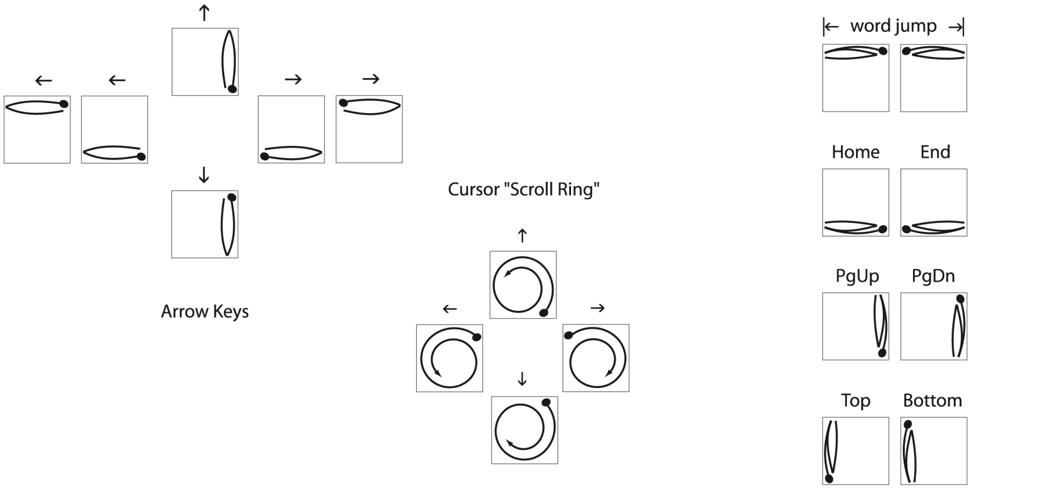
Numbers



Cursor Control

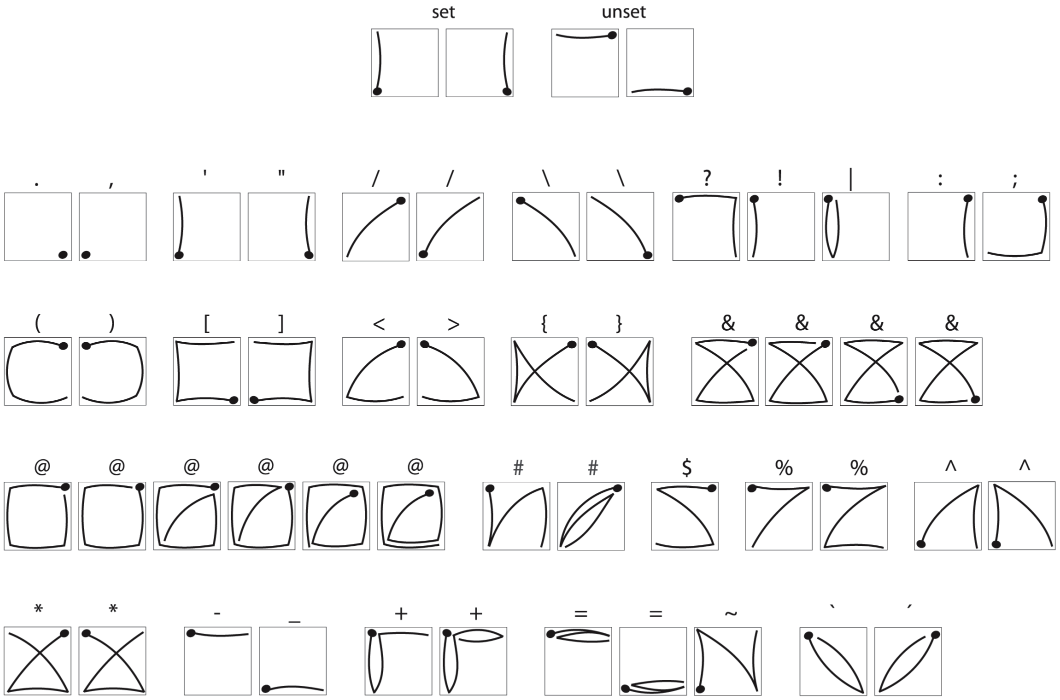
(Cursor movement is available in all modes.)

This section is only available on the PC.



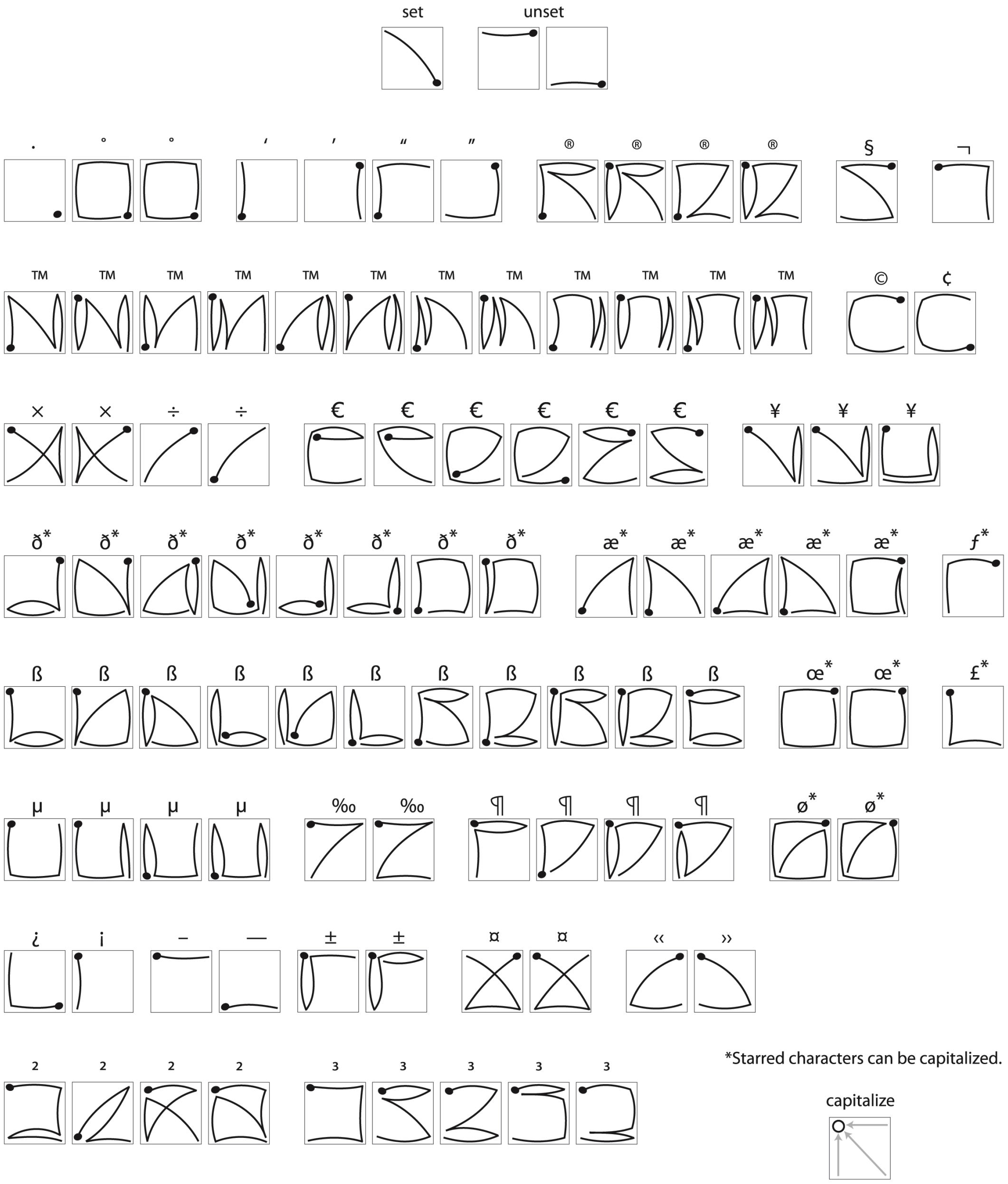
Punctuation Mode

version 3.0.1



Extended Mode

version 3.0.1



*Starred characters can be capitalized.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<charset name="EdgeWrite" version="3.0.1">
  <mode name="All">
    <entry char="backspace" uint="8" sequence="21" comment="character backspace" />
    <entry char="backspace" uint="8" sequence="48" comment="word backspace" />
    <entry char="left" uint="28" sequence="212" comment="cursor left" />
    <entry char="left" uint="28" sequence="484" comment="cursor left" />
    <entry char="right" uint="29" sequence="121" comment="cursor right" />
    <entry char="right" uint="29" sequence="848" comment="cursor right" />
    <entry char="up" uint="30" sequence="424" comment="cursor up" />
    <entry char="down" uint="31" sequence="242" comment="cursor down" />
    <entry char="Ctrl+left" uint="131100" sequence="2121" comment="word left" />
    <entry char="Ctrl+right" uint="131101" sequence="1212" comment="word right" />
    <entry char="Home" uint="15" sequence="4848" comment="Home key" />
    <entry char="End" uint="14" sequence="8484" comment="End key" />
    <entry char="PgUp" uint="11" sequence="4242" comment="Page Up key" />
    <entry char="PgDn" uint="12" sequence="2424" comment="Page Down key" />
    <entry char="Ctrl+Home" uint="131087" sequence="8181" comment="document top" />
    <entry char="Ctrl+End" uint="131086" sequence="1818" comment="document bottom" />
  </mode>
  <mode name="Alphanumeric">
    <mode name="Modes">
      <entry char="Punctuation" uint="2" sequence="81" comment="mode stroke" />
      <entry char="Punctuation" uint="2" sequence="42" comment="mode stroke" />
      <entry char="Extended" uint="3" sequence="41" comment="mode stroke" />
    </mode>
    <mode name="Accents">
      <entry char="à" uint="768" sequence="141" comment="grave" />
      <entry char="á" uint="769" sequence="282" comment="acute" />
      <entry char="â" uint="710" sequence="428" comment="circumflex" />
      <entry char="â" uint="710" sequence="418" comment="circumflex" />
      <entry char="ã" uint="771" sequence="2418" comment="tilde" />
      <entry char="ä" uint="168" sequence="42481" comment="diaeresis" />
      <entry char="å" uint="730" sequence="42184" comment="ring or dot" />
      <entry char="å" uint="730" sequence="48124" comment="ring or dot" />
      <entry char="č" uint="711" sequence="281" comment="caron" />
      <entry char="ă" uint="728" sequence="241" comment="breve" />
      <entry char="ç" uint="184" sequence="841" comment="cedilla" />
      <entry char="ç" uint="184" sequence="842" comment="cedilla" />
      <entry char="ą" uint="731" sequence="481" comment="ogonek" />
      <entry char="ą" uint="731" sequence="482" comment="ogonek" />
    </mode>
    <entry char="space" uint="32" sequence="12" comment="space" />
    <entry char="space" uint="32" sequence="84" comment="space" />
    <entry char="newline" uint="10" sequence="28" comment="enter" />
    <entry char="tab" uint="9" sequence="14" comment="tab" />
    <entry char="alt" uint="18" sequence="82" comment="menu" />

    <entry char="a" uint="97" sequence="824" comment="" />
    <entry char="a" uint="97" sequence="814" comment="" />
    <entry char="a" uint="97" sequence="8248" comment="" />
    <entry char="a" uint="97" sequence="8148" comment="" />
    <entry char="a" uint="97" sequence="218424" comment="" />

    <entry char="b" uint="98" sequence="1848" comment="" />
    <entry char="b" uint="98" sequence="18248" comment="" />
    <entry char="b" uint="98" sequence="18148" comment="" />
    <entry char="b" uint="98" sequence="84818" comment="" />
    <entry char="b" uint="98" sequence="824818" comment="" />
    <entry char="b" uint="98" sequence="81848" comment="" />
    <entry char="b" uint="98" sequence="812148" comment="" />
    <entry char="b" uint="98" sequence="812848" comment="" />
    <entry char="b" uint="98" sequence="1812148" comment="" />
    <entry char="b" uint="98" sequence="1812848" comment="" />
    <entry char="b" uint="98" sequence="121848" comment="" />
```

```
<entry char="c" uint="99" sequence="2184" comment="" />
<entry char="ç" uint="231" sequence="4812" comment="cedilla" />

<entry char="d" uint="100" sequence="2484" comment="d" />
<entry char="d" uint="100" sequence="24184" comment="d" />
<entry char="d" uint="100" sequence="24284" comment="d" />
<entry char="d" uint="100" sequence="48424" comment="d" />
<entry char="d" uint="100" sequence="418424" comment="d" />
<entry char="d" uint="100" sequence="42484" comment="d" />
<entry char="d" uint="100" sequence="81248" comment="D" />
<entry char="d" uint="100" sequence="181248" comment="D" />

<entry char="e" uint="101" sequence="12184" comment="" />
<entry char="e" uint="101" sequence="1214" comment="" />
<entry char="e" uint="101" sequence="82184" comment="" />
<entry char="e" uint="101" sequence="48128" comment="" />
<entry char="e" uint="101" sequence="21284" comment="" />
<entry char="e" uint="101" sequence="21484" comment="" />
<entry char="e" uint="101" sequence="214" comment="quick e" />
<entry char="e" uint="101" sequence="284" comment="quick e" />

<entry char="f" uint="102" sequence="218" comment="" />

<entry char="g" uint="103" sequence="21248" comment="" />
<entry char="g" uint="103" sequence="2128" comment="from 9" />
<entry char="g" uint="103" sequence="212484" comment="" />
<entry char="g" uint="103" sequence="218484" comment="cap form" />
<entry char="g" uint="103" sequence="2184248" comment="" />
<entry char="g" uint="103" sequence="21842484" comment="" />
<entry char="g" uint="103" sequence="214248" comment="" />
<entry char="g" uint="103" sequence="2142484" comment="" />
<entry char="g" uint="103" sequence="284248" comment="" />
<entry char="g" uint="103" sequence="2842484" comment="" />
<entry char="g" uint="103" sequence="281248" comment="" />
<entry char="g" uint="103" sequence="2812484" comment="" />

<entry char="h" uint="104" sequence="1824" comment="" />
<entry char="h" uint="104" sequence="18124" comment="" />
<entry char="h" uint="104" sequence="1814" comment="" />

<entry char="i" uint="105" sequence="18" comment="" />
<entry char="j" uint="106" sequence="248" comment="" />

<entry char="k" uint="107" sequence="18284" comment="" />
<entry char="k" uint="107" sequence="1828" comment="" />
<entry char="k" uint="107" sequence="18184" comment="" />
<entry char="k" uint="107" sequence="18484" comment="" />
<entry char="k" uint="107" sequence="18214" comment="" />

<entry char="l" uint="108" sequence="184" comment="" />

<entry char="m" uint="109" sequence="81424" comment="" />
<entry char="m" uint="109" sequence="181424" comment="" />
<entry char="m" uint="109" sequence="81824" comment="" />
<entry char="m" uint="109" sequence="181824" comment="" />
<entry char="m" uint="109" sequence="82424" comment="" />
<entry char="m" uint="109" sequence="182424" comment="" />
<entry char="m" uint="109" sequence="81814" comment="" />
<entry char="m" uint="109" sequence="181814" comment="" />
<entry char="m" uint="109" sequence="812424" comment="" />
<entry char="m" uint="109" sequence="1812424" comment="" />
<entry char="m" uint="109" sequence="818124" comment="" />
<entry char="m" uint="109" sequence="1818124" comment="" />

<entry char="n" uint="110" sequence="8142" comment="" />
<entry char="n" uint="110" sequence="18142" comment="w?" />
```

```
<entry char="n" uint="110" sequence="8124" comment="" />
<entry char="n" uint="110" sequence="81242" comment="" />
<entry char="n" uint="110" sequence="8242" comment="" />

<entry char="o" uint="111" sequence="21842" comment="ccw" />
<entry char="o" uint="111" sequence="24812" comment="cw" />

<entry char="p" uint="112" sequence="1218" comment="" />
<entry char="p" uint="112" sequence="8128" comment="" />
<entry char="p" uint="112" sequence="18128" comment="" />
<entry char="p" uint="112" sequence="12818" comment="" />

<entry char="q" uint="113" sequence="21242" comment="q" />
<entry char="q" uint="113" sequence="24212" comment="q" />
<entry char="q" uint="113" sequence="4214" comment="q" />
<entry char="q" uint="113" sequence="24214" comment="q" />
<entry char="q" uint="113" sequence="21424" comment="q" />
<entry char="q" uint="113" sequence="2184212" comment="Q" />
<entry char="q" uint="113" sequence="2184214" comment="Q" />
<entry char="q" uint="113" sequence="184212" comment="Q" />
<entry char="q" uint="113" sequence="184214" comment="Q" />
<entry char="q" uint="113" sequence="124812" comment="Q" />
<entry char="q" uint="113" sequence="124814" comment="Q" />
<entry char="q" uint="113" sequence="812484" comment="Q" />
<entry char="q" uint="113" sequence="842184" comment="Q" />

<entry char="r" uint="114" sequence="812" comment="" />
<entry char="r" uint="114" sequence="1812" comment="" />
<entry char="r" uint="114" sequence="81214" comment="" />
<entry char="r" uint="114" sequence="181214" comment="" />
<entry char="r" uint="114" sequence="81284" comment="" />
<entry char="r" uint="114" sequence="181284" comment="" />

<entry char="s" uint="115" sequence="2148" comment="" />
<entry char="t" uint="116" sequence="124" comment="" />
<entry char="u" uint="117" sequence="1842" comment="" />

<entry char="v" uint="118" sequence="182" comment="" />
<entry char="v" uint="118" sequence="142" comment="" />

<entry char="w" uint="119" sequence="18242" comment="" />
<entry char="w" uint="119" sequence="14242" comment="" />
<entry char="w" uint="119" sequence="184242" comment="" />
<entry char="w" uint="119" sequence="181842" comment="" />

<entry char="x" uint="120" sequence="1428" comment="" />
<entry char="x" uint="120" sequence="2814" comment="Graffiti k" />
<entry char="x" uint="120" sequence="1482" comment="" />

<entry char="y" uint="121" sequence="1424" comment="" />
<entry char="y" uint="121" sequence="14248" comment="" />
<entry char="y" uint="121" sequence="184248" comment="" />

<entry char="z" uint="122" sequence="1284" comment="" />

<entry char="0" uint="48" sequence="218428" comment="" />
<entry char="0" uint="48" sequence="248128" comment="" />

<entry char="1" uint="49" sequence="24" comment="" />

<entry char="2" uint="50" sequence="12484" comment="" />
<entry char="2" uint="50" sequence="8284" comment="" />
<entry char="2" uint="50" sequence="12814" comment="" />
<entry char="2" uint="50" sequence="124184" comment="" />
```

```
    <entry char="3" uint="51" sequence="1248" comment="" />
    <entry char="3" uint="51" sequence="12148" comment="" />
    <entry char="3" uint="51" sequence="12848" comment="" />
    <entry char="3" uint="51" sequence="121248" comment="" />
    <entry char="3" uint="51" sequence="124848" comment="" />

    <entry char="4" uint="52" sequence="18424" comment="" />
    <entry char="4" uint="52" sequence="28424" comment="" />
    <entry char="4" uint="52" sequence="2842" comment="" />
    <entry char="4" uint="52" sequence="4824" comment="Brad's" />

    <entry char="5" uint="53" sequence="21848" comment="from g" />
    <entry char="5" uint="53" sequence="4148" comment="" />
    <entry char="5" uint="53" sequence="21428" comment="" />
    <entry char="5" uint="53" sequence="218248" comment="" />

    <entry char="6" uint="54" sequence="2848" comment="" />

    <entry char="7" uint="55" sequence="128" comment="" />
    <entry char="7" uint="55" sequence="1242" comment="Brad's" />

    <entry char="8" uint="56" sequence="21482" comment="" />
    <entry char="8" uint="56" sequence="28412" comment="" />
    <entry char="8" uint="56" sequence="212848" comment="" />
    <entry char="8" uint="56" sequence="121484" comment="" />

    <entry char="9" uint="57" sequence="2124" comment="" />
  </mode>
  <mode name="Punctuation">
    <entry char="." uint="46" sequence="4" comment="period" />
    <entry char="," uint="44" sequence="8" comment="comma" />

    <entry char="'" uint="39" sequence="81" comment="sgl quote" />
    <entry char=""" uint="34" sequence="42" comment="dbl quote" />

    <entry char="/" uint="47" sequence="28" comment="" />
    <entry char="/" uint="47" sequence="82" comment="" />
    <entry char="\" uint="92" sequence="14" comment="" />
    <entry char="\" uint="92" sequence="41" comment="" />

    <entry char="?" uint="63" sequence="124" comment="" />
    <entry char="!" uint="33" sequence="18" comment="" />
    <entry char="|" uint="124" sequence="181" comment="" />

    <entry char=":" uint="58" sequence="24" comment="" />
    <entry char=";" uint="59" sequence="248" comment="" />

    <entry char="(" uint="40" sequence="2184" comment="" />
    <entry char=")" uint="41" sequence="1248" comment="" />
    <entry char="[" uint="91" sequence="4812" comment="" />
    <entry char="]" uint="93" sequence="8421" comment="" />
    <entry char="<" uint="60" sequence="284" comment="" />
    <entry char=">" uint="62" sequence="148" comment="" />
    <entry char="{" uint="123" sequence="2814" comment="" />
    <entry char="}" uint="125" sequence="1428" comment="" />

    <entry char="@" uint="64" sequence="21842" comment="ccw" />
    <entry char="@" uint="64" sequence="24812" comment="cw" />
    <entry char="@" uint="64" sequence="284218" comment="" />
    <entry char="@" uint="64" sequence="2842184" comment="" />
    <entry char="@" uint="64" sequence="218428" comment="Brad's" />
    <entry char="@" uint="64" sequence="248128" comment="" />

    <entry char="#" uint="35" sequence="1824" comment="" />
    <entry char="#" uint="35" sequence="2828" comment="" />
    <entry char="$" uint="36" sequence="2148" comment="" />
    <entry char="%" uint="37" sequence="128" comment="" />
    <entry char="%" uint="37" sequence="1284" comment="" />
    <entry char="^" uint="94" sequence="824" comment="" />
    <entry char="^" uint="94" sequence="814" comment="" />

    <entry char="&" uint="38" sequence="21482" comment="" />
    <entry char="&" uint="38" sequence="28412" comment="" />
    <entry char="&" uint="38" sequence="41284" comment="" />
    <entry char="&" uint="38" sequence="48214" comment="" />

    <entry char="*" uint="42" sequence="2841" comment="" />
    <entry char="*" uint="42" sequence="1482" comment="" />

    <entry char="-" uint="45" sequence="12" comment="hyphen" />
    <entry char="_" uint="95" sequence="84" comment="underscore" />

    <entry char="+" uint="43" sequence="1812" comment="" />
    <entry char="+" uint="43" sequence="18121" comment="" />
    <entry char="=" uint="61" sequence="1212" comment="" />
    <entry char="=" uint="61" sequence="8484" comment="" />

    <entry char="`" uint="96" sequence="141" comment="" />
    <entry char="´" uint="180" sequence="282" comment="not on keyboard" />
    <entry char="~" uint="126" sequence="8142" comment="" />
  </mode>
  <mode name="Extended">
    <entry char="•" uint="8226" sequence="4" comment="bullet" />
    <entry char="°" uint="176" sequence="42184" comment="degree" />
    <entry char="°" uint="176" sequence="48124" comment="degree" />

    <entry char="™" uint="8482" sequence="81424" comment="TM" />
    <entry char="™" uint="8482" sequence="181424" comment="TM" />
    <entry char="™" uint="8482" sequence="81824" comment="TM" />
    <entry char="™" uint="8482" sequence="181824" comment="TM" />
    <entry char="™" uint="8482" sequence="82424" comment="TM" />
    <entry char="™" uint="8482" sequence="182424" comment="TM" />
    <entry char="™" uint="8482" sequence="81814" comment="TM" />
    <entry char="™" uint="8482" sequence="181814" comment="TM" />
    <entry char="™" uint="8482" sequence="812424" comment="TM" />
    <entry char="™" uint="8482" sequence="1812424" comment="TM" />
    <entry char="™" uint="8482" sequence="818124" comment="TM" />
    <entry char="™" uint="8482" sequence="1818124" comment="TM" />

    <entry char="®" uint="174" sequence="81214" comment="(R)" />
    <entry char="®" uint="174" sequence="181214" comment="(R)" />
    <entry char="®" uint="174" sequence="81284" comment="(R)" />
    <entry char="®" uint="174" sequence="181284" comment="(R)" />

    <entry char="©" uint="169" sequence="2184" comment="(C)" />
    <entry char="¢" uint="162" sequence="4812" comment="cent" />

    <entry char="‘" uint="8216" sequence="81" comment="left sgl quote" />
    <entry char="“" uint="8220" sequence="812" comment="left dbl quote" />
    <entry char="’" uint="8217" sequence="24" comment="right sgl quote" />
    <entry char="”" uint="8221" sequence="248" comment="right dbl quote" />

    <entry char="§" uint="167" sequence="2148" comment="section" />
    <entry char="×" uint="215" sequence="1428" comment="multiply" />
    <entry char="×" uint="215" sequence="2814" comment="multiply" />
    <entry char="÷" uint="247" sequence="28" comment="divide" />
    <entry char="÷" uint="247" sequence="82" comment="divide" />

    <entry char="€" uint="8364" sequence="12184" comment="euro" />
    <entry char="€" uint="8364" sequence="1214" comment="euro" />
    <entry char="€" uint="8364" sequence="82184" comment="euro" />
```

```
<entry char="€" uint="8364" sequence="48128" comment="euro" />
<entry char="€" uint="8364" sequence="21284" comment="euro" />
<entry char="€" uint="8364" sequence="21484" comment="euro" />

<entry char="¥" uint="165" sequence="1424" comment="yen" />
<entry char="¥" uint="165" sequence="14248" comment="yen" />
<entry char="¥" uint="165" sequence="184248" comment="yen" />

<entry char="£" uint="163" sequence="184" comment="pound" />
<entry char="£" uint="163" sequence="1841" comment="pound" />

<entry char="ð" uint="240" sequence="2484" comment="eth" />
<entry char="ð" uint="240" sequence="24184" comment="eth" />
<entry char="ð" uint="240" sequence="24284" comment="eth" />
<entry char="ð" uint="240" sequence="48424" comment="eth" />
<entry char="ð" uint="240" sequence="418424" comment="eth" />
<entry char="ð" uint="240" sequence="42484" comment="eth" />
<entry char="ð" uint="240" sequence="81248" comment="eth" />
<entry char="ð" uint="240" sequence="181248" comment="eth" />

<entry char="Đ" uint="208" sequence="24841" comment="Eth" />
<entry char="Đ" uint="208" sequence="241841" comment="Eth" />
<entry char="Đ" uint="208" sequence="242841" comment="Eth" />
<entry char="Đ" uint="208" sequence="484241" comment="Eth" />
<entry char="Đ" uint="208" sequence="4184241" comment="Eth" />
<entry char="Đ" uint="208" sequence="424841" comment="Eth" />
<entry char="Đ" uint="208" sequence="812481" comment="Eth" />
<entry char="Đ" uint="208" sequence="1812481" comment="Eth" />

<entry char="¿" uint="191" sequence="481" comment="inverted ?" />
<entry char="¡" uint="161" sequence="18" comment="inverted !" />

<entry char="æ" uint="230" sequence="824" comment="ae" />
<entry char="æ" uint="230" sequence="814" comment="ae" />
<entry char="æ" uint="230" sequence="8248" comment="ae" />
<entry char="æ" uint="230" sequence="8148" comment="ae" />
<entry char="æ" uint="230" sequence="218424" comment="ae" />

<entry char="Æ" uint="198" sequence="8241" comment="AE" />
<entry char="Æ" uint="198" sequence="8141" comment="AE" />
<entry char="Æ" uint="198" sequence="82481" comment="AE" />
<entry char="Æ" uint="198" sequence="81481" comment="AE" />
<entry char="Æ" uint="198" sequence="2184241" comment="AE" />

<entry char="œ" uint="339" sequence="21842" comment="oe" />
<entry char="œ" uint="339" sequence="24812" comment="oe" />

<entry char="Œ" uint="338" sequence="218421" comment="OE" />
<entry char="Œ" uint="338" sequence="248121" comment="OE" />

<entry char="ß" uint="223" sequence="1848" comment="ss" />
<entry char="ß" uint="223" sequence="18248" comment="ss" />
<entry char="ß" uint="223" sequence="18148" comment="ss" />
<entry char="ß" uint="223" sequence="84818" comment="ss" />
<entry char="ß" uint="223" sequence="824818" comment="ss" />
<entry char="ß" uint="223" sequence="81848" comment="ss" />
<entry char="ß" uint="223" sequence="812148" comment="ss" />
<entry char="ß" uint="223" sequence="812848" comment="ss" />
<entry char="ß" uint="223" sequence="1812148" comment="ss" />
<entry char="ß" uint="223" sequence="1812848" comment="ss" />
<entry char="ß" uint="223" sequence="121848" comment="ss" />

<entry char="µ" uint="181" sequence="1842" comment="mu" />
<entry char="µ" uint="181" sequence="18424" comment="mu" />
<entry char="µ" uint="181" sequence="81842" comment="mu" />
<entry char="µ" uint="181" sequence="818424" comment="mu" />
```

```
    <entry char="ƒ" uint="402" sequence="218" comment="florin" />
    <entry char="₣" uint="8355" sequence="2181" comment="Franc" />

    <entry char="ø" uint="248" sequence="218428" comment="o-slash" />
    <entry char="ø" uint="248" sequence="248128" comment="o-slash" />
    <entry char="Ø" uint="216" sequence="2184281" comment="O-slash" />
    <entry char="Ø" uint="216" sequence="2481281" comment="O-slash" />

    <entry char="–" uint="8211" sequence="12" comment="en dash" />
    <entry char="—" uint="8212" sequence="84" comment="em dash" />
    <entry char="±" uint="177" sequence="1812" comment="+/-" />
    <entry char="±" uint="177" sequence="18121" comment="+/-" />

    <entry char="²" uint="178" sequence="12484" comment="^2" />
    <entry char="²" uint="178" sequence="8284" comment="^2" />
    <entry char="²" uint="178" sequence="12814" comment="^2" />
    <entry char="²" uint="178" sequence="124184" comment="^2" />

    <entry char="³" uint="179" sequence="1248" comment="^3" />
    <entry char="³" uint="179" sequence="12148" comment="^3" />
    <entry char="³" uint="179" sequence="12848" comment="^3" />
    <entry char="³" uint="179" sequence="121248" comment="^3" />
    <entry char="³" uint="179" sequence="124848" comment="^3" />

    <entry char="¬" uint="172" sequence="124" comment="not" />

    <entry char="¶" uint="182" sequence="1218" comment="paragraph" />
    <entry char="¶" uint="182" sequence="8128" comment="paragraph" />
    <entry char="¶" uint="182" sequence="18128" comment="paragraph" />
    <entry char="¶" uint="182" sequence="12818" comment="paragraph" />

    <entry char="¤" uint="164" sequence="2841" comment="currency" />
    <entry char="¤" uint="164" sequence="1482" comment="currency" />

    <entry char="‰" uint="8240" sequence="128" comment="permille" />
    <entry char="‰" uint="8240" sequence="1284" comment="permille" />

    <entry char="«" uint="171" sequence="284" comment="<<" />
    <entry char="»" uint="187" sequence="148" comment=">>" />
  </mode>
</charset>
```

Bibliography

Abascal, J. and Civit, A. (2001) "Universal access to mobile telephony as a way to enhance the autonomy of elderly people." *Proceedings of the 2001 EC/NSF Workshop on Universal Accessibility of Ubiquitous Computing*. Alcácer do Sal, Portugal (May 22-25, 2001). New York: ACM Press, pp. 93-99.

Accot, J. and Zhai, S. (1997) "Beyond Fitts' law: Models for trajectory-based HCI tasks." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '97)*. Atlanta, Georgia (March 22-27, 1997). New York: ACM Press, pp. 295-302.

Accot, J. and Zhai, S. (1999) "Performance evaluation of input devices in trajectory-based tasks: An application of the Steering Law." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, Pennsylvania (May 1999). New York: ACM Press, pp. 466-472.

Accot, J. and Zhai, S. (2002) "More than dotting the i's: Foundations for crossing-based interfaces." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '02)*. Minneapolis, Minnesota (April 20-25, 2002). New York: ACM Press, pp. 73-80.

Andersen, T. H. and Zhai, S. (2004) "Explorations and experiments on the use of auditory and visual feedback in pen-gesture interfaces." *DIKU Technical Report 04/16* (December 14, 2004). Copenhagen, Denmark: University of Copenhagen.

Anson, D. K. (1997) *Alternative Computer Access: A Guide to Selection*. Philadelphia: F. A. Davis Company.

Anson, D. K., Moist, P., Przywara, M., Wells, H., Saylor, H. and Maxime, H. (2005) "The effects of word completion and word prediction on typing rates using on-screen keyboards." *Proceedings of the 28th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '05)*. Atlanta, Georgia (June 23-27, 2005). Arlington, Virginia: RESNA Press.

Apitz, G. and Guimbretière, F. (2004) "CrossY: A crossing-based drawing application." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '04)*. Santa Fe, New Mexico (October 24-27, 2004). New York: ACM Press, pp. 3-12.

Arvo, J. and Novins, K. (2000) "Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '00)*. San Diego, California (November 6-8, 2000). New York: ACM Press, pp. 73-80.

Austen, I. (2004) "A new rule of cursor control: Just follow your nose." *The New York Times*. October 28, 2004, New York.

Balakrishnan, R. (2004) ""Beating" Fitts' law: Virtual enhancements for pointing facilitation." *International Journal of Human-Computer Studies 61* (6), pp. 857-874.

Barrett, R. C., Selker, E. J., Rutledge, J. D. and Olyha, R. S. (1995) "Negative inertia: A dynamic pointing function." *Companion to the ACM Conference on Human Factors in Computing Systems (CHI '95)*. Denver, Colorado (May 7-11, 1995). New York: ACM Press, pp. 316-317.

BBC News. (2006) "Gadget firms tackled on usability." *BBC News*. May 15, 2006, London, UK.

Bellman, T. and MacKenzie, I. S. (1998) "A probabilistic character layout strategy for mobile text entry." *Proceedings of Graphics Interface 1998*. Vancouver, B.C. (June 18-20, 1998). Toronto: Canadian Information Processing Society, pp. 168-176.

Bergman, E. and Johnson, E. (1995) "Towards accessible human-computer interaction." In *Advances in Human-Computer Interaction, Vol. 5*, J. Nielsen (ed). Norwood, New Jersey: Ablex.

Bertini, E. and Kimani, S. (2003) "Mobile devices: Opportunities for users with special needs." *Proceedings of the 5th Int'l Symposium on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI '03)*. Udine, Italy (September 8-11, 2003). Berlin: Springer-Verlag, pp. 486-491.

Bishop, J. B. and Myers, G. A. (1993) "Development of an effective computer interface for persons with mobility impairment." *Proceedings of the 15th Annual Conference on Engineering in Medicine and Biology*. IEEE Press, pp. 1266-1267.

Blair, P. (2005) "A customizable hardware input interface for wireless, mobile devices." *Proceedings of the 28th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '05)*. Atlanta, Georgia (June 23-27, 2005). Arlington, Virginia: RESNA Press.

Blaskó, G. and Feiner, S. (2004) "An interaction system for watch computers using tactile guidance and bidirectional segmented strokes." *Proceedings of the 8th IEEE Symposium on Wearable Computers (ISWC '04)*. Arlington, Virginia (October 31-November 3, 2004). Los Alamitos, California: IEEE Computer Society, pp. 120-123.

Blickenstorfer, C. H. (1995) "Graffiti: Wow!" *Pen Computing Magazine*, January 1995, pp. 30-31.

Bloor, R. (2003) "Symbian DevZone taks to What Next: A better small device interface?" *The Symbian DevZone*, June 16, 2003.

Brinck, T., Gergle, D. and Wood, S. D. (2001) *Usability for the Web*. San Francisco: Morgan Kaufmann.

Burzagli, L., Billi, M., Gabbanini, F., Graziani, P., Palchetti, E., Bertini, E. and Gabrielli, S. (2005) "Testing accessibility in mobile computing." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Buxton, W., Hill, R. and Rowley, P. (1985) "Issues and techniques in touch-sensitive tablet input." *Proceedings of the ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*. San Francisco, California (July 1985). New York: ACM Press, pp. 215-223.

Buxton, W. and Myers, B. A. (1986) "A study in two-handed input." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '86)*. Boston, Massachusetts (April 13-17, 1986). New York: ACM Press, pp. 321-326.

Buxton, W. (2005) "Piloting through the maze." *interactions 12* (6), November/December 2005, p. 10.

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), pp. 601-613.

Card, S. K., Moran, T. P. and Newell, A. (1983) *The Psychology of Human-Computer Interaction*. Hillsdale, New Jersey: Lawrence Erlbaum.

Card, S. K., Mackinlay, J. D. and Robertson, G. (1990) "The design space of input devices." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '90)*. Seattle, Washington (April 1-5, 1990). New York: ACM Press, pp. 117-124.

Carroll, L. (1891) "The Nyctograph." In *The Magic of Lewis Carroll*, J. Fisher (ed). Great Britain: Thomas Nelson and Sons (1973), pp. 214-217.

Chau, D. H., Wobbrock, J. O., Myers, B. A. and Rothrock, B. (2006) "Integrating isometric joysticks into mobile phones for text entry." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 640-645.

Clarkson, E., Clawson, J., Lyons, K. and Starner, T. (2005) "An empirical study of typing rates on mini-QWERTY keyboards." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (April 2-7, 2005). New York: ACM Press, pp. 1288-1291.

Clarkson, P. R. and Rosenfeld, R. (1997) "Statistical language modeling using the CMU-Cambridge toolkit." *Fifth European Conference on Speech Communication and Technology (Eurospeech '97)*. Rhodes, Greece (September 1997). pp. 2707-2710.

Co-operwrite Ltd. (1997) *myText*. Co-operwrite, Ltd. Available at http://www.my-text.com/technical_memorandum.pdf

Coleman, M. (2001) *Weegie*. SourceForge.net. Available at http://weegie.sourceforge.net

Connell, B. R., Jones, M., Mace, R. L., Mueller, J., Mullick, A., Ostroff, E., Sanford, J., Steinfeld, E., Story, M. and Vanderheiden, G. (1997) "The Principles of Universal Design, Version 2.0." *The Center for Universal Design* (April 1, 1997). Raleigh, North Carolina: North Carolina State University.

Cook, A. M. and Hussey, S. M. (2001) *Assistive Technologies: Principles and Practice*. 2nd ed. St. Louis: Mosby Press.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990) *Introduction to Algorithms*. Cambridge, Massachusetts: The MIT Press.

Coyne, K. P. (2005) "Conducting simple usability studies with users with disabilities." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Damerau, F. (1964) "A technique for computer detection and correction of spelling errors." *Communications of the ACM 7* (3), pp. 171-176.

Dawe, M. (2004) "Complexity, cost and customization: Uncovering barriers to adoption of assistive technology." *Refereed Poster at the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '04)*. Atlanta, Georgia (October 18-20, 2004).

Dawe, M. (2006) "Desperately seeking simplicity: How young adults with cognitive disabilities and their families adopt assistive technologies." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 1143-1152.

Denoue, L. and Chiu, P. (2005) "Ink completion." *Refereed Poster at Graphics Interface 2005*. Victoria, British Columbia (May 9-11, 2005). Canadian Human-Computer Communications Society.

Dimond, T. L. (1957) "Devices for reading handwritten characters." *Proceedings of the Eastern Joint Computer Conference*. Washington, D.C. (December 9-13, 1957). New York: Institute of Radio Engineers, pp. 232-237.

Douglas, S. A., Kirkpatrick, A. E. and MacKenzie, I. S. (1999) "Testing pointing device performance and user assessment with the ISO 9241, Part 9 standard." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, Pennsylvania (May 15-20, 1999). New York: ACM Press, pp. 215-222.

Ehrlich, K. (1997) "A conversation with Ted Selker." *interactions 4* (5), pp. 34-47.

Enns, N. R. N. and MacKenzie, I. S. (1998) "Touchpad-based remote control devices." *Summary for the ACM Conference on Human Factors in Computing Systems (CHI '98)*. Los Angeles, California (April 1998). New York: ACM Press, pp. 229-230.

Epps, B. W., Snyder, H. L. and Muto, W. H. (1986) "Comparison of six cursor devices on a target acquisition task." *Proceedings of the Society for Information Display*. San Diego, California (May 6-8, 1986). Society for Information Display, pp. 302-305.

Epps, B. W. (1987) "A comparison of cursor control devices on a graphics editing task." *Proceedings Human Factors Society 31st Annual Meeting*. New York. Human Factors Society, pp. 442-446.

Evreinova, T., Evreinov, G. and Raisamo, R. (2004) "Four-key text entry for physically challenged people." *Adjunct Proceedings of the 8th ERCIM Workshop on User Interfaces for All (UI4ALL '04)*. Vienna, Austria (June 28-29, 2004).

Farris, J. S., Jones, K. S. and Anders, B. A. (2001) "Acquisition speed with targets on the edge of the screen: An application of Fitts' Law to commonly used web browser controls." *Proceedings of the Human Factors and Ergonomics Society 45th Annual Meeting (HFES '01)*. Minneapolis, Minnesota (October 2001). Human Factors and Ergonomics Society, pp. 1205-1209.

Farris, J. S., Jones, K. S. and Anders, B. A. (2002a) "Using impenetrable borders in a graphical web browser: How does distance influence target selection speed?" *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting (HFES '02)*. Baltimore, Maryland (September/October 2002). Human Factors and Ergonomics Society, pp. 1300-1304.

Farris, J. S., Jones, K. S. and Anders, B. A. (2002b) "Using impenetrable borders in a graphical web browser: Are all angles equal?" *Proceedings of the Human Factors and Ergonomics Society 46th Annual Meeting (HFES '02)*. Baltimore, Maryland (September/October 2002). Human Factors and Ergonomics Society, pp. 1251-1255.

Feng, J., Sears, A. and Law, C. M. (2005) "Conducting empirical experiments involving participants with spinal cord injuries." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Fichten, C. S., Barile, M., Asuncion, J. V. and Fossey, M. E. (2000) "What government, agencies, and organizations can do to improve access to computers for postsecondary students with disabilities: Recommendations based on Canadian empirical data." *International Journal of Rehabilitation Research 23* (3), pp. 191-199.

Fitts, P. M. (1954) "The information capacity of the human motor system in controlling the amplitude of movement." *Journal of Experimental Psychology 47* (6), pp. 381-391.

Fleetwood, M. D., Byrne, M. D., Centgraf, P., Dudziak, K. Q., Lin, B. and Mogilev, D. (2002) "An evaluation of text-entry in Palm OS—Graffiti and the virtual keyboard." *Proceedings of the Human Factors and Ergonomics Society 46th*

Annual Meeting (HFES '02). Baltimore, Maryland (September 30-October 4, 2002). Human Factors and Ergonomics Society, pp. 617-621.

Fuhrer, C. S. and Fridie, S. E. (2001) "There's a mouse out there for everyone." *Proceedings of CSUN's 16th Annual Conference on Technology and Persons with Disabilities*. Los Angeles, California (March 19-24, 2001). California State University Northridge.

Furnas, G. W., Landauer, T. K., Gomez, L. M. and Dumais, S. T. (1984) "Statistical semantics: Analysis of the potential performance of keyword information systems." In *Human Factors in Computer Systems*, J. C. Thomas and M. L. Schneider (eds). Norwood, New Jersey: Ablex, pp. 187-242.

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), pp. 964-971.

Gentner, D. R., Grudin, J. T., Larochelle, S., Norman, D. A. and Rumelhart, D. E. (1984) "A glossary of terms including a classification of typing errors." In *Cognitive Aspects of Skilled Typewriting*, W. E. Cooper (ed). New York: Springer-Verlag, pp. 39-43.

Goldberg, D. and Richardson, C. (1993) "Touch-typing with a stylus." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '93)*. Amsterdam, The Netherlands (January 1993). New York: ACM Press, pp. 80-87.

Gong, J., Haggerty, B. and Tarasewich, P. (2005) "An enhanced multitap text entry method with predictive next-letter highlighting." *ACM Conference on Human Factors in Computing Systems*. Portland, Oregon (April 2-7, 2005). New York: ACM Press, pp. 1399-1402.

Gong, J. and Tarasewich, P. (2005) "Alphabetically constrained keypad designs for text entry on mobile devices." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (April 2-7, 2005). ACM Press, pp. 211-220.

Good, M. D., Whiteside, J. A., Wixon, D. R. and Jones, S. J. (1984) "Building a user-derived interface." *Communications of the ACM 27* (10), pp. 1032-1043.

Goodenough-Trepagnier, C., Rosen, M. J. and Galdieri, B. (1986) "Word menu reduces communication rate." *Proceedings of the 9th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '86)*. Minneapolis, Minnesota (June 23-26, 1986). Arlington, Virginia: RESNA Press, pp. 354-356.

Göransson, B. (2004) "The re-design of a PDA-based system for supporting people with Parkinson's disease." *Proceedings of the 18th British HCI Group Annual Conference (HCI '04)*. Leeds, United Kingdom (September 6-10, 2004). British Computer Society.

Grossman, T. and Balakrishnan, R. (2005) "The Bubble Cursor: Enhancing target acquisition by dynamic resizing of the cursor's activation area." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (April 2-7, 2005). New York: ACM Press, pp. 281-290.

Grudin, J. T. (1984) "Error patterns in skilled and novice transcription typing." In *Cognitive Aspects of Skilled Typewriting*, W. E. Cooper (ed). New York: Springer-Verlag, pp. 121-143.

GSM World. (2004) "The Netsize Guide." http://www.gsmworld.com.

Guerette, P. and Sumi, E. (1994) "Integrating control of multiple assistive devices: A retrospective review." *Assistive Technology 6* (1), pp. 67-76.

Herz, J. C. (1997) *Joystick Nation: How Videogames Ate Our Quarters, Won Our Hearts, and Rewired Our Minds*. Boston: Little, Brown and Company.

Hick, W. E. (1952) "On the rate of gain of information." *Quarterly Journal of Experimental Psychology 4*, pp. 11-26.

Hinckley, K., Czerwinski, M. and Sinclair, M. (1998) "Interaction and modeling techniques for desktop two-handed input." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98)*. San Francisco, California (November 1998). New York: ACM Press, pp. 49-58.

Hinckley, K. and Sinclair, M. (1999) "Touch-sensing input devices." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, Pennsylvania (May 15-20, 1999). New York: ACM Press, pp. 223-230.

Hinckley, K., Baudisch, P., Ramos, G. and Guimbretière, F. (2005) "Design and analysis of delimiters for selection-action pen gesture phrases in Scriboli." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (April 2-7, 2005). New York: ACM Press, pp. 451-460.

Hiraoka, S., Miyamoto, I. and Tomimatsu, K. (2003) "Behind Touch: A text input method for mobile phone by the back and tactile sense interface." *Proceedings of Interaction 2003*. Tokyo, Japan (February 27-28, 2003). Information Processing Society of Japan, pp. 131-138.

Hirotaka, N. (2003) "Reassessing current cell phone designs: Using thumb input effectively." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 938-939.

Horstmann, H. M. and Levine, S. P. (1991) "The effectiveness of word prediction." *Proceedings of the 14th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '91)*. Kansas City, Missouri (June 21-26, 1991). Arlington, Virginia: RESNA Press, pp. 100-102.

Hull, L. (2004) "Accessibility: It's not just for disabilities any more." *interactions 11* (2), pp. 36-41.

Hwang, F. (2002) "A study of cursor trajectories of motion-impaired users." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '02)*. Minneapolis, Minnesota (April 20-25, 2002). New York: ACM Press, pp. 842-843.

Hwang, F., Keates, S., Langdon, P. and Clarkson, P. J. (2003) "Multiple haptic targets for motion-impaired computer users." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 41-48.

Hyman, R. (1953) "Stimulus information as a determinant of reaction time." *Journal of Experimental Psychology 45* (3), pp. 188-196.

Ingmarsson, M., Dinka, D. and Zhai, S. (2004) "TNT - A numeric keypad based text input method." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 24-29, 2004). New York: ACM Press, pp. 639-646.

Isokoski, P. (2000) "Text input methods for eye trackers using off-screen targets." *Proceedings of the ACM Symposium on Eye Tracking Research and Applications (ETRA '00)*. Palm Beach Gardens, Florida (November 2000). New York: ACM Press, pp. 15-21.

Isokoski, P. and Raisamo, R. (2000) "Device independent text input: A rationale and an example." *Proceedings of the ACM Conference on Advanced Visual Interfaces (AVI '00)*. Palermo, Italy (May 2000). New York: ACM Press, pp. 76-83.

Isokoski, P. (2001) "Model for unistroke writing time." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '01)*. Seattle, Washington (March 31-April 5, 2001). New York: ACM Press, pp. 357-364.

Isokoski, P. and Kaki, M. (2002) "Comparison of two touchpad-based methods for numeric entry." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '02)*. Minneapolis, Minnesota (April 20-25, 2002). New York: ACM Press, pp. 25-32.

Isokoski, P. and Raisamo, R. (2004) "Quikwriting as a multi-device text entry method." *Proceedings of the 4th Nordic Conference on Human-Computer Interaction (NordiCHI '04)*. Tampere, Finland (October 23-27, 2004). New York: ACM Press, pp. 105-108.

Ivanhoe Broadcast News. (Year) "Hi-tech typing." *Discoveries and Breakthroughs Inside Science*. Released October 2005 by The American Institute of Physics, U.S.A. Transcript available at http://www.ivanhoe.com/science/story/2005/10/65a.html

James, C. L. and Reischel, K. M. (2001) "Text input for mobile devices: Comparing model prediction to actual performance." *Proceedings of the ACM Conference on*

Human Factors in Computing Systems (CHI '01). Seattle, Washington (March 31-April 5, 2001). New York: ACM Press, pp. 365-371.

Jannotti, J. (2002) "Iconic text entry using a numeric keypad." Unpublished work. Available at http://www.jannotti.com/papers/iconic-uist02.pdf

Jeffries, R., Miller, J. R., Wharton, C. and Uyeda, K. M. (1991) "User interface evaluation in the real world: A comparison of four techniques." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '91)*. New Orleans, Louisiana (April 27-May 2, 1991). New York: ACM Press, pp. 119-124.

Johnson, B. R., Farris, J. S. and Jones, K. S. (2003) "Selection of web browser controls with and without impenetrable borders: Does width make a difference?" *Proceedings of the Human Factors and Ergonomics Society 47th Annual Meeting (HFES '03)*. Denver, Colorado (October 2003). Human Factors and Ergonomics Society, pp. 1380-1384.

Kanellos, M. (2006) "Microsoft scientists pushing keyboard into the past." *CNET News.com*, May 3, 2006.

Karlson, A. K., Bederson, B. and Contreras-Vidal, J. L. (2006) "Studies in one-handed mobile design: Habit, desire and agility." Unpublished work. Available at http://www.cs.umd.edu/hcil/mobile/survey/

Keates, S., Clarkson, P. J. and Robinson, P. (1998) "Developing a methodology for the design of accessible interfaces." *Proceedings of the 4th ERCIM Workshop on User Interfaces for All (UI4ALL '98)*. Stockholm, Sweden (October 1998). pp. 1-15.

Keates, S., Clarkson, P. J., Harrison, L.-A. and Robinson, P. (2000) "Towards a practical inclusive design approach." *Proceedings of the ACM Conference on Universal Usability (CUU '00)*. Arlington, Virginia (November 16-17, 2000). New York: ACM Press, pp. 45-52.

Keates, S., Hwang, F., Langdon, P., Clarkson, P. J. and Robinson, P. (2002) "Cursor measures for motion-impaired computer users." *Proceedings of the ACM SIGCAPH Conference on Assistive Technologies (ASSETS '02)*. Edinburgh, Scotland (July 2002). New York: ACM Press, pp. 135-142.

Kjeldskov, J. and Stage, J. (2004) "New techniques for usability evaluation of mobile systems." *International Journal of Human-Computer Studies 60* (5-6), pp. 599-620.

Koester, H. H. (2003) "Abandonment of speech recognition by new users." *Proceedings of the 26th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '03)*. Atlanta, Georgia (June 19-23, 2003). Arlington, Virginia: RESNA Press.

Költringer, T. and Grechenig, T. (2004) "Comparing the immediate usability of Graffiti 2 and virtual keyboard." *Extended Abstracts of the ACM Conference on Human*

Factors in Computing Systems (CHI '04). Vienna, Austria (April 24-29, 2004). New York: ACM Press, pp. 1175-1178.

Kraus, L. E., Stoddard, S. and Gilmartin, D. (1996) *Chartbook on Disability in the United States*. Washington, D.C: National Institute on Disability and Rehabilitation Research.

Kristensson, P. and Zhai, S. (2004) "SHARK^2: A large vocabulary shorthand writing system for pen-based computers." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '04)*. Santa Fe, New Mexico (October 24-27, 2004). New York: ACM Press, pp. 43-52.

Kucera, H. and Francis, W. N. (1967) *Computational Analysis of Present-Day American English*. Providence, Rhode Island: Brown University Press.

Kurihara, K., Goto, M., Ogata, J. and Igarashi, T. (2005) "Speech Pen: Predictive handwriting based on ambient multimodal recognition." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 851-860.

Kurniawan, S., King, A., Evans, D. G. and Blenkhorn, P. (2003) "Design and user evaluation of a joystick-operated full-screen magnifier." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 2003). New York: ACM Press, pp. 25-32.

Kurtenbach, G. and Buxton, W. (1994) "User learning and performance with marking menus." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94)*. Boston, Massachusetts (April 24-28, 1994). New York: ACM Press, pp. 258-264.

Landraud, A. M., Avril, J.-F. and Chretienne, P. (1989) "An algorithm for finding a common structure shared by a family of strings." *IEEE Transactions on Pattern Analysis and Machine Intelligence 11* (8), pp. 890-895.

Langolf, G. D., Chaffin, D. B. and Foulke, J. A. (1976) "An investigation of Fitts' Law using a wide range of movement amplitudes." *Journal of Motor Behavior 8* (2), pp. 113-128.

Law, C. M., Sears, A. and Price, K. J. (2005) "Issues in the categorization of disabilities for user testing." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Lee, S., Hong, S. H. and Jeon, J. W. (2003) "Designing a universal keyboard using chording gloves." *Proceedings of the ACM Conference on Universal Usability (CUU '03)*. Vancouver, British Columbia (November 10-11, 2003). New York: ACM Press, pp. 142-147.

Lee, S., Hong, S. H. and Jeon, J. W. (2005) "Universal access to PDA by modular I/O design." *Proceedings of the 11th International Conference on Human-Computer*

Interaction (HCI Int'l '05). Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Leonard, V. K., Jacko, J. A. and Pizzimenti, J. J. (2005) "An exploratory investigation of handheld computer interaction for older adults with visual impairments." *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '05)*. Baltimore, Maryland (October 9-12, 2005). New York: ACM Press, pp. 12-19.

Levenshtein, V. I. (1965) "Binary codes capable of correcting deletions, insertions, and reversals." *Doklady Akademii Nauk SSSR 163* (4), pp. 845-848.

Lewis, J. R. (1999) "Input rates and user preference for three small-screen input methods: Standard keyboard, predictive keyboard, and handwriting." *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*. Houston, Texas (September 27-October 1, 1999). Santa Monica, California: Human Factors and Ergonomics Society, pp. 425-428.

Lewis, J. R., Kennedy, P. J. and LaLomia, M. J. (1999a) "Development of a digram-based typing key layout for single-finger/stylus input." *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*. Houston, Texas (September 27-October 1, 1999). Santa Monica, California: Human Factors and Ergonomics Society, pp. 415-419.

Lewis, J. R., LaLomia, M. J. and Kennedy, P. J. (1999b) "Evaluation of typing key layouts for stylus input." *Proceedings of the Human Factors and Ergonomics Society 43rd Annual Meeting*. Houston, Texas (September 27-October 1, 1999). Santa Monica, California: Human Factors and Ergonomics Society, pp. 420-424.

Li, J., Zhang, X., Ao, X. and Dai, G. (2005) "Sketch recognition with continuous feedback based on incremental intention extraction." *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI '05)*. San Diego, California (January 9-12, 2005). New York: ACM Press, pp. 145-150.

Lin, M., Price, K. J., Goldman, R., Sears, A. and Jacko, J. (2005) "Tapping on the move—Fitts' law under mobile conditions." *Proceedings of the 16th Annual Information Resources Management Association International Conference (IRMA '05)*. San Diego, California (May 15-18, 2005). Hershey, Pennsylvania: Idea Group, pp. 132-135.

Littell, R. C., Milliken, G. A., Stroup, W. W. and Wolfinger, R. D. (1996) *SAS System for Mixed Models*. Cary, North Carolina: SAS Institute, Inc.

Long, A. C., Landay, J. A. and Rowe, L. A. (1999) "Implications for a gesture design tool." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, Pennsylvania (May 15-20, 1999). New York: ACM Press, pp. 40-47.

LoPresti, E. F., Romich, B. A., Hill, K. J. and Spaeth, D. M. (2004) "Evaluation of mouse emulation using the wheelchair joystick." *Proceedings of the 27th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society* *of North America (RESNA '04)*. Orlando, Florida (June 2004). Arlington, Virginia: RESNA Press.

Ludolph, F. and Perkins, R. (1998) "The Lisa user interface." *Summary for the ACM Conference on Human Factors in Computing Systems (CHI '98)*. Los Angeles, California (April 1998). New York: ACM Press, pp. 18-19.

Lyons, K., Starner, T., Plaisted, D., Fusia, J., Lyons, A., Drew, A. and Looney, E. W. (2004) "Twiddler typing: One-handed chording text entry for mobile phones." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 2004). New York: ACM Press, pp. 671-678.

Mace, R. L., Hardie, G. J. and Place, J. P. (1991) "Accesible environments: Toward universal design." In *Design Intervention: Toward a More Humane Architecture*, W. E. Preiser, J. C. Vischer, and E. T. White (eds). New York: Van Nostrand Reinhold.

MacKenzie, I. S., Sellen, A. and Buxton, W. (1991) "A comparison of input devices in elemental pointing and dragging tasks." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '91)*. New Orleans, Louisiana (March 1991). New York: ACM Press, pp. 161-166.

MacKenzie, I. S. (1992) "Fitts' law as a research and design tool in human-computer interaction." *Human-Computer Interaction 7* (1), pp. 91-139.

MacKenzie, I. S. and Oniszczak, A. (1997) "The tactile touchpad." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI' 97)*. Atlanta, Georgia (March 1997). New York: ACM Press, pp. 309-310.

MacKenzie, I. S. and Zhang, S. X. (1997) "The immediate usability of Graffiti." *Proceedings of Graphics Interface 1997*. Kelowna, British Columbia (May 21-23, 1997). Toronto: Canadian Information Processing Society, pp. 129-137.

MacKenzie, I. S. and Oniszczak, A. (1998) "A comparison of three selection techniques for touchpads." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '98)*. Los Angeles, California (April 1998). New York: ACM Press, pp. 336-343.

MacKenzie, I. S. and Zhang, S. X. (1999) "The design and evaluation of a high-performance soft keyboard." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '99)*. Pittsburgh, Pennsylvania (May 15-20, 1999). New York: ACM Press, pp. 25-31.

MacKenzie, I. S., Zhang, S. X. and Soukoreff, R. W. (1999) "Text entry using soft keyboards." *Journal of Behaviour and Information Technology 18* (4), pp. 235-244.

MacKenzie, I. S., Kauppinen, T. and Silfverberg, M. (2001a) "Accuracy measures for evaluating computer pointing devices." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '01)*. Seattle, Washington (March 31-April 5, 2001). New York: ACM Press, pp. 9-16.

MacKenzie, I. S., Kober, H., Smith, D., Jones, T. and Eugene, S. (2001b) "LetterWise: Prefix-based disambiguation for mobile text input." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '01)*. Orlando, Florida (November 2001). New York: ACM Press, pp. 111-120.

MacKenzie, I. S. (2002a) "A note on calculating text entry speed." Unpublished work. Available at http://www.yorku.ca/mack/RN-TextEntrySpeed.html

MacKenzie, I. S. (2002b) "KSPC (keystrokes per character) as a characteristic of text entry techniques." *Proceedings of the 4th Int'l Symposium on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI '02)*. Pisa, Italy (September 18-20, 2002). Heidelberg, Germany: Springer-Verlag, pp. 195-210.

MacKenzie, I. S. (2002c) "Mobile text entry using three keys." *Proceedings of the 2nd Nordic Conference on Human-Computer Interaction (NordiCHI '02)*. Århus, Denmark (October 19-23, 2002). New York: ACM Press, pp. 27-34.

MacKenzie, I. S. and Soukoreff, R. W. (2002a) "A character-level error analysis technique for evaluating text entry methods." *Proceedings of the 2nd Nordic Conference on Human-Computer Interaction (NordiCHI '02)*. Århus, Denmark (October 19-23, 2002). New York: ACM Press, pp. 243-246.

MacKenzie, I. S. and Soukoreff, R. W. (2002b) "Text entry for mobile computing: Models and methods, theory and practice." *Human-Computer Interaction 17* (2), pp. 147-198.

MacKenzie, I. S. and Soukoreff, R. W. (2003) "Phrase sets for evaluating text entry techniques." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 754-755.

Magnuson, T. and Hunnicutt, S. (2002) "Measuring the effectiveness of word prediction: The advantage of long-term use." *Speech, Music and Hearing 43*, pp. 57-67.

Manaris, B., MacGyvers, V. and Lagoudakis, M. (1999) "Universal access to mobile computing devices through speech input." *Proceedings of 12th International Florida AI Research Symposium (FLAIRS '99)*. Orlando, Florida (May 3-5, 1999). Menlo Park, California: AAAI Press, pp. 286-292.

Mankoff, J. and Abowd, G. D. (1998) "Cirrin: A word-level unistroke keyboard for pen input." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98)*. San Francisco, California (November 1998). New York: ACM Press, pp. 213-214.

Marentakis, G. N. and Brewster, S. A. (2006) "Effects of feedback, mobility and index of difficulty on deictic spatial audio target acquisition in the horizontal plane." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 359-368.

Matias, E., MacKenzie, I. S. and Buxton, W. (1996) "One-handed touch-typing on a QWERTY keyboard." *Human-Computer Interaction 11* (1), pp. 1-27.

Mithal, A. K. and Douglas, S. A. (1996) "Differences in movement microstructure of the mouse and the finger-controlled isometric joystick." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '96)*. Vancouver, British Columbia (April 13-18, 1996). New York: ACM Press, pp. 300-307.

Morgan, H. L. (1970) "Spelling correction in systems programs." *Communications of the ACM 13* (2), pp. 90-94.

Murata, A. (1991) "An experimental evaluation of a mouse, joystick, joycard, lightpen, trackball, and touchscreen for pointing: Basic study on human interface design." *Proceedings of the 4th International Conference on Human-Computer Interaction (HCI Int'l '91)*. Stuttgart, Germany (September 1-6, 1991). New York: Elsevier Science, pp. 123-127.

Mustonen, T., Olkkonen, M. and Häkkinen, J. (2004) "Examining mobile phone text legibility while walking." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 24-29, 2004). New York: ACM Press, pp. 1243-1246.

Myers, B. A. (2001) "Using hand-held devices and PCs together." *Communications of the ACM 44* (11), pp. 34-41.

Myers, B. A., Wobbrock, J. O., Yang, S., Yeung, B., Nichols, J. and Miller, R. (2002) "Using handhelds to help people with motor impairments." *Proceedings of the ACM SIGCAPH Conference on Assistive Technologies (ASSETS '02)*. Edinburgh, Scotland (July 8-10, 2002). New York: ACM Press, pp. 89-96.

n-e-ware. (2004) *KeyStick text entry system*, v. 3.3.0. n-e-ware, Inc. Available at http://www.n-e-ware.com/KeyStick.htm

Nelson, J. A. (1994) "The virtual community: A place for the no-longer-disabled." *Proceedings of the 2nd Annual Conference on Virtual Reality and Persons with Disabilities*. Northridge, California. California State University Northridge, pp. 98-102.

Nesbat, S. B. (2003) "A system for fast, full-text entry for small electronic devices." *Proceedings of the ACM Int'l Conference on Multimodal Interfaces (ICMI '03)*. Vancouver, British Columbia (November 5-7, 2003). New York: ACM Press, pp. 4-11.

Nielsen, J. (1993) *Usability Engineering*. San Diego, California: Academic Press.

Orth, M., Post, R. and Cooper, E. (1998) "Fabric computing interfaces." *Conference Summary of the ACM Conference on Human Factors in Computing Systems (CHI '98)*. Los Angeles, California (April 18-23, 1998). New York: ACM Press, pp. 331-332.

Oulasvirta, A., Tamminen, S., Roto, V. and Kuorelahti, J. (2005) "Interaction in 4-second bursts: the fragmented nature of attentional resources in mobile HCI." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (April 2-7, 2005). New York: ACM Press, pp. 919-928.

Paradise, J., Trewin, S. and Keates, S. (2005) "Using pointing devices: Difficulties encountered and strategies employed." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Parsons, L. M. and Shimojo, S. (1987) "Perceived spatial organization of cutaneous patterns on surfaces of the human body in various positions." *Journal of Experimental Psychology 13* (3), pp. 488-504.

Pavlovych, A. and Stuerzlinger, W. (2003) "Less-Tap: A fast and easy-to-learn text input technique for phones." *Proceedings of Graphics Interface 2003*. Halifax, Nova Scotia (June 2003). Waterloo, Ontario: Canadian Human-Computer Communications Society, pp. 97-104.

Pavlovych, A. and Stuerzlinger, W. (2004) "Model for non-expert text entry speed on 12-button phone keypads." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 24-29, 2004). New York: ACM Press, pp. 351-358.

Perkins, R., Keller, D. S. and Ludolph, F. (1997) "Inventing the Lisa user interface." *interactions 4* (1), pp. 40-53.

Perlin, K. (1998) "Quikwriting: Continuous stylus-based text entry." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '98)*. San Francisco, California (November 1-4, 1998). New York: ACM Press, pp. 215-216.

Price, K. J., Lin, M., Feng, J., Goldman, R., Sears, A. and Jacko, J. (2004) "Data entry on the move: An examination of nomadic speech-based text entry." *Proceedings of the 8th ERCIM Workshop on User Interfaces for All (UI4ALL '04)*. Vienna, Austria (June 28-29, 2004). Berlin, Germany: Springer, pp. 460-471.

Raghunath, M. T. and Narayanaswami, C. (2002) "User interfaces for applications on a wrist watch." *Personal and Ubiquitous Computing 6* (1), pp. 17-30.

Rekimoto, J. (2003) "ThumbSense: Automatic input mode sensing for touchpad-based interactions." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 2003). New York: ACM Press, pp. 852-853.

Rekimoto, J., Ishizawa, T., Schwesig, C. and Oba, H. (2003) "PreSense: Interaction techniques for finger sensing input devices." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '03)*. Vancouver, British Columbia (November 2-5, 2003). New York: ACM Press, pp. 203-212.

Riemer-Reiss, M. L. and Wacker, R. R. (2000) "Factors associated with assistive technology discontinuance among individuals with disabilities." *Journal of Rehabilitation 66* (3), pp. 44-50.

Rodriguez, N. J., Borges, J. A. and Acosta, N. (2005) "A study of text and numeric input modalities on PDAs." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

Romich, B. A., LoPresti, E. F., Hill, K. J., Spaeth, D. M., Young, N. A. and Springsteen, J. P. (2002) "Mouse emulation using the wheelchair joystick: Preliminary performance comparison using four modes of control." *Proceedings of the 25th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '02)*. Minneapolis, Minnesota (June/July 2002). Arlington, Virginia: RESNA Press, pp. 106-108.

Rubine, D. (1991) "Specifying gestures by example." *Proceedings of the ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '91)*. Las Vegas, Nevada (July 28-August 2, 1991). New York: ACM Press, pp. 329-337.

Rutledge, J. D. and Selker, T. (1990) "Force-to-motion functions for pointing." *Proceedings of the IFIP TC13 Third Int'l Conference on Human-Computer Interaction (INTERACT '90)*. Cambridge, England (August 27-31, 1990). North-Holland, pp. 701-706.

Ryu, H. and Cruz, K. (2005) "LetterEase: Improving text entry on a handheld device via letter reassignment." *Proceedings of the 19th Conference of CHISIG Australia on Computer-Human Interaction (OzCHI '05)*. Canberra, Australia (November 21-25, 2005). Narrabundah, Australia: CHISIG of Australia, pp. 1-10.

Salem, C. and Zhai, S. (1997) "An isometric tongue pointing device." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '97)*. Atlanta, Georgia. New York: ACM Press, pp. 538-539.

Schwesig, C., Poupyrev, I. and Mori, E. (2004) "Gummi: A bendable computer." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 24-29, 2004). New York: ACM Press, pp. 263-270.

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), pp. 413-433.

Sears, A., Lin, M., Jacko, J. and Xiao, Y. (2003) "When computers fade: Pervasive computing and situationally-induced impairments and disabilities." *Proceedings of the 10th International Conference on Human-Computer Interaction (HCI Int'l '03)*. Crete, Greece (June 22-27, 2003). Mahwah, New Jersey: Lawrence Erlbaum Associates, pp. 1298-1302.

Sears, A. and Young, M. (2003) "Physical disabilities and computing technologies: An analysis of impairments." In *The Human-Computer Interaction Handbook*, J. Jacko and A. Sears (eds). Mahwah, New Jersey: Lawrence Erlbaum Associates, pp. 482-503.

Sears, A. and Zha, Y. (2003) "Data entry for mobile devices using soft keyboards: Understanding the effects of keyboard size and user tasks." *International Journal of Human-Computer Interaction 16* (2), pp. 163-184.

Shein, F., Hamann, G., Brownlow, N., Treviranus, J., Milner, M. and Parnes, P. (1991) "WiViK: A visual keyboard for Windows 3.0." *Proceedings of the 14th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '91)*. Kansas City, Missouri (June 21-26, 1991). Arlington, Virginia: RESNA Press, pp. 160-162.

Shropshire, C. (2003) "Local Web veteran launches Internet service aimed at seniors." *Pittsburgh Post-Gazette*. November 19, 2003, Pittsburgh, Pennsylvania.

Silfverberg, M., MacKenzie, I. S. and Korhonen, P. (2000) "Predicting text entry speed on mobile phones." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '00)*. The Hague, The Netherlands (April 1-6, 2000). New York: ACM Press, pp. 9-16.

Silfverberg, M., MacKenzie, I. S. and Kauppinen, T. (2001) "An isometric joystick as a pointing device for handheld information terminals." *Proceedings of Graphics Interface 2001*. Ottawa, Ontario (June 7-9, 2001). Toronto: Canadian Information Processing Society, pp. 119-126.

Silfverberg, M. (2003) "Using mobile keypads with limited visual feedback: Implications to handheld and wearable devices." *Proceedings of the 5th Int'l Symposium on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI '03)*. Udine, Italy (September 8-11, 2003). Berlin: Springer-Verlag, pp. 76-90.

Silfverberg, M., Korhonen, P. and MacKenzie, I. S. (2003) "Zooming and panning content on a display screen." International patent WO 03/021568 A1, March 13, 2003.

Soede, M. and Foulds, R. A. (1986) "Dilemma of prediction in communication aids." *Proceedings of the 9th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '86)*. Minneapolis, Minnesota (June 23-26, 1986). Arlington, Virginia: RESNA Press, pp. 357-359.

Soukoreff, R. W. and MacKenzie, I. S. (1995) "Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard." *Behaviour and Information Technology 14* (6), pp. 370-379.

Soukoreff, R. W. and MacKenzie, I. S. (2001) "Measuring errors in text entry tasks: An application of the Levenshtein string distance statistic." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '01)*. Seattle, Washington (March 31-April 5, 2001). New York: ACM Press, pp. 319-320.

Soukoreff, R. W. and MacKenzie, I. S. (2003) "Metrics for text entry research: An evaluation of MSD and KSPC, and a new unified error metric." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 113-120.

Soukoreff, R. W. and MacKenzie, I. S. (2004) "Recent developments in text-entry error rate measurement." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 24-29, 2004). New York: ACM Press, pp. 1425-1428.

Spaeth, D. M., Jones, D. K. and Cooper, R. A. (1998) "Universal control interface for people with disabilities." *Saudi Journal of Disability and Rehabilitation 4* (3), pp. 207-214.

Sperling, B. B. and Tullis, T. S. (1988) "Are you a better "mouser" or "trackballer"? A comparison of cursor-positioning performance." *SIGCHI Bulletin 19* (3), pp. 77-81.

Stack, J. B. (2001) "Palm Pilot connects girl with classroom." *QUEST 8* (1), February 2001, pp. 48-49.

Stary, C. (1997) "The role of design and evaluation principles for user interfaces for all." *Proceedings of the 7th International Conference on Human-Computer Interaction (HCI Int'l '97)*. San Francisco, California (August 24-29, 1997). New York: Elsevier Science, pp. 477-480.

Steinfeld, E. (1994) "The concept of universal design." *Proceedings of the Sixth Ibero-American Conference on Accessibility*. Rio de Janeiro (June 19, 1994).

Sutherland, I. E. (1963) "SketchPad: A man-machine graphical communication system." *Proceedings of the AFIPS Joint Computer Conference 23*. pp. 323-328.

Tandler, P. and Prante, T. (2001) "Using incremental gesture recognition to provide immediate feedback while drawing pen gestures." Unpublished work. Available at http://www.ipsi.fraunhofer.de/ambiente/paper/2001/UIST-2001-tandler-gesture-feedback.pdf

Tannenbaum, A. (2005) "Square Alice." *Spring meeting booklet*. Austin, Texas: Lewis Carroll Society of North America.

Tarasewich, P. (2003) "Evaluation of thumbwheel text entry methods." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 756-757.

Tiainen, S. (2000) *The global keyboard optimised for small wireless devices*. GKOS. Available at http://gkos.com

Tremaine, M. (2001) "Making technology accessible economically." *EC/NSF Workshop on Universal Accessibility of Ubiquitous Computing: Providing for the Elderly (WUAUC '01)*. Alcácer do Sal, Portugal (May 22-25, 2001).

Trewin, S. and Pain, H. (1999) "Keyboard and mouse errors due to motor disabilities." *International Journal of Human-Computer Studies 50* (2), pp. 109-144.

Venolia, D. and Neiberg, F. (1994) "T-Cube: A fast, self-disclosing pen-based alphabet." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '94)*. Boston, Massachusetts (April 24-28, 1994). New York: ACM Press, pp. 265-270.

Wagner, R. A. and Fischer, M. J. (1974) "The string-to-string correction problem." *Journal of the Association for Computing Machinery 21* (1), pp. 168-173.

Walker, N. and Smelcer, J. B. (1990) "A comparison of selection time from walking and bar menus." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '90)*. Seattle, Washington (April 1-5, 1990). New York: ACM Press, pp. 221-225.

Ward, D. J., Blackwell, A. F. and MacKay, D. J. C. (2000) "Dasher—A data entry interface using continuous gestures and language models." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '00)*. San Diego, California (November 6-8, 2000). New York: ACM Press, pp. 129-137.

Ward, D. J. and MacKay, D. J. C. (2002) "Fast hands-free writing by gaze direction." *Nature 418*, p. 838.

Wiedenbeck, S. (1999) "The use of icons and labels in an end user application program: An empirical study of learning and retention." *Behavior and Information Technology 18* (2), pp. 68-82.

Wigdor, D. and Balakrishnan, R. (2004) "A comparison of consecutive and concurrent input text entry techniques for mobile phones." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 2004). New York: ACM Press, pp. 81-88.

Willey, M. (1997) "Design and implementation of a stroke interface library." *IEEE Region 4 Student Paper Contest* (March 24, 1997).

Wilson, A. D. and Agrawala, M. (2006) "Text entry using a dual joystick game controller." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 475-478.

Wobbrock, J. O. (2003) "The benefits of physical edges in gesture-making: Empirical support for an edge-based unistroke alphabet." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 942-943.

Wobbrock, J. O., Myers, B. A. and Hudson, S. E. (2003a) "Exploring edge-based input techniques for handheld text entry." *Proceedings of the 3rd Int'l Workshop on Smart Appliances and Wearable Computing (IWSAWC '03). In 23rd Int'l Conference on Distributed Computing Systems Workshops (ICDCSW '03).*

Providence, Rhode Island (May 19-22, 2003). Los Alamitos, California: IEEE Computer Society, pp. 280-282.

Wobbrock, J. O., Myers, B. A. and Kembel, J. A. (2003b) "EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '03)*. Vancouver, British Columbia (November 2-5, 2003). New York: ACM Press, pp. 61-70.

Wobbrock, J. O., Myers, B. A. and Aung, H. H. (2004a) "Writing with a joystick: A comparison of date stamp, selection keyboard, and EdgeWrite." *Proceedings of Graphics Interface 2004*. London, Ontario (May 17-19, 2004). Waterloo, Ontario: Canadian Human-Computer Communications Society, pp. 1-8.

Wobbrock, J. O., Myers, B. A. and Aung, H. H. (2004b) "Joystick text entry with date stamp, selection keyboard, and EdgeWrite." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '04)*. Vienna, Austria (April 24-29, 2004). New York: ACM Press, p. 1550.

Wobbrock, J. O., Myers, B. A., Aung, H. H. and LoPresti, E. F. (2004c) "Text entry from power wheelchairs: EdgeWrite for joysticks and touchpads." *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '04)*. Atlanta, Georgia (October 18-20, 2004). New York: ACM Press, pp. 110-117.

Wobbrock, J. O., Aung, H. H., Myers, B. A. and LoPresti, E. F. (2005a) "Integrated text entry from power wheelchairs." *Journal of Behaviour and Information Technology 24* (3), pp. 187-203.

Wobbrock, J. O., Aung, H. H., Rothrock, B. and Myers, B. A. (2005b) "Maximizing the guessability of symbolic input." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '05)*. Portland, Oregon (April 2-7, 2005). New York: ACM Press, pp. 1869-1872.

Wobbrock, J. O. and Myers, B. A. (2005a) "Gestural text entry on multiple devices." *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '05)*. Baltimore, Maryland (October 9-12, 2005). New York: ACM Press, pp. 184-185.

Wobbrock, J. O. and Myers, B. A. (2005b) "EdgeWrite: A new text entry technique designed for stability." *Proceedings of the 28th Annual Conference of the Rehabilitation Engineering and Assistive Technology Society of North America (RESNA '05)*. Atlanta, Georgia (June 23-27, 2005). Arlington, Virginia: RESNA Press.

Wobbrock, J. O. and Myers, B. A. (2006a) "From letters to words: Efficient stroke-based word completion for trackball text entry." *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '06)*. Portland, Oregon (October 23-25, 2006). New York: ACM Press. To appear.

Wobbrock, J. O. and Myers, B. A. (2006b) "In-stroke word completion." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '06)*.

Montreux, Switzerland (October 15-18, 2006). New York: ACM Press. To appear.

Wobbrock, J. O. and Myers, B. A. (2006c) "Adding gestural text entry to input devices for people with motor impairments." In *Universal Usability*, J. Lazar (ed). New York: John Wiley & Sons. To appear.

Wobbrock, J. O. and Myers, B. A. (2006d) "Trackball text entry for people with motor impairments." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 479-488.

Wobbrock, J. O., Myers, B. A. and Rothrock, B. (2006) "Few-key text entry revisited: Mnemonic gestures on four keys." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '06)*. Montréal, Québec (April 22-27, 2006). New York: ACM Press, pp. 489-492.

Wobbrock, J. O. and Myers, B. A. (2007) "Analyzing the input stream for character-level errors in unconstrained text entry evaluations." *Transactions on Computer-Human Interaction (TOCHI)*. To appear.

Worden, A., Walker, N., Bharat, K. and Hudson, S. E. (1997) "Making computers easier for older adults to use: Area cursors and sticky icons." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '97)*. Atlanta, Georgia (March 1997). New York: ACM Press, pp. 266-271.

Worth, C. D. (2003) "Xstroke: Full-screen gesture recognition for X." *Proceedings of the USENIX Annual Technical Conference (USENIX '03)*. San Antonio, Texas (July 2003). USENIX Association, pp. 187-196.

Wu, T.-F., Wang, H.-P. and Chen, M. C. (2005) "Enabling computer access for children with cerebral palsy." *Proceedings of the 11th International Conference on Human-Computer Interaction (HCI Int'l '05)*. Las Vegas, Nevada (July 22-27, 2005). Mahwah, New Jersey: Lawrence Erlbaum Associates. On proceedings CD.

York, J. and Pendharkar, P. C. (2004) "Human–computer interaction issues for mobile computing in a variable work context." *International Journal of Human-Computer Studies 60* (5-6), pp. 771-797.

Zaborowski, P. S. (2004) "ThumbTec: A new handheld input device." *Proceedings of the 2004 Conference on New Interfaces for Musical Expression (NIME '04)*. Hamamatsu, Japan (June 3-5, 2004). Singapore: National University of Singapore, pp. 112-115.

Zhai, S., Smith, B. A. and Selker, T. (1997) "Dual stream input for pointing and scrolling." *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI '97)*. Atlanta, Georgia (March 22-27, 1997). ACM Press, pp. 305-306.

Zhai, S., Hunter, M. and Smith, B. A. (2000) "The Metropolis keyboard—An exploration of quantitative techniques for virtual keyboard design." *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '00)*. San Diego, California (November 2000). New York: ACM Press, pp. 119-128.

Zhai, S., Hunter, M. and Smith, B. A. (2002) "Performance optimization of virtual keyboards." *Human-Computer Interaction 17* (3), pp. 229-269.

Zhai, S. and Kristensson, P. (2003) "Shorthand writing on stylus keyboard." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '03)*. Ft. Lauderdale, Florida (April 5-10, 2003). New York: ACM Press, pp. 97-104.

Zhai, S., Kristensson, P. and Smith, B. A. (2005) "In search of effective text input interfaces for off the desktop computing." *Interacting with Computers 17* (3), pp. 229-250.

Zhao, R. (1993) "Incremental recognition in gesture-based and syntax-directed diagram editors." *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '93)*. Amsterdam, The Netherlands (April 24-29, 1993). New York: ACM Press, pp. 95-100.

Zipf, G. (1932) *Selective Studies and the Principle of Relative Frequency in Language*. Cambridge, Massachusettes: MIT Press.