A Minimal Device-Independent Text Input Method

Poika Isokoski

University of Tampere
Department of Computer Science
Master's thesis
10.11.1999

University of Tampere
Department of Computer Science
Poika Isokoski: A Minimal Device-Independent Text Input Method
Master's thesis, 86 + 14 pages
November 1999

Abstract

Recently the need to write text on paper with a pen has diminished. One reason for this development is the emergence of various portable computing and communications devices. Each device typically has its own method for text input. Some rely on miniature keyboards while others are used with a stylus. This means that a person owning several of these gadgets needs to learn many writing methods. In an attempt to alleviate the user's learning burden we have constructed a minimal device-independent text input method which can be used with a variety of input devices. In our experiment with five volunteers we found the method to be rather slow at 7.6 words per minute after five hours of practice. A handheld touchpad was used for the practice period. After the practice we measured the writing speed and error rate for mouse, trackball, joystick and keyboard without device specific training. We found that the writing speed is only slightly slower with unpracticed devices. The error rate varied more between the devices. Keyboard and joystick had the best error rates of roughly 3%. Trackball was the worst with 7% of characters requiring corrections.

Contents

Figures

Tables

Acknowledgments

My work would not have been possible (or at least it would have been a great deal more expensive) without GNU and other free software. My thanks go to people who have worked with gcc, Perl, StarOffice, gnuplot, KDE, Redhat Linux, GhostScript and Ghostview, LaTeX, and Netscape Communicator.

Another group of helpful people worth mentioning are those who keep their articles available on the Internet. Many publications are hard to acquire and getting copies may in addition require significant (for a student) amounts of money and time. Electronic versions are in many cases available almost instantly and with minimal cost. Additionally no hard-copies are necessary which is good for the forests and bad for the paper industry.

The people mentioned above directed their helpful efforts to the universe in general without knowing that it would be me who would benefit from their efforts. More personal aid was given by numerous people. My employer, the Department of Mathematics, Statistics and Philosophy at the University of Tampere, graciously granted me the use of office space and computing facilities. Professor Roope Raisamo supervised my work giving numerous suggestions, which at the time seemed like just more work, but in retrospect seem to have made both the project and my report on it better. Petri Latva-Rasku earns a special thanks for many helpful discussions and aid in debugging the thesis text. Possibly the most important help was, however, the effort that the unpaid test subjects were willing to put into the project. Each gladly suffered the heat of summer afternoons in my lab trying to learn the useless skill of writing in a new alphabet.

1 Introduction

The author got interested in writing methods during the first grades of school while spending hundreds of hours practicing the complex curving strokes that were at that time required for proper cursive script in Finland. As his school career progressed and most of the teachers started emphasizing content over form, he moved to much simpler script and typing whenever possible.

While handwriting systems are interesting, and although there probably is a lot of room for optimization in the popular scripts taught in schools throughout the western countries, traditional handwriting is rapidly losing importance. With modern communications gadgets speech and images can be transferred easily enough and when text is needed on paper, it is usually prepared electronically and printed mechanically. However, while memory chips and various disks have replaced paper as storage medium for text, the text still needs to be written using a keyboard or some other device that can transform physical events into digital data.

The term handwriting is thus starting to signify the act of manually inputting text into a computer. Consequently, a new set of rules is used to evaluate a handwriting method. Possibly the most important change is that the physical act of writing is now separated from the appearance of the resulting text. The computer can be programmed to generate any appearance for the characters. This means that the designer of a writing method no longer needs to compromise speed for legibility. Any motor activity that can be repeated accurately enough and tracked by a computer can be used for text input. In other words the field of handwriting is suddenly wide open for inventions.

The liberation from the confines of paper and ink may at first seem to offer endless possibilities for improvement in the speed and ease of writing. However, writing is much more than hand movements. The text that is written needs to be generated too. Humans can produce sensible text only at limited rates. Also, our hands are the most agile and accurate devices we have for manipulating objects. Traditional handwriting has already been using the best abilities we have and therefore it may be unrealistic to expect very large improvements even if the design space may seem to have expanded significantly.

So far the best writing device that has evolved for computer use is the keyboard. Most people with equal training in both write much faster with a keyboard than with a pen (see chapter 6 or Card et al. [1983]). A traditional keyboard has its drawbacks. It is relatively large compared to some modern computing devices and it needs something to support it when both hands are used for writing. When these limitations are taken into account, methods that are much worse in terms of speed and error rate have proven valuable. Even regular handwriting is used as text input method with some pen-based portable computing devices.

The problem with handwriting recognition as a text input method for computers is that the character set is not designed to exploit the strengths of the input device which consists usually of a stylus and a touch screen or a digitizing tablet. The type of input may also be different from what the traditional alphabet such as the Latin[1] alphabet were developed for. Today we may want to input an arbitrary string such as a single 0. This raises the question of how to differentiate it from a single handwritten O or o. If we are not inputting single arbitrary characters, as is the case when writing normal text, this is not a problem. Text recognition algorithms, like people, can guess the correct letter using the context as a guide. Still, several such small problems are enough to force the accuracy of the best handwriting recognizers below 98% [Tappert *et al.*, 1994]. While 98% accuracy is enough for many applications [Frankish *et al.*, 1995, LaLomia, 1994] it may still be somewhat unwise to use a system that has a built in 2% error frequency if more accurate systems are available.

Several new systems have been proposed. In addition to near-perfect accuracy, many of them aim for better speed than can be maintained with Latin hand printing. The problem that arises from many competing text input methods is that people have to learn many different systems to be able to use different computing devices effectively. Currently it seems that there is no hope for a prompt solution to this problem. The more successful text input methods are tightly coupled with certain input devices such as keyboards or touch screens. All computing devices, however, do not have all or any of these input devices and therefore the current text input methods cannot become universally accepted.

With the rise of ubiquitous computing many have turned to speech recognition for answers. While functional speech recognition could be a great aid in some cases, there are situations where one may not wish to talk to computers. Presently

---

[1] The history of the alphabet known today as the Latin alphabet has gone through many cultures. The Phoenicians, Etruscs, Greeks, and finally the Romans all had their significant influence on the alphabet before it was adopted by the Christian church whose Roman Catholic wing kept it going until the invention of mechanical printing and subsequent explosive growth of written text as a communications medium. Thus, it is somewhat a matter of taste or preference which culture we name the alphabet after. Roman and Latin are the most common names we have encountered.

there is serious animosity between people who talk to their mobile phones in buses, trains and street corners and those who do not wish to hear their conversations. If in addition to speaking to their peers, people are to start dictating their email-messages and whatever else they may write with their Personal Digital Assistants (PDAs), the animosity is only going to grow. Sometimes the PDA users would not want the whole railway car to know what they are doing (paying their bills online for example) and would much prefer a silent mode of operation. Thus speech recognition, while very helpful in some situations, cannot always be used.

In summary it seems that if one could create a manually operated universally compatible text input method, it would most likely be very useful. The difficulty is that input device independence may force us to compromise heavily on speed and ease of use.

Unifying interaction techniques is a direct opposite to the trend of developing more task specific interaction devices and techniques. The aim in fragmenting the interaction techniques into task specific methods is to optimize expert performance. The opposite goal attainable with a universal technique is optimal novice performance through skill transfer.

A success in creating a device-independent universal text input method that is good enough for expert use is likely to be a great boon for mankind. Thus, despite the fact that the results seem uncertain, at least one attempt is a responsibility for an aspiring computer scientist.

Before proceeding to the description and evaluation of a new text input method, we will go through a lengthy discussion of preceding and preparatory work. First in Chapter 2 we discuss some general issues in learning and writing. Then in Chapter 3 we cast a short glance on existing text input methods. Chapter 4 presents a proposal for a new manually operated, minimalist and almost input device independent text input method. Chapter 5 describes an empirical testing environment and procedure that were used to obtain the results given in Chapter 6. Finally, in Chapter 7 a summary of the success of the new method in comparison with other existing methods is discussed along with ideas for improvement and further research.

2 Issues in Text Input

Writing in the sense of manually transferring data from the human mind to electronic gadgets has two equally important sides. The human information processing on one and the computer on the other side of the common ground of the writing interface. Both have their own characteristics which must be accommodated in a successful writing method. In this chapter we discuss some findings in psychology and linguistics along with the technical nature of the data that must be transferred through a given writing method.

2.1 The alphabet and the character sets

It should be made clear that there is a difference between an *alphabet* and a *character set* used for its visual representation. For literate people the alphabet carries much more information than can be seen on paper. The division of words into syllables and further into phonemes is intricately connected to the alphabet that a person has used to learn the language in question. This internal process that is important for understanding the language and fluent mapping between sounds, semantics and text is not necessarily connected to the shape of the alphabet.

While the claim on the alphabet having substance beyond a given visual representation may not seem clear at first, we all have observed some proof for it. Compare, for example, the shape of an upper case R and a lower case r. A person uneducated in the appearance of modern Latin alphabet would probably say that they represent different letters. However, as we know they both map to the same phoneme and should, in many cases, be understood exactly the same way. With a little imagination one can sketch an imaginary world where the phonetic and grammatical structure of English would be exactly the same, but where the visual appearance of the alphabet is totally different.

In some languages two or more separate writing systems are used. In some, like in Serbo-Croatian where Cyrillic and Latin alphabet are used, the alphabet share some characters. In others, like in Japanese the character sets are complete and independent, but are curiously enough often used together. Japanese can be, and often is, written in three different ways. The pronunciation of a word can be written using either the phoneme based Latin alphabet or the syllable based 46-character Kana script. These techniques are often used with computers because the number of Latin and Kana characters is small enough to allow construction of relatively compact keyboards and because the computer allows the correct written form of the word to be chosen from a list of words matching the entered pronunciation. The third way is to write the word level Kanji characters directly. When we consider the multitude of shapes that a word can take in Japan, we should notice that the number of different representations listed above should be multiplied at least by two because the hand-written representations may be substantially different. Thus it seems that at least some people can handle significantly more than one or two writing systems for one language.

While the interrelation of language and written text on the whole is fascinating, it is somewhat beside the point. For the purposes of this study we are only interested in the outbound translation of language into the actions necessary for inputting data that can be shown as Latin characters on a computer display.

When using a keyboard, the mapping is between language and the series of commands needed for causing the motor system to press a certain key or a combination of keys. When using a pen interface equipped with a character recognition software, one maps language into the series of commands needed for moving the pen. These two input modalities should be different enough to persuade one to believe that *any* reasonably straightforward motor activity can be used for writing after some practice. The length of the practice period and the general convenience of the motor activity is another question. The empirical part of this study tries to measure these variables for one unorthodox text input method.

2.2 The importance of the visual experience

One would expect that learning an alphabet without seeing the characters would be more difficult than doing it in the traditional way where a seen figure is copied by hand until it can be produced without a prototype for reference.

Some psychologists claim that when we read, the access to the meaning that is associated with a visual representation is more visual than phonological [Lukatela and Turvey, 1998]. The current understanding of reading process is that we have two somewhat separate mechanisms for decoding text into semantics. One is visual and the other phonological.

The phonological mechanism means first associating the text with phonemes and then accessing the meaning of the word using the sound of the word as a search key. The phonological mechanism is the one we usually learn first. The visual mechanism is learned with years of further practice. It offers a shortcut where the visual representation of a word can be used directly to access the meaning. Which of the two mechanisms dominates the reading process of a given reader reading in a given language depends on several variables. In addition to the experience of the reader, the language is important. Native English speakers have been claimed to use the visual mechanism, while native speakers of Serbo-Croatian use the phonological mechanism [Lukatela and Turvey, 1998].

It is not clear, whether a similar visual shortcut from meaning to appearance is in operation in the opposite direction when we write. However, a visual representation for a new alphabet is probably in order even if the alphabet is used only for writing and not for reading. The visual appearance can be helpful when utilizing skill transfer in teaching a new alphabet. If the new characters resemble characters known to the user, the learning process will probably benefit as long as the meanings do not conflict. Also, having a visual appearance for the characters will help to describe and discuss them in tangible terms.

2.3 Good characteristics of a handwriting alphabet

According to Ragnheidur Karlsdottir [1997] the two most important qualities of a handwriting alphabet are the attainable writing speed and the legibility of the resulting text. Legibility has a different meaning when we are developing an alphabet for text input. The computer reads the input on-line, which means that the action of writing must be standardized and identifiable by the computer while the visual appearance of the trajectories used for input is largely irrelevant. The computer can synthetise any appearance for the alphabet once it has been properly recognized. Speed, on the other hand, retains its meaning in text input and is very important.

Karlsdottir also lists some undesirable qualities for a handwriting alphabet. These are acute angles, high curvature of strokes and unnecessary connecting strokes between letters. In general this includes all trajectories that cause the pen to slow down. The slowdown may be caused by two important reasons. The more obvious one is that if the pen has to suddenly reverse its direction, the average speed of the pen will drop because the instantaneous speed will visit 0. The more intricate reason is that some strokes are more difficult to draw than others. The connecting strokes are difficult because there are so many of them and because writers do not have as much training per connection as they have for each character.

Karlsdottir studied the performance of students on grades from 3 to 6 using four different cursive writing styles. She observed that when speed was important the students tended to change their style so that no strokes were absolutely straight, but not too curved either. Acute angles were replaced with loops and some strokes were simplified by shortening or dropping some curve features. In addition she points out that the styles that are easier to keep close to the model under maximum writing speed are the ones that support a regular hand movement pattern.

2.4 Modeling writing methods

If we were to construct a new writing method, it would be appropriate to test it before putting it into use. A sensible first step could be to create a theoretical model that can be used to predict the performance of the method in human use. Models of human performance do exist.

Models like GOMS in a general form [Card *et al.*, 1983] or with more specific handling of pointing and clicking performance with Fitts' law have been used. In order to accurately predict something as complex as handwriting, these models need to be calibrated for a specific task. Thus the first thing to do is to conduct a small pilot experiment to gain some insight into the task of writing using the text input method that we are interested in. After we have some data to feed to our models, we can make predictions on user performance and build a larger, statistically significant, experiment to validate our predictions. The computed predictions alone are not enough, because the models do not tell us everything. GOMS for example does not predict initial learning rate for motor skills [John, 1995].

2.5 Skill transfer

It is a well established fact that skill acquired for one task can be used in other similar tasks. Generally, the more the tasks resemble each other, the better the skill transfer is likely to succeed.

The situation is not as simple as that when we look more closely on what exactly constitutes the "close resemblance" that ensures strong transfer. Certainly it means that the features that are critical to performing the two tasks share common characteristics in the internal representation of the tasks in the human mind and motor system. Now, however, we are getting close to the real problem. We simply do not know enough about ourselves as information processors to be able to list the critical features of a task in the order of importance [Lintern, 1991].

Thus, we are left with the choice of embarking on a long journey of empirical trial and error testing or being satisfied with using the vague notion of general similarity.

In the case of learning to write with a new alphabet, we would naturally be interested in using the considerable skill that most literate people have in working with existing alphabet. The idea has been used by developers of text input methods (for examples see Unistrokes and Graffiti in Chapter 3).

2.6 Character and digram frequencies

When constructing a writing method one has to do some compromises. A significant cause for compromises is the shortage of easy and simple characters. There is a very limited supply of simple and distinct shapes, movements or sounds that can be used in writing. One must choose which characters are associated with these easy and simple actions. To make writing method fast and robust one is inclined towards choosing the most common actions to be the simple ones. This way the average speed and ease of the writing method is closer to optimum.

The common characters can be found by running text through programs that count the desired frequencies. Computations like this have been performed by people working with natural languages, data compression, cryptology and text input methods. While previously gathered data is available, we chose to gather some more to verify earlier findings and to make a point on the varying text input tasks. The new data is gathered from two sources. The first one is project Gutenberg text collection [Gutenberg, 1999] and the second is the Linux kernel source tree that consists mainly of C-code [Red Hat, 1998]. This data is compared to the table on character and digram frequencies in English given by Soukoreff and MacKenzie [1995].

2.6.1 Character frequencies

Table 2.1 lists the top 35 characters in three text samples. Kernel stands for the Linux kernel source tree version 2.0.36 distributed with Red Hat Linux 5.2 [1998]. Gutenberg is the Project Gutenberg etext base as found in the ftp.sunet.se mirror on April 20th 1999. Three Gutenberg files having a ".txt" extension contained zeroes (integer value 0, not character "0"). This was taken as an indication of that the files were not plain text and the files were excluded from the data. Similarly no files with name extensions other than ".txt" were included. The data labeled as Soukoreff is not completely Soukoreff's and MacKenzie's original work. They only added the figures for the space character into a table gathered by Mayzner and Tresselt [1965]. The Soukoreff sample contains only 27 characters while the other two were computed over all 256 possible eight bit values.

Of the three frequency listings in Table 2.1 only Soukoreff data is claimed to be representative of English text (without punctuation, numbers and other symbols not listed in Table 2.1). The Soukoreff sample consisted of 107 199 characters.

While the Gutenberg sample is quite a bit larger consisting of 970 428 426 characters, it may not be representative of English. Firstly, because all texts were not in English. In addition to English the sample contains at least Latin, French, Italian and Spanish. Secondly, the Gutenberg data also contains several HTML-files and files consisting mainly of numbers (like pi to the millionth decimal). Thus the Gutenberg sample is more of a mix of natural and computer languages spiced up with some numeric data. The text mixture in the Gutenberg sample is closest to something that a person writing email, html-pages and spreadsheets would type into her desktop computer. The most important feature of the Gutenberg sample is the fact that it gives some idea of the frequencies of period, comma and other punctuation characters not included in Soukoreff's data.

The kernel sample is another mixture of languages. It has a significant bias towards character frequencies typical to C-code, but it also contains natural language (mostly English), gnu make compatible makefiles and small portion of other C and assembler programming related text files. In the 40 919 330 characters that were counted, non-alphabet characters are much more frequent than in the two other samples. The kernel sample represents rather closely the kind of text that a person would type when coding a portable operating system kernel.

The thing to be learned from Table 2.1 is that lower case alphabet and space are not the only frequently needed characters in text input. On the other hand they are a very good guess for the most frequent characters even in C.

2.6.2 Digram frequencies

The character frequencies should be the first to be considered when designing a writing method. The second thing to look at are the transitions between characters. As Karlsdottir's study [1997] showed, in a sequential writing method such as regular handwriting, the inter-character transitions are one of the biggest factors in writing speed and a major source for errors.

Table 2.2 lists the top 35 digrams in the same data from which Table 2.1 is derived. The fact that of the 35 top ranking digrams shown only 11 are shared by

	Kernel		Gutenberg		Soukoreff	
Rank	Character	Frequency	Character	Frequency	Character	Frequency
1	sp	0.145832	sp	0.152503	sp	0.186550
2	e	0.054217	e	0.087374	e	0.108321
3	t	0.041608	t	0.061606	t	0.079711
4	i	0.036502	a	0.054373	a	0.066101
5	lf	0.034443	o	0.053207	h	0.062808
6	r	0.032816	n	0.047330	o	0.053881
7	s	0.032546	i	0.045921	s	0.049366
8	n	0.032239	s	0.043563	n	0.048965
9	tab	0.031355	r	0.041673	r	0.047798
10	a	0.029416	h	0.041089	i	0.041987
11	0	0.028855	d	0.030009	l	0.036380
12	o	0.027240	l	0.028298	d	0.035168
13	d	0.025076	lf	0.021562	u	0.024981
14	_	0.023130	cr	0.020194	w	0.023349
15	c	0.021343	u	0.019984	m	0.020149
16	l	0.017964	c	0.017858	c	0.019151
17	f	0.017488	m	0.016829	g	0.017733
18	u	0.016313	f	0.016643	y	0.017043
19	,	0.015705	w	0.014504	f	0.014561
20	*	0.014940	p	0.013297	b	0.013218
21	p	0.014098	,	0.013222	p	0.012472
22	)	0.013029	g	0.013181	k	0.008703
23	(	0.013008	y	0.012954	v	0.008059
24	h	0.011776	.	0.011150	j	0.001296
25	m	0.011410	b	0.010172	x	0.001119
26	-	0.011201	<	0.008856	q	0.000615
27	;	0.010696	>	0.008795	z	0.000503
28	=	0.009123	v	0.006582		
29	x	0.008942	k	0.005138		
30	b	0.008808	/	0.004391		
31	g	0.008708	1	0.003691		
32	E	0.008210	I	0.003555		
33	/	0.007591	"	0.003353		
34	S	0.007483	0	0.003045		
35	1	0.007348	-	0.002898		
Sum		0.830475		0.938818		1.0

Table 2.1 Top 35 character frequencies.

all three suggests that optimizing a writing system for digrams is more difficult than it is for single characters. Furthermore, the benefit is likely to be smaller because the transitions are likely to be less complex and time consuming than the characters.

While optimizing a writing method for digrams may be difficult, it may also be worthwhile, for reasons that can be seen in figure 2.1. Digrams less frequent than the 256th most frequent are rather rare in all three examples. The 256 most frequent digrams account for 64% of the text in the kernel data, 81% in Gutenberg data, and 97% in Soukoreff's data. Thus at least in task specific writing method the set of digrams that needs to be taken into account to make the optimization worthwhile is rather small compared to the number of all possible digrams. A big problem, however, is making a universal writing method that is well suited for all writing tasks. From the data discussed above, it seems that writing tasks can be too diverse to allow significant advantages from digram optimization. If we optimize for speed in English text input, C language writing speed will probably suffer and vice versa.

Even though it seems that there is no way to make a universally optimal writing method, the character and digram frequency data is not useless. When designing a writing method for a specific task, we should generate similar figures for the expected text and optimize the method accordingly. Also, with a general purpose method we probably should try to find the most frequent writing tasks and see if the expected text allows optimization. If it seems to be possible to make a writing method better for a task without significant penalty in other tasks, one should obviously do the optimization.

2.7 Summary

When text is input into a computer the input itself does not need to be human legible. As long as the computer interprets the input correctly, any visual appearance can be generated for human viewing. However, there are other features of handwriting that are relevant to text input. Firstly, we must remember that under stress handwriting converges towards a style where no sharp corners or straight lines exist. A writing method should allow this to happen without penalty. Secondly, the alphabet used for text input should have *some* visual appearance so that it can be easily taught and discussed.

There are differences in the frequencies at which different characters appear in texts. Some characters constitute over 10% of a given text while others almost

	Kernel		Gutenberg		Soukoreff	
Rank	Character	Frequency	Character	Frequency	Character	Frequency
1	sp-sp	0.046290	e-sp	0.024210	e-sp	0.045746
2	lf-tab	0.014489	cr-lf	0.020183	sp-t	0.036492
3	tab-tab	0.012626	sp-t	0.019995	t-h	0.035205
4	0-0	0.011660	t-h	0.018517	h-e	0.029431
5	e-sp	0.010648	h-e	0.018147	d-sp	0.024505
6	i-n	0.010423	sp-a	0.014644	t-sp	0.021856
7	;-lf	0.009324	d-sp	0.013859	s-sp	0.020783
8	,-sp	0.008969	s-sp	0.012394	sp-a	0.017556
9	lf-sp	0.008284	i-n	0.011860	sp-w	0.016669
10	t-sp	0.007525	t-sp	0.011857	sp-s	0.014888
11	sp-*	0.007232	e-r	0.011415	a-n	0.014701
12	r-e	0.006916	,-sp	0.011181	r-sp	0.013834
13	d-e	0.006789	a-n	0.010802	sp-h	0.012947
14	s-t	0.006214	sp-o	0.009338	e-r	0.012257
15	e-r	0.005906	n-sp	0.009125	n-d	0.011315
16	sp-i	0.005486	r-e	0.009007	y-sp	0.010923
17	sp-t	0.005392	sp-s	0.008744	h-a	0.010858
18	n-t	0.005294	sp-w	0.008480	n-sp	0.010746
19	0-x	0.005201	sp-h	0.008319	r-e	0.010625
20	=-sp	0.005129	n-d	0.008223	o-u	0.010401
21	sp-0	0.004910	sp-sp	0.008165	i-n	0.010354
22	s-sp	0.004829	o-n	0.007448	sp-f	0.009878
23	s-e	0.004794	r-sp	0.007162	sp-b	0.009636
24	d-sp	0.004713	e-n	0.007000	sp-m	0.008171
25	sp-(	0.004539	a-t	0.006904	sp-c	0.008059
26	-->	0.004535	sp-i	0.006657	h-i	0.007686
27	*-sp	0.004530	y-sp	0.006639	o-r	0.007574
28	sp-s	0.004495	o-u	0.006632	a-r	0.007481
29	---	0.004367	e-d	0.006575	e-n	0.007453
30	o-n	0.004332	o-r	0.006458	a-t	0.007322
31	n-e	0.004323	h-a	0.006257	n-g	0.007192
32	a-t	0.004307	o-sp	0.006136	e-d	0.007154
33	sp-=	0.004242	t-o	0.005901	s-t	0.007033
34	t-e	0.004212	e-s	0.005815	sp-o	0.006725
35	t-h	0.004131	t-e	0.005674	sp-l	0.006688
Sum		0.267056		0.359723		0.500163

Table 2.2 Top 35 digram frequencies.

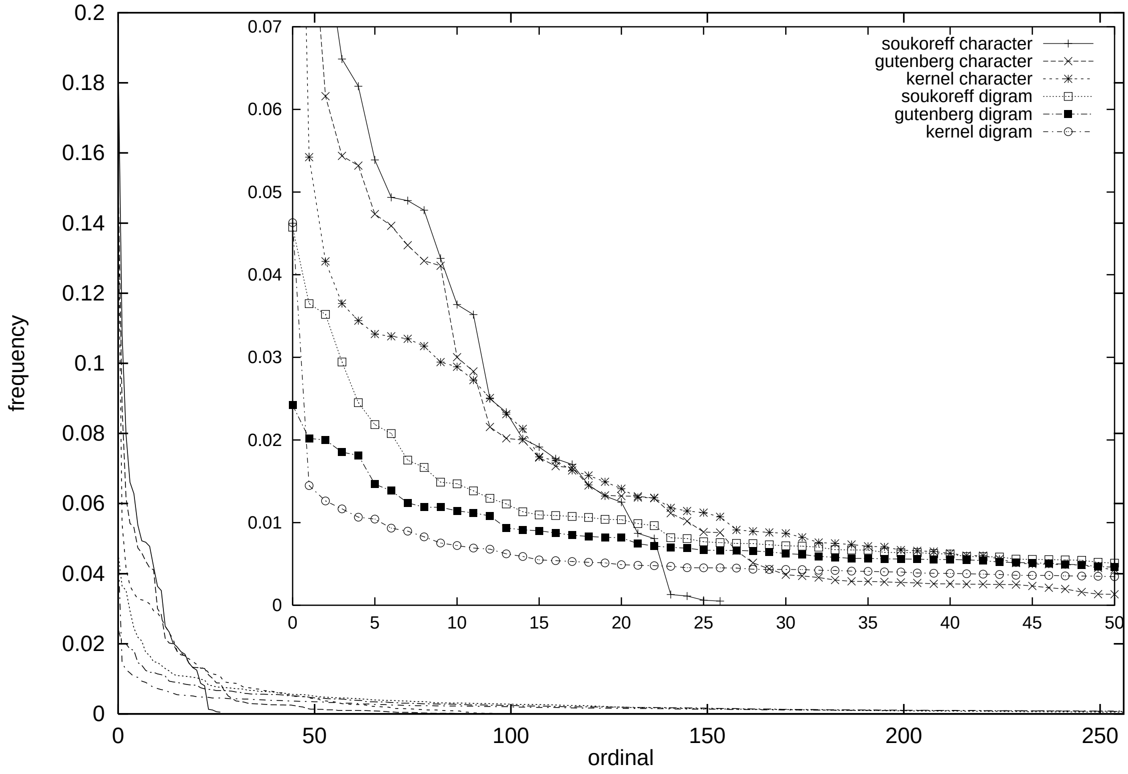

Figure 2.1 Character and digram frequencies down to the 256th most frequent item.

never appear. Similar results hold for digrams. While it is very unusual for any digram to be more frequent than 2%, the top 256 digrams cover more than half of all normal texts. When designing a writing method one should try to make the frequent characters and digrams easy and fast to write.

3 Existing Text Input Methods

Often, as is the case with the new text input method given in chapter 4, a close look at the previously existing methods shows that the ideas are not so novel but rather slight variations of preceding work.

To give the reader an idea of the multitude of methods that have been developed over the years an approximate map of the field is given in this chapter. Examples of some important text input methods are presented with short discussion about their strengths and weaknesses.

The map takes a form of a tree seen in figure 3.1. The tree has five main branches: keyboards, text recognition, unistrokes, speech recognition and gesture recognition. Each branch has a number of sub-branches of which the most relevant ones are shown in figure 3.1.

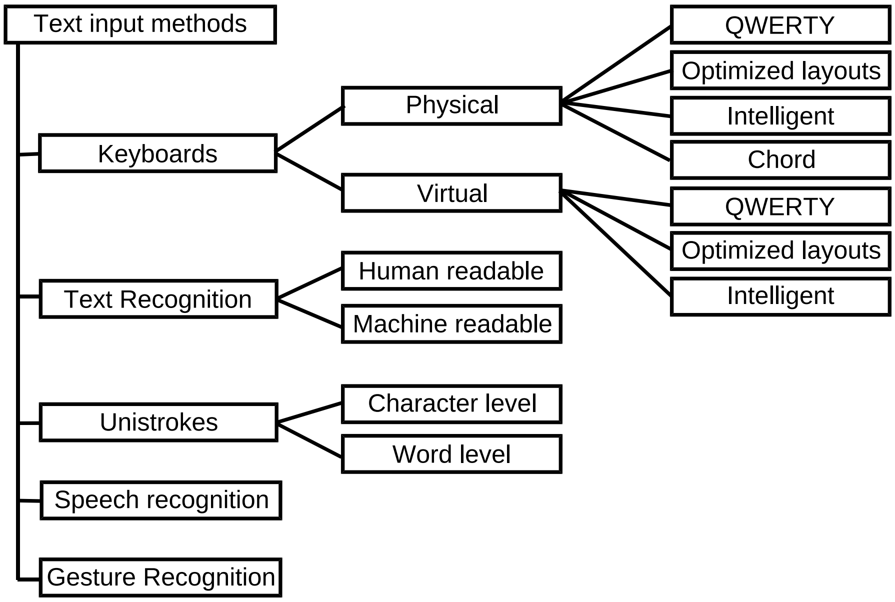

Figure 3.1 Map of text input methods.

We intentionally avoid calling the map a taxonomy. The main reason for this is that we do not have unambiguous criteria for the classification. We wanted to give a special place for text recognition as computationally expensive and algorithmically difficult method and for unistrokes as computationally simple alternative. The rest of the map is built around these two main branches so that it accommodates most of the methods that have been developed for small portable computing devices.

3.1 Keyboards

The most popular method for transferring text from human mind into the memory of the computers is the keyboard. The keyboard gained popularity and developed a large body of skilled users already before the computers invaded the offices. The QWERTY-layout that was patented in 1868 and intended to be used in mechanical railroad ticket typewriters [Barber, 1997], is still the prevalent layout in our digital PCs. Some refinements have been introduced and the appearance of the keyboard may be very varied, but QWERTY is still the most common arrangement of the alphabet keys by a huge majority. Competing arrangements have been suggested during the years, but they have not gained much popularity.

Today, as computing devices are getting ever-smaller, there is an increasing interest in new, smaller, keyboards. Simultaneously keyboards have shown yet more evidence of their attractiveness. Like many other things in our time, they have moved to virtual worlds. Of all possible text input methods virtual keyboards displayed on a touch screen seem to be the best choice in many situations [MacKenzie *et al.*, 1994].

The range of different keyboards available today is so large that it is pointless to list all the different devices here. Only a few main branches of the keyboard family tree are discussed. The two main branches are the physical and the virtual keyboards. These branch again into sub-branches (see figure 3.1). The most significant physical keyboard types are traditional QWERTY-layouts, different optimized layouts with full number of keys, intelligent keyboards with fewer keys, and chord keyboards. The virtual keyboard branch has the same sub-branches except for the chord keyboard branch.

3.1.1 Physical keyboards

A self-evident example of a traditional physical QWERTY-keyboard is the 104-key PC-keyboard. The most important optimized layout with full number of keys (i.e. at least a key for every lowercase letter in English) is the Dvorak-layout. As examples of keyboards with fewer number of keys the Half-QWERTY, FOCL, and SHK keyboards are discussed followed by three ways of inputting text with the telephone keyboard. A good example of a chord keyboard has been described by Gopher and Raij [1988].

QWERTY

The physical characteristics and operation of the QWERTY-keyboard should be familiar to everyone. Because QWERTY-typing has been a major cost generator in organizations throughout the century, it has been studied carefully. We know that QWERTY-touch-typing is difficult to master, but with the proper learning effort it is a fast and accurate method for text input in controlled office-like surroundings.

The difficulties in learning QWERTY-touch-typing may have many explanations. Three very convincing reasons given by Gopher and Raij [1988] are:

- The visual appearance of the characters has no connection to the motor activity needed for pressing the key.

- The keyboard has no structure that could be easily converted into a cognitive internalization of the keyboard layout and typing activity.

- The number of keys and especially the number of key to key transitions is simply too large to be learned swiftly.

On the other hand, once a typist has with persistent training attained a good skill in QWERTY-typing, the speed is amazing. Having separate keys for most common characters allows the typist to prepare several strokes in parallel by moving fingers to the correct locations. As soon as a key has been pressed, the finger that is left free can again be moved to a new location while typing continues elsewhere on the keyboard.

Optimized layouts

Because the QWERTY-layout was designed by a mechanical engineer mainly from the perspective of trying to make it mechanically sound, it is not optimal from the typist's point of view. Some have claimed that the QWERTY-layout was designed so that members of frequent digrams are scattered across the keyboard to avoid jamming the arms of a mechanical typewriter. Others point out that this cannot be true because the QWERTY-layout forces the typist to use more adjacent keys than a random layout [Barber, 1997]. Soles, Glidden and Soulé (the inventors of QWERTY) would have been pretty lousy engineers if QWERTY is their best effort in avoiding closely located common digrams. Regardless of the motivations and skill levels of the 19th century engineers, we can today agree on that the keys can be arranged in a way that requires far less finger travel and distributes work better to both hands than QWERTY does. The most famous of the newer layouts optimized for ten finger typing speed is the Dvorak keyboard shown in figure 3.2

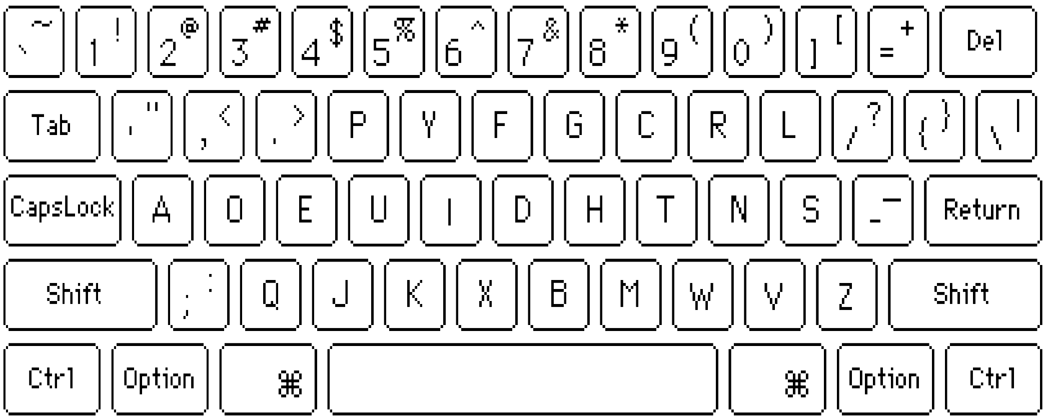

Figure 3.2 The Dvorak keyboard layout [Huerta, 1999].

The improvement in speed over QWERTY is not as great as one might expect. The main reason for this is the parallel nature of typing that was described above. The finger travel is not such a great problem, because it can occur in parallel. Better alternation between the hands may, however, help to achieve better parallelism. Significant speedup can also occur within the set of characters pressed by the same finger and in the difficulty of typing repeating characters.

Dvorak-layout does require less finger travel, and the most frequent strokes do not require as much finger acrobatics as they do with QWERTY-layout. The more natural operating posture of the hands is probably responsible for the claims that Dvorak keyboard causes fewer repetitive stress injury cases than QWERTY.

Although the Dvorak-layout is not always clearly faster, it seems to be at least slightly faster. Much of the information available is claimed to be biased by the “holy war” on keyboard superiority that has been going on since the Dvorak layout was first published.

A significant problem with the optimized layouts is that their benefits are to an extent tied to the language or task that they were optimized for. Using an optimized layout for every language could cause too many problems to be a useful solution.

A universally optimized layout could be better, but even this solution may not be significantly better than QWERTY because the advantage gained from the optimization may be in the scale of just few percent due to the counterbalancing effect of the different tasks and languages. For example the left curly bracket "{" is very rare in English, but very common in C++.

Intelligent keyboards

As the number of keys decreases, some other measures must step in to keep the input rich enough. One feature that can be introduced is a selection of operating modes. Even on the standard PC keyboard we have Caps-Lock and Num-Lock modes which are frequently used to set the keyboard into a different operating mode. With very few keys the number of modes would grow to be larger than can be conveniently handled and therefore more complex algorithms are used.

The Half-QWERTY keyboard [Matias *et al.*, 1993] shown in figure 3.3 is an example of a keyboard where the number of keys has been halved, and the number of operating modes has been increased. The keyboard can be used with either left or right hand. Both arrangements are shown in figure 3.3. The spacebar key is used for switching either the actual half of the keyboard or the mirror image of the other half active. The missing side of the keyboard is mapped as a mirror image on the existing side. Matias et al. demonstrated significant skill transfer from regular QWERTY-keyboard in learning to type with Half-QWERTY. They estimate that with adequate practice typists will achieve up to 88% of their two handed QWERTY-typing rate with Half-QWERTY.

The Single Hand Key Card (SHK) is an eighteen key keyboard designed to be used with four fingers of one hand. The card is held between the thumb and the fingers with the base of the card resting against the palm. The key arrangement is shown in figure 3.4. Most of the keys are labeled with two characters. When a user wishes to input a word she presses the keys on which the desired characters are seen. When the key-sequence for the word is finished the user presses the ambiguity resolution key (AR). The SHK support software takes the key sequence and searches a dictionary for words that can be constructed using the characters

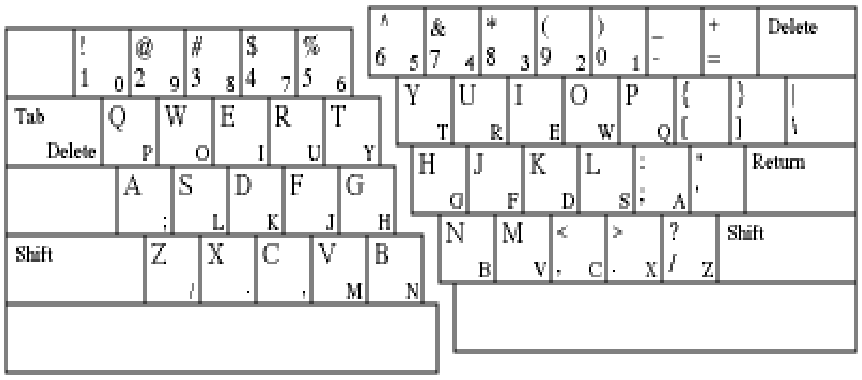

Figure 3.3 The Half-QWERTY keyboard [Matias *et al.*, 1996].

imprinted on the pressed keys. [Sugimoto and Takahashi, 1996]

Sugimoto and Takahashi do not give precise data on user performance, but they expect speeds faster than 40 words per minute (wpm) to be possible. SHK is a system for inputting English text. It does not support other languages. The keyboard and the dictionary will have to be tailored for each language. As shown in figure 3.4 SHK has a small joystick with three switches for mouse-like input. Sugimoto and Takahashi clearly tried to make SHK a complete solution for mobile interfaces. Without further testing it is hard to say whether SHK would outperform a pen interface. At least it should allow for more eyes-free operation than most pen-interfaces.

The traditional way of inputting text with a mobile phone keyboard is that each key is mapped to several characters. Pressing the key once produces the first character, pressing twice the second etc. The speed of writing is a lot slower than it is with a full sized keyboard. With a dictionary the method can be improved at least in two different ways. First, each key is mapped to several characters as before and characters are entered exactly the same way. The dictionary is scanned for words with a beginning that matches the string input so far. When the typing proceeds and the word becomes unambiguous, the rest of the word is inserted and typing proceeds from the end of the word. The second way is to

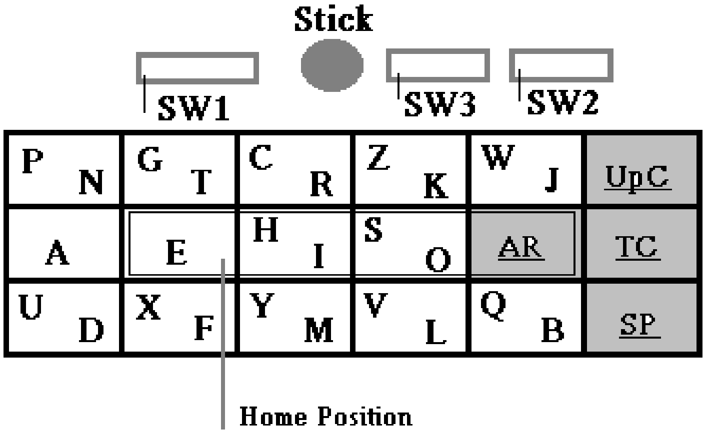

Figure 3.4 The SHK key arrangement [Sugimoto and Takahashi, 1996].

do the dictionary search with an algorithm similar to the one used in the SHK support software. Each key is pressed only once and the dictionary is searched for words that match to any sequence of characters that is mapped to the keys in question. Both of these speed-improved methods require a backup method for inputting words or character sequences not found in the dictionary.

Chord keyboards

Adding operating modes to a keyboard can be done with a state that persists until the state-inducing key is pressed again. This is the way that the Caps-Lock and Num-Lock keys on a regular keyboard work. Another way is to tie the mode to a state of a given key as is done with the Shift, Ctrl and Alt keys on the common QWERTY-keyboards. While the key is pressed, the keyboard is in a different mode than when the key is not pressed. Taking this a bit further gives us a keyboard where all keys change the mode of the keyboard. Now we can map a mode to a character and we have a chord keyboard where characters are input by pressing a combination of keys simultaneously. Doing this is not as difficult as working with the same number of "sticky" keys like Caps-Lock because the user touches each active key and thus it is harder to forget the state of the keys. A chord keyboard with eight keys has 256 different modes and can therefore be used to input any eight bit ASCII code. Further modifications of the basic idea may take into account the order in which the keys were pressed and gives even richer input with fewer keys.

The intelligent keyboards discussed above have a more complex algorithm for setting the keyboard state. In intelligent keyboards the state of the keyboard is implicit because the computer handles it automatically. With chord keyboards the large set of states is not hidden. Instead, the user explicitly chooses the keyboard state by pressing several keys simultaneously.

In their 1988 article Gopher and Raij give some of their results on experiments with a two-hand chord keyboard [1988]. Their keyboard has two similar units of five keys and an extra shift-key for the thumb. The units are mirror images of each other and each is operated with one hand. The five keys allow 31 different combinations. Each shift key gives another 31 combinations. Thus even one hand can be used to input English lower and upper case letters and there still is room for several other characters.

Gopher and Raij compared the rate of increase in typing speed in three test groups. The first group used a two-handed chord keyboard. The second group used a one-handed chord keyboard and the third group used a traditional keyboard with Hebrew layout.

They found that for the first 25 hours of practice the chord keyboards were clearly faster. After that the two-handed chord keyboard continued with faster learning rate and the one-handed chord keyboard was roughly equal to the traditional keyboard in learning speed. At the end of the 35-hour training group one was typing at an average speed of 42 words per minute. Group two reached the speed of 36 wpm and the group with the traditional keyboard finished with the speed of 24 wpm. Clearly learning was faster with the chord keyboards.

The speeds for any of the groups had by no means peaked yet. Two of the two-handed group continued up to 50 hours of practice and reached the speed of 51 wpm. One continued to 60 hours and finished with 59 wpm. However, Gopher and Raij suspect that the speed increase with the chord keyboards will level off sooner than with traditional keyboards. Thus with training period significantly longer than 35 hours the traditional keyboard will start gaining in the speed comparison eventually finishing with superior speed. The reason for this is the possibility of parallel preparation of the strokes. Chord keyboards are more serial because many consequent characters require the same keys and fingers to be used.

However, the training required for superior speeds with traditional keyboards may be too much for most users. Furthermore, the skill may be so complex that it requires constant formal training to keep. Therefore the chord keyboard could be better for casual typists. It gives good enough speed and is faster to learn. According to Gopher and Raij the chord keyboard does not exhibit negative transfer from existing typing skills because the cognitive structure of the keyboard and the typing activity is different and better suited for human capabilities.

Despite their good qualities chord keyboards have not gained much popularity outside some special areas such as mail sorting or stenography. One reason for this is that in comparison the QWERTY-keyboard seems deceivingly easy to use. Having a separate key for each character makes QWERTY look so easy that most people do not bother to train themselves in typing. Often they are not aware that their typing, despite the illusion of productivity, is in fact slow and laborous.

3.1.2 Virtual keyboards

Due to skill transfer benefits QWERTY-keyboard is popular amongst the virtual keyboards too. Layout optimization, however, is very different for virtual keyboards because touch screens often allow only one touch point at a time. The one touch point limitation slows down the typing speeds and therefore adds value to dictionary based guessing schemes that may be included in the text input system.

QWERTY

Because text input is a complex undertaking and people are slow to learn, a familiar keyboard layout gives a lot better walk-up performance for virtual keyboards than even the most optimal unfamiliar layout. This is the main reason for the fact that when we walk to a touch screen based info kiosk or vending machine, the system used for text input is most likely a virtual keyboard with QWERTY-layout. Most people today have some familiarity with the QWERTY-layout and thus it is beneficial for the manufacturers to use QWERTY. Some users might even be annoyed if the machines presented them with a new "optimal" keyboard layout.

Optimized layouts

Most touch screens and digitizing tablets allow only one point of input at a time. Thus the user cannot keep many fingers on the keys of the virtual keyboard like is customary to do in preparation for strokes with a physical keyboard. This limitation effectively reduces typing into a serial task. The parallelism that is essential for gaining great speeds with a physical keyboard does not exist and typing speeds cannot reach the magnitude that we have observed for well trained touch typists.

By killing the parallelism, the virtual keyboards give new value to finger-travel optimization. The difference in speed between an unoptimized layout such as QWERTY and a layout optimized for single-finger typing should be much greater with virtual keyboards than the difference between physical QWERTY and ten-finger optimized layout such as Dvorak.

OPTI is one of the optimized virtual keyboard layouts for the English language. Figure 3.5 shows the OPTI layout as described by MacKenzie and Zhang [1999]. The keyboard layout was optimized for speed using trial and error, Fitts' law, and character and digram frequencies in English. Fitts' law gives a function for computing the tapping time given the length of the movement needed and the width of the target thus enabling a researcher to compute a prediction for the upper bound of user performance given the keyboard layout. Trial and error is needed to generate the keyboard layouts.

Figure 3.5 The OPTI keyboard [MacKenzie and Zhang, 1999].

According to the calculations of MacKenzie and Zhang [1999] the OPTI layout is theoretically 35% faster than QWERTY and 5% faster than FITALY [Isokoski, 1998, Textware Solutions, 1998] which is another one-finger "optimized" layout. In a longitudinal study described by MacKenzie and Zhang the speed difference between OPTI and QWERTY seemed to exist in the real world too. The test group of five previously QWERTY aware students reached their QWERTY tapping rate in just ten 22 minute sessions. At the end of the 20 session experiment during which the subjects received equal amount of training in both QWERTY and OPTI tapping, OPTI was clearly faster with average speed of 45 wpm. With QWERTY-layout the group reached the speed of 40 wpm. MacKenzie's and Zhang's test subjects were instructed to aim for both speed and accuracy. Emphasis on speed may have contributed to the error rate which was over four percent for both keyboard layouts. The error rate with OPTI was consistently slightly lower than with QWERTY.

Intelligent keyboards

Fluctuating Optimal Character Layout (FOCL) is a text input method for five keys and a display. Four keys are used to move a cursor over an optimally arranged keyboard and the fifth key is pressed to select a character. Between characters the keyboard is rearranged to reflect the probabilities of the next character. The cursor is also returned to the upper left corner to be close to the most probable characters. Bellman and MacKenzie [1998] tested FOCL with 26 lower case characters and space using eleven test subjects. For comparison a traditional QWERTY-arranged virtual keyboard was used with a similar selection technique except for returning the cursor to a home position. Both arrangements performed equally well reaching a speed of 10 wpm by the tenth and last 15 minute test session.

The emphasis on FOCL is to minimize the key presses that are needed for operating the virtual keyboard. POBox is a system for doing a similar thing using a pen interface. The keyboard layout is static, but on each key press a menu with fluctuating layout is shown. The menu holds the most probable completions for the current word. A POBox keyboard with an active menu after pressing the f-key is shown in figure 3.6. When the user lifts the pen after pressing a key the menu moves below the keyboard. [Masui, 1998]

In addition to English, the POBox has been implemented for the Japanese language. In Japanese operation the keyboard consists of the 46 characters of the Hiragana phoneme alphabet. Thus ideally the user types in the beginning of the pronunciation of the word and then chooses the written form of the word (a mixture of Kanji and kana characters) from the menu as soon as it appears there. While the English writing mode may not seem very inviting, Masui claims that

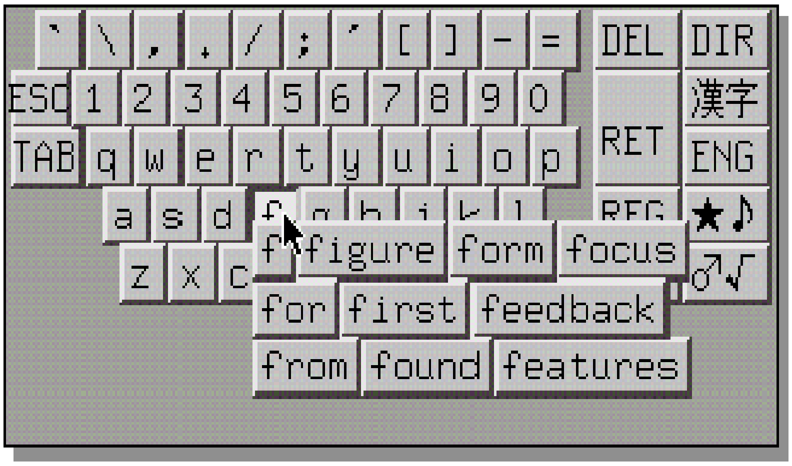

Figure 3.6 The POBox input area [Masui, 1998].

the Japanese mode is very useful because Japanese with its numerous characters and ambiguous pronunciations makes typing a slow and difficult task even with a physical keyboard. [Masui, 1998]

The problem with many intelligent keyboards, physical and virtual alike, is that they rely on visual information. Thus they cannot in general be used in eyes-free manner. Often, this is not a problem, but when eyes-free operation is needed (as often is the case in note taking), even the best method that requires visual attention becomes annoyingly clumsy.

3.2 Text recognition

A huge effort has been put into teaching the computer to recognize different character systems that have developed over the years for human use. The effort has not been unsuccessful. Today Optical Character Recognition (OCR) technology is on a level that suffices for transferring printed text from paper to digital form with only occasional need for corrections.

This is not the case with handwritten text. The Cursive script produced by most English speakers is next to incomprehensible to a computer. In general the writing systems developed for human use are more or less problematic for algorithmic recognition. However, there are writing systems that have been developed for computer use. The most familiar of these is the bar code which is routinely read by computers without any errors.

3.2.1 Human readable characters

Text recognition for computers is very different depending on whether only the end result is available or whether the computer can observe the act of writing. The kind of text recognition that OCR software does, where only the visual appearance of the text is available, is called off-line recognition. When the computer records data as writing happens we are talking about on-line recognition. In on-line recognition we can record data like pen pressure and speed. These may contain important information that can be used to make the recognition more accurate.

There is not much objective test data available on the performance of different handwriting recognition packages. Much of the data that the manufacturers have available is marketing oriented and undoubtedly shows the software in slightly better light than is believable. The manufacturers may have achieved the results in laboratory with world's best specialists teaching both the algorithm and the user, but in real world the accuracy may prove to be worse. For this reason some of the accuracy figures listed by Tappert et al. [1994] probably belong to the marketing oriented category of results. Furthermore, the best accuracy figures are typically reached by training the recognition algorithms individually for each writer in addition to requiring the writer to write each character in separate boxes using a particular writing style.

MacKenzie et al. [1994] tested the handwriting recognizer distributed with Microsoft's Pen Windows. They used non-connected lower case alphabet without user specific training. The result was an entry rate of 16.3 words per minute with 8.1% error rate.

Even the most accurate handwriting recognizer is still bound by the limitations of the traditional character sets. Top handwriting speeds for most people tend to be around 33 words per minute [MacKenzie and Zhang, 1997, Karlsdottir, 1997]. That is about half of the speed that many people can reach with touch typing on a keyboard [Card *et al.*, 1983]. Thus it is questionable whether traditional handwriting should be used for text input even if an error free recognition system could be developed.

3.2.2 Machine readable characters

Because human writing is difficult for computers to read, the computers need their own writing systems. The one that most of us see every day is the bar code system. Computers write bar codes on paper and glue these labels on various items to be read again by other computers.

Figure 3.7 shows examples of the two main types of bar codes in use today. UPC/EAN codes on the left are the familiar 1-dimensional bar codes that most manufacturers use for product identification in Europe and North America. Datamatrix and PDF417 shown on the right are 2-dimensional bar codes that allow much higher data density.

Figure 3.7 1- and 2-dimensional bar-codes [Adams, 1999].

As one probably noticed while looking at figure 3.7, bar codes are not easy for humans to read. The coding is based on the size and precise location of the elements. These are variables that a machine can easily measure, but which are almost impossible for humans to see with the required precision. The data in the codes may also be compressed and it may contain error correction information which further confuses human readers.

3.3 Unistrokes

One way to overcome some of the limitations of the Latin alphabet is to design a new alphabet. Goldberg and Richardson [1993] point to the direction of shorthand systems for valuable information in designing a new text input alphabet. The lesson learned is that for a trained writer one stroke, even a complex one, is faster than several simpler ones. For one-character-per-stroke system a touch sensitive writing tablet offers the additional bonus of explicit segmentation between characters. Armed with this information Goldberg and Richardson set out to design the Unistroke[1] alphabet which can be seen as a forefather of a whole family of unistroke text input methods that have appeared lately.

Today some unistroke text input methods have moved beyond the original one character per stroke rule. Therefore, the systems will be discussed in two groups: character-level unistroke systems and word-level unistroke systems.

3.3.1 Character-level

The design of character-level unistroke characters is a compromise between stroke simplicity and easy learnability. If we want easy to learn strokes, we should use ones that strongly resemble regular handwriting. If on the other hand we want to emphasize simple mechanical and algorithmic efficiency we should use the simplest strokes that are recognizable by our recognition algorithm. T-Cube has the simplest strokes and longest learning time for eyes-free operation, Graffiti is slow to use, but very fast to learn [MacKenzie and Zhang, 1997]. Unistrokes are somewhere in between.

Unistroke

Goldberg and Richardson [1993] claim that traditional handwriting recognition is like "hunt-and-peck" typing and propose that a touch-typing like alternative should be available for expert users. Their candidate for this touch-typing method is the Unistroke alphabet shown in figure 3.8. As seen in figure 3.8 the more common characters such as e, a, t, i and r are mapped to the simplest possible strokes and the less frequent ones are a little slower to draw. Goldberg and Richardson

[1] To avoid confusion, I will use capital U when referring to the Unistroke alphabet described in the original paper [Goldberg and Richardson, 1993]. Unistroke without capital U refers to unistrokes in general.

report writing speeds up to twice as fast as others [MacKenzie *et al.*, 1994] have measured for handwriting recognition with regular alphabet.

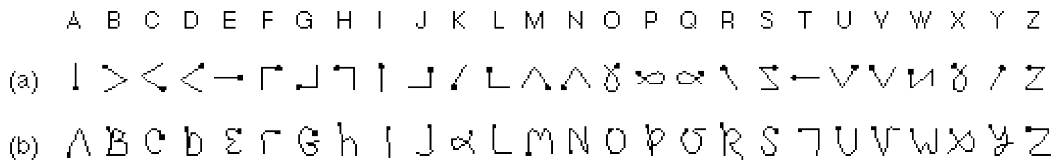

Figure 3.8 Unistroke (a) and Graffiti (b) unistroke alphabet [MacKenzie and Zhang, 1997].

Graffiti

Graffiti is a handwriting alphabet developed by Palm Computing[2] for their PDA product series known as Palm Pilots. From the user's point of view Graffiti resembles unistrokes in the sense that characters are generally drawn in one stroke. Strictly speaking Graffiti does not completely follow the unistroke idea of one stroke per character. For example x can be drawn as two separate strokes and characters like ä and ö must be drawn in two parts.

As seen in figure 3.8 Graffiti strokes are more complex than Unistroke strokes. The added complexity makes the recognition algorithm more challenging to program and potentially slower and more inaccurate. The work of the recognition algorithm is even more difficult because Graffiti alphabet resembles the regular Latin albhabet very closely. Some of the problems with ambiguous Latin script have been avoided. For example ambiguity between 0 an o is avoided by writing numbers on different area. However some characters remain very close to each other. For example H and L are easily mixed if the horizontal part of L is even slightly curved.

Despite the obvious problems with speed and ambiguity the Pilots and Graffiti along with them have been a commercial success. This should always be noted

[2] Later bought by USRobotics and currently a part of 3Com.

before claiming Graffiti to be inferior to other methods. In other words, although Graffiti is not perfect it works well enough and shows that users may value easy learnability over eventual expert performance.

T-Cube

Marking menus are a system where instead of drawing the menubar at the top of the window a radial (pie) menu can be used on top of the working area of the window. The menu is activated by clicking and holding for a short period of time (0.3 s). The selection is done by moving the pen (or mouse or whatever the pointing device in use happens to be) over the desired menu item and lifting the pen (or mouse button). When multi-level menus are used the sub-menu is displayed when the user moves over the item that has the sub-menu. Thus choosing an item from a multi-level menu is a continuous stroke with sharp corners between the levels of the menu. Users can learn to draw these strokes without seeing the menu and achieve significantly better speeds than with traditional menus. [Kurtenbach and Buxton, 1993]

Marking menus can have varying number of items per level and varying depths. Kurtenbach and Buxton measured user performance using menus with breadths of 4, 8 and 12 combined with depths from 1 to 4 [1993]. They found that the response time of the user increases linearly with increasing depth. The response time is also greater when using menus with more items per level. In terms of speed it does not matter much whether we use a menu with three levels of four items, two levels of eight items or two levels of twelve items. However with depths greater than one the error rate increases dramatically when we move from four items per level to eight or twelve items per level.

The error rate for four levels of four items per level is around 5% while the error rate for one level of eight items per level is already in the same range. With more than one level and 8 or 12 items per level the error rate grows quickly above 10%.

Kurtenbach and Buxton also compared mouse and pen as input devices. Their results show that mouse and pen were roughly equal in speed and error rate when using four items per level with mouse being slightly slower and more error prone. With more items per level the mouse was clearly inferior.

All in all, it seems that to achieve the speed and accuracy needed for text input one should stay below four items per menu and use a pen if possible.

We have not encountered reports on using Marking menus directly for text input. However, a slightly modified implementation exists. This system is called T-Cube and is shown in figure 3.9. The difference to Marking menus is that the initial menu is always shown. Input is initiated by touching one of the slices of the radial menu. The touch causes the second level of the menu to be displayed offset so that it never is hidden by the user's hand on a touch-screen. A character is chosen by moving the pen to the direction of the desired slice in the second level menu. The second level menu is displayed just to remind the writer of its layout and the proper direction for the stroke. The stroke is drawn starting from the chosen slice of the first level menu to the direction shown in the second level menu. T-cube uses 9 items in the first level menu and 8 in the second. Thus the user can choose from 81 items by tapping on a slice or performing a simple flick gesture. [Venolia and Neiberg, 1994]

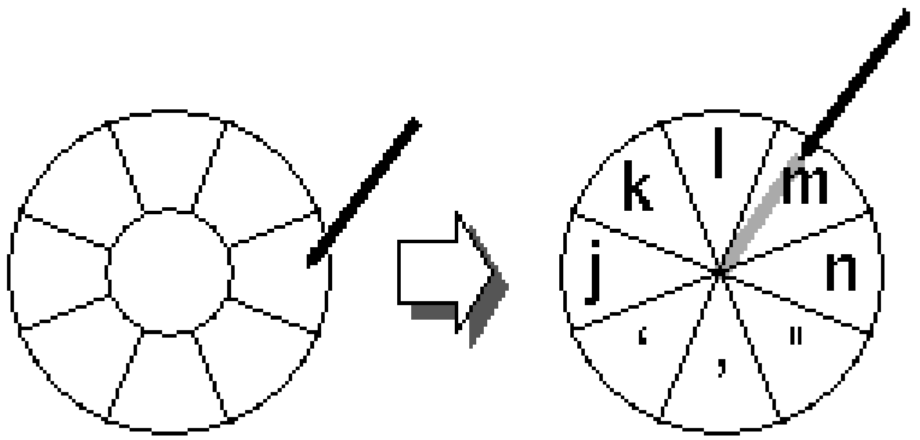

Figure 3.9 T-Cube [Venolia and Neiberg, 1994].

The input used in T-Cube is very close to the minimum. One cannot make pen input much simpler than a tap or a flick. However, T-cube suffers from the limitations of virtual keyboards. It does not allow eyes-free operation because choosing accurately from the 9 initial slices is very difficult without seeing the menu.

The learning path of T-Cube is also keyboard-like. A novice can use "hunt-and-peck" like operation by looking for the desired character in the menus and then choosing it. An expert user will learn to choose from the second level menu without looking and potentially achieve very high speeds. Venolia's and Neiberg's results [1994] show that learning to be an expert user with T-Cube is a long undertaking. None of their 11 subjects showed clear signs of decay in speed improvement during the course of four to nine learning session. The fastest user reached the speed of 20 wpm while the average was around 16 wpm. These speeds are already comparable to hand printing.

Unfortunately Venolia and Neiberg do not give figures for error rate. We can assume that the error rate is comparable to what others have found on 8-item pie menus [Kurtenbach and Buxton, 1993, Balakrishnan and Patel, 1998]. That means that T-Cube error rate is within or close to the range of 5% to 7.3%

3.3.2 Word-level

While character level unistrokes make the design of handwriting recognizers a lot easier with their explicit segmentation and better differentiation of character forms, they also make the writer lift the pen between every character. Undoubtedly writing could be more efficient if the pen could be kept down all the time like most of us do when we write fast using our own variations of cursive Latin script.

Two new systems allowing words or even whole texts to be written with a single gesture were introduced in 1998. The first one called Cirrin [Mankoff and Abowd, 1998], while probably usable, does not offer great improvements in speed or accuracy, but the second one, Quikwriting [Perlin, 1998], is very promising.

Cirrin

Figure 3.10 shows the input area of the Cirrin text input method [Mankoff and Abowd, 1998]. Cirrin is meant to be used with a stylus. One puts the stylus down inside the ring and then moves it over the areas labeled with the characters. Input is generated from the coordinates of the points where the pen crosses the circumference of the inner circle.

Only lowercase alphabet are shown along the circumference in figure 3.10. Putting all 128 characters of the ASCII character set or the thousands in Unicode is clearly not an option. The slices would become too narrow to hit with the pen. The authors suggest using the non-dominant hand to input the characters not found in the Cirrin input area. The device operated with the non-dominant

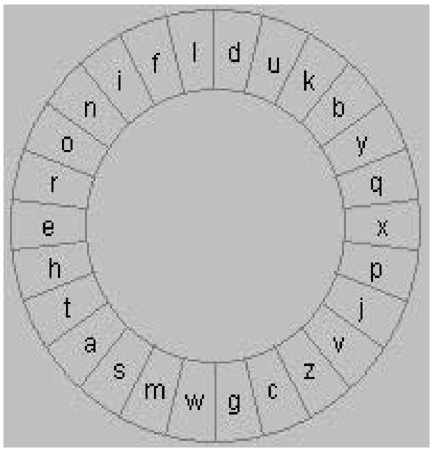

Figure 3.10 Cirrin [Mankoff and Abowd, 1998].

hand may be chosen freely. Mankoff and Abowd have used a regular QWERTY-keyboard. They also discuss using a regular handwriting recognizer for the task. In a mobile setting a QWERTY-keyboard is not an option. A handwriting recognizer might be workable, but that raises the question of whether Cirrin offers enough benefits to be worth the screen real estate if we will have a handwriting recognizer anyway.

While we have found no thorough test reports on Cirrin, it seems that the accuracy could be good enough for text input. The speed may prove to be a bottleneck since visual feedback is needed in order to hit the relatively small areas with the pen. Cirrin is very dependent on the pen-interface and offers no eyes-free operation.

Quikwriting

Quikwriting is a text input method for stylus-based user interfaces. Figure 3.11 shows two quikwriting input areas with a stylus trace. The trace on the left shows the gesture needed for inputting an f. The trace on the right shows a gesture that results in the word “the” being input.

The method works by dividing the input area into 9 zones in a 3x3 grid formation. The zones are numbered 1 through 9. The central zone (5) is the “home” zone from which each stroke starts and onto which all strokes end. Although the zones can be thought to reside in a grid formation, they are not necessarily rectangular. In fact the most functional layout is likely to resemble the Cirrin

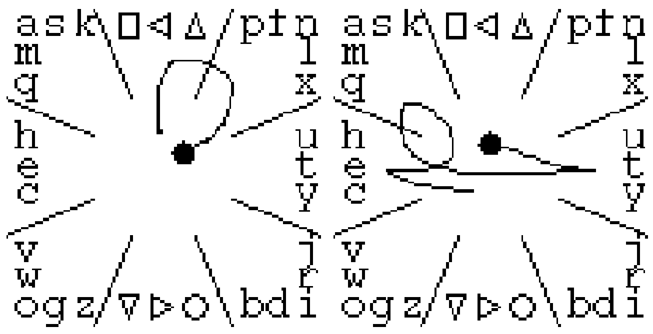

Figure 3.11 Quikwriting [Perlin, 1998].

layout shown in figure 3.10 with only eight slices in the ring. A character is drawn as a loop that crosses one or several of the zones surrounding the home zone. Quikwriting recognizes the character from a sequence of zone changes. The changes that are taken into account are the ones from or to the home zone. For example the f in figure 3.11 is chosen because the pen leaves from zone 5 to zone 3 and returns from zone 2.

Because the space is limited on the Quikwriting input area, all characters cannot be input using this simple method. The method chosen for expanding the input space is to have different modes of the input area. The implementation that Perlin describes in [1998] has four modes between which the user can switch by inputting a "shift" character. One shift character causes the next character to be chosen from the character set associated with the shift character in question. Two consequent shift characters "lock" the mode so that several characters in sequence can be chosen from the character set associated with the "locked" shift. The shift characters are shown in figure 3.11 as up-arrow, rectangle and circle.

Proper test data on the speed and accuracy of Quikwriting is thus far unavailable. In the original paper Perlin claims that users have achieved speeds up to three times as fast as with Graffiti while maintaining a comparable error rate [1998]. However, in a recent posting to the quickwrite-talk mailing list Perlin announced new anecdotical information which suggests that Quikwriting and Graffiti are equally fast, but that Quikwriting is more comfortable to use [1999].

Quikwriting is a very promising system for pen interfaces because it guides the user with a visual interface and allows relatively high speeds for expert users. Whether Quikwriting is usable in other than pen interfaces remains to be seen. It may be possible for experienced users to draw Quikwriting gestures without visual feedback or with tactile feedback only.

3.4 Speech recognition

Producing a speech recognizer that is speaker independent and can recognize connected words with high enough accuracy in a suboptimal sound environment has proved to be a very difficult task. To be as good as a keyboard the speech recognizer should recognize all languages that can be written in the Latin alphabet. Currently such speech recognizers do not exist [Campbell, 1999]. However, the speech recognizer does not have to be perfect to be usable. Already, a recognizer that can speaker-dependently recognize words from a vocabulary of ten can be useful in a phone that can dial numbers that it hears. In a personal device such as cell-phone speaker dependence may be a good property. It prevents others from using the phone in a meaningful way. However, in other purposes such as public phones, light switches, locks and automated-teller machines speaker independence is a requirement that cannot be dropped.

However, to be good for a general text input method speech recognition has to develop. There still are languages that can be written with a keyboard, but do not have a functional speech recognition software. Also, recognizers that are considered to be close to the state-of-the-art today do not work well enough with all speakers [Barber, 1997].

The problems with speech recognizers are similar to the ones encountered in online handwriting recognition. Speech, like handwriting, does not have the kind of a structure that computers can easily handle. Speech has ambiguous parts and human speakers tend to be rather sloppy with pronunciation. Of course speakers can be taught to speak so that the recognizer can understand them, but if we are going to teach the users, why not teach them touch typing?

3.5 Gesture recognition

Gesture recognition is a term that covers a wide range of various techniques. For this study it is enough to make a distinction between two groups of gesture recognition systems. The first one is the rather traditional convention of using gestures drawn with a stylus on the sensing surface of a pen based user interface. These gestures are used to accomplish various tasks that might require several key presses on a keyboard or a sequence of menu selections when using a traditional windowed GUI. Thus, gestures do not directly compete with text input methods in pen interfaces. However, they can replace some text input in some situations. The second group of gesture recognition systems are those that track human gestures such as hand movements in three dimensional space using techniques like data-gloves and machine vision.

3.5.1 Pen interfaces

Traditional windowed user interfaces can be used with touch screens or digitizing tablets and styli. An interface used with a pen gives a more direct feeling than the same interface used with a mouse. The indirectness arising from the use of menus may also start to seem unnecessary. Many menu commands can be replaced with gestures that are drawn on top of the object that is being manipulated. Prototypes of such systems have been tested at least on spreadsheets [Burnett and Gottfried, 1998, Watson, 1993], word processing applications [Watson, 1993], and air traffic control workstations [Chatty and Lecoanet, 1996].

Gestures can partly replace text input in some situations. However, the gesture sets seem to be application specific. This means that gestures needed in word processing and spreadsheets may be different from each other, but most certainly are different from gestures needed in a graphics package. Gestures may be more intuitive to use than written commands, but they do cause cognitive load that is partly avoided by using menus that can easily be searched for the right command. Moreover, gestures cannot completely replace menus for giving commands or general text input methods for writing. Thus, it seems that gestures may be suitable for expert use, but a well known text input method is still needed for novice users.

3.5.2 3-D gestures

Three dimensional gestures can be used in a way that is similar to the use of gestures in a pen interface. The three dimensional gestures simply replace menu items or function keys. However, three dimensional hand gestures can be used for text input too. Hearing impaired people have for a long time used sign language as a communications medium. With modern sensors such as data-gloves and video cameras we can measure the movements well enough to recognize them.

Researchers at Hitachi Central Laboratory have reported some success in recognizing Japanese Sign Language using data-gloves [Sagawa *et al.*, 1997]. The vocabulary was very limited and their recognition algorithm was trained with data from the same speaker that was used for testing the recognition accuracy. However their results show that sign language can be computer recognizable at least in some situations.

3.6 Comparison of speeds and error rates

Because speed and error rate are the most important variables of a text input method, we should have some knowledge on the relative speeds of different methods. Table 3.1 contains a collection of speeds and error rates that were available in the papers listed in the references section. The unit used for the speed is words per minute (wpm). One word per minute equals to five characters per minute in most of the sources cited in Table 3.1. One cannot be very specific about the wpm unit because in handwriting the space character does not need to be written whereas in typing the spacebar must be pressed. All papers did not report whether the wpm figure includes spaces or not. Where speeds were given in different units they have been converted to wpm assuming that five characters per word includes spaces.

The speeds and error rates are not directly comparable due to various reasons. The original authors may have used different procedures for measuring the speeds. Some of the procedures were aimed to find the maximum speed of a method while others concentrated on accuracy. Some studies were longitudinal and thus the test subjects were trained in using the methods. Other studies measured walk-up performance with no or very little training.

The speed for POBox is given as Japanese Kanji characters. Kanji is a word level pictogram character set and thus each character has roughly the same expressive power as an English word. Within the English language this would mean that one Kanji character per minute equals to one wpm. In Table 3.1 the comparison between POBox and other methods is done across languages, but still the speed estimate for POBox is close enough given the imprecise nature of the whole table.

When a measured speed or an estimate for an average trained writer was given, it was used. In other cases we had to estimate the speed from the maximum and minimum speeds given. Most experiments did not continue long enough to get a precise measurements for truly experienced writers. A good example of the huge

Method	wpm	errors	source
Physical QWERTY	64	3,5%	[Matias *et al.*, 1993]
Virtual OPTI	45	4,18%	[MacKenzie and Zhang, 1999]
Two-Handed chord keyboard	42	<2%	[Gopher and Raij, 1988]
POBox	41		[Masui, 1998]
Half-QWERTY	40	7,4%	[Matias *et al.*, 1993]
Unistroke	37		[Goldberg and Richardson, 1993]
Quikwriting	28		[Perlin, 1999]
Graffiti	28	<1%	[MacKenzie and Zhang, 1997]
Virtual QWERTY	23	0,6%	[MacKenzie *et al.*, 1994]
Cirrin	20		[Mankoff and Abowd, 1998]
T-Cube	16		[Venolia and Neiberg, 1994]
Microsoft handwriting recognizer	16	1-20%	[MacKenzie *et al.*, 1994]
Virtual ABC keyboard	13	1,1%	[MacKenzie *et al.*, 1994]
FOCL	10		[Bellman and MacKenzie, 1998]

Table 3.1 Approximate speeds and error rates for some text input methods.

variation caused by personal abilities is the range of physical QWERTY-typing speeds. Many people, like the author, consider themselves to be reasonably able typists while their typing speed is in the range of 40-60 wpm. Some professional typists reach speeds up to 150 wpm and beyond being three times as fast. QWERTY-typing has possibly the longest range of speeds because the learning path is so long. Serial typing as performed with the virtual keyboards probably has smaller variation of speeds because it offers a shorter learning path and the upper bound for the speed is limited by human motor capabilities. Multi-finger typing may be limited more by cognitive capabilities related to the parallelization of the finger movements than by the motor ability to speedily move one's fingers.

On the whole the numbers in Table 3.1 are not accurate, but do, however, give the approximate range in which the speeds vary. Furthermore while the speed order may not be correct for two methods very close in the table, two methods further apart are likely to be in the correct order. Also, based on the information in Table 3.1 we can conclude that in general over 40 wpm is good performance and below 20 is poor in comparison to other available methods.

3.7 Summary

With all the methods discussed above and the dozens that were not discussed it is hard to see which are the ones that have potential for developing to be the writing methods of the future. Goldberg and Goodisman suggested in their article that we should not be satisfied if we can do the same things with a computer that we have done for millenia using pen and paper [1991]. We should look for ways of doing new things and doing old things better instead of doing the same things in new ways that are not intrinsically any better than the old ways. Thus one criteria for serious evaluation of a writing method is whether it is significantly faster, easier or more reliable than the old ones. If it is not, it probably is not worth investing into.

Some systems have been tested more carefully than others. These well investigated ones include handwriting recognition, speech recognition, and traditional keyboards. The good and bad sides of these systems are mostly known. This may be part of the reason why the new and relatively unknown handwriting methods like Quikwriting and Unistrokes seem to be very promising. Undoubtedly careful investigation will show faults in them that have not been mentioned thus far. Some of these faults may be bad enough to render the methods useless.

Regardless of which system proves to be the best, the reality today is such that the technically superior system does not always prevail. Marketing, politics and prejudices may change the situation so that technically bad systems become standard.

4 A Minimal Device-Independent Text Input Method

In this chapter a new text input method called Minimal Device-Independent Text Input Method (MDTIM) is introduced. It draws heavily on some of the designs discussed above. At least Unistrokes, word level unistrokes, T-Cube and Marking Menus have had their impact on the design. However, the resulting combination is yet to be found by the author in existing literature.

First the reasons for the word minimal in the name of the method are explained. Then a general tour of the method is given. The rest of the chapter will discuss some design decisions in detail concentrating on the implications of the unistroke capability, the choice of the code-tree, and input device specific details encountered in the implementation of the method.

4.1 Minimalism

Currently computing systems have many different input devices. There is a multitude of different keyboards, mice, touchpads, touch screens, pens, trackballs, joysticks ranging all the way to brainwave analysators and other rather exotic apparatuses. This long list of different devices makes one fear that if used for text input they would each require a different method. Users could probably learn to use all methods, but it would be a great burden for them. A burden to the user should always be considered a defeat for user interface design.

Maybe one cannot design a method that works *exactly* the same way on all input devices. After all, the devices *are* different. However, skill transfer makes it beneficial to use a method that resembles in some critical respects another method already known to the user. A good example of the power of transfer effect is the success of the QWERTY-keyboard layout in virtual keyboards. Methods that allow for a significant skill transfer between different input devices are a desirable path of development.

In search for a common input method one should probably look at the common features of all input devices. If they have a rich enough set of common features, the next step is to find out whether this set of features can be used for text input. The other side of using only features common to all input devices is that the set of features may turn out to be rather small. However, following this idea will lead to a text input method that works on all input devices. To ensure that it will work on all future input devices as well, one should make only minimal assumptions about the capabilities of the input device. Thus the guiding principle in the design should be to make do with as little as possible - i.e. minimalism.

4.2 The method in general

Many input devices allow input of at least four separate entities. On a joystick these are the four principal directions. Accordingly the four entities used for input will henceforth be called directions regardless of the input device and the actual data that is fed to the computer. In the following discussion the directions are called North, East, South and West (or N, E, S and W) just like in the compass.

On a keyboard any four keys can be used for inputting the directions. On all mouse-like two-dimensional pointing devices the mapping is also easily achieved. Thus we have a set of entities that can be input with many, if not all, input devices. What remains to be done in order to turn this revelation into a text input method is the design of usable coding of text with the four entities we have.

Mathematically an efficient coding would be something like Huffman-code. That is, an uneven-length prefix code with minimal number of bits per alphabet. We would construct the code using character frequencies computed from the expected input. The resulting code would guarantee that minimal number of directions would be needed for average input. However, this straightforward mathematical approach needs some refinements in order to be usable.

Firstly, if we want to maintain character-level unistroke capability, we cannot allow a direction to repeat consecutively. Thus, NN, EE, SS and WW are illegal combinations if we wish to be able to draw a letter without lifting the pen (assuming an input device that deals with drawing and pens). Furthermore, if we wish to maintain multi-character unistroke capability, we cannot have any letter's code starting with the same direction that another letter's code ends with. I.e. if we have a letter code NWS, we cannot have letter code SNE or any other letter code that begins with S. A general multi-character unistroke capability would mean that on a pen-interface-like situation the user never needs to lift the stylus. The price to be paid for general multi-character unistroke capability is thinning of the coding tree. With four directions we quickly find so many branches illegal that the letter-codes will grow much longer than they would without unistroke capability.

Secondly, we may not want to keep the code strictly optimized for the number of directions needed for the expected text since this may result in letter codes that cause the user either cognitive or motor difficulties. In other words, we may want to use transfer effect in teaching the new alphabet as was done with Unistrokes and Graffiti, or we may want to avoid direction combinations that are difficult to input using some input devices.

Device specific measurements on user performance do exist. Venolia and Neiberg measured speed differences in simple flick gestures using a pen interface [1994]. Goldberg and Richardson [1993] present some data on the speed differences in drawing the Unistroke alphabet using a similar pen interface. A mouse mounted touch-pad has also been used to measure user performance while inputting gestures very much like the MDTIM alphabet with maximum length of two directions [Balakrishnan and Patel, 1998]. The results show significant differences in the difficulty of various gestures. It is, however, hard to tell how much of this is due to cognitive and how much is due to motor difficulties. Furthermore, it remains unclear how much of the motor difficulties depend on the input device used. MDTIM should be device-independent and thus design decisions cannot be based on experiences on one or few devices only. A thorough psychomotoric study involving a large number of input devices might reveal whether the differences found in device specific studies can be generalized.

4.3 Unistroke-capability and segmentation

Determining whether we really need unistroke capability and exactly which codes are cognitively or motorically difficult, cannot be done without some hard data. As described above acquiring this data requires an effort that far exceeds the time frame of the current study. Therefore, the design decisions on these two points are currently based on slightly educated guessing as follows.

From the user's perspective the character-level unistroke capability seems to be a good thing to have on a pen-interface. Repeating directions would also be somewhat challenging to extract from the data that some input devices such as mouse-like pointer devices and joysticks generate. For example, how do we detect the change from the first N to the second N when all we get is a lengthy stroke to the general direction of N. If we were to measure the length of the stroke, we would force the user to coordinate her movements much more carefully. This seems like a bad thing to require from the user.

The word-level unistroke capability is a good thing as well, but we can live without it. However, a compromise is also available. We will allow multi-character unistrokes as long as they do not contain repeating directions. With some luck the whole word can be input without any conflicts. In many cases a word can be input in two pieces and more segments are needed only rarely.

Along with the lack of general multi-character unistroke capability goes the need for explicit user initiated or implicit system generated segmentation of the input stream needed to mark a repeating direction. When we have to repeat a direction, we may require the user to lift the pen, click on a button or give us some other signal or we can generate the repeating direction in software. The user-initiated segmentation was chosen for the empirical tests reported later in this document.

4.4 Mid-character correction

Because repeating directions are illegal within characters, they can be used for something else. A good use is a character level undo or mid-character correction capability. This is an important feature because unlike in a keyboard, one character is composed of more than one part and the parts are input at different points of time. The user may notice that she is writing a wrong character after some directions have already been input. In this situation the user should be able to restart drawing the character from scratch. MDTIM uses a repeated direction as a token that means that the user wishes to undo the current character.

4.5 The character codes

As described above, the code used for text input should be the best approximation of a statistically optimized prefix-code that is easy for the user to learn, remember and draw. Character-level unistroke capability requires that no child in the coding tree may have the same name as its parent (for example an “N” parent may not have an “N” child). Thus an inner node in the coding tree may not have more than three children. In the relaxed form of multi-character unistroke capability that was chosen, the root node may have four children, and all three leaves are allowed.

The coding tree could be drawn to illustrate the code. The tree is a useful concept if a detailed proof or description of the algorithm is needed. However, the user is unlikely to find the tree helpful when writing with MDTIM. Therefore, two other kinds of visual representations are used. First, in text it is sometimes useful to refer to the characters with the corresponding direction strings. Thus an “a” may be written as NSW. This representation is enough to describe the action needed for inputting the characters. However, it does require some decoding for novice users. Therefore, another representation is used for reference cards and visual feedback during training. Examples of this representation are shown in figure 4.1.

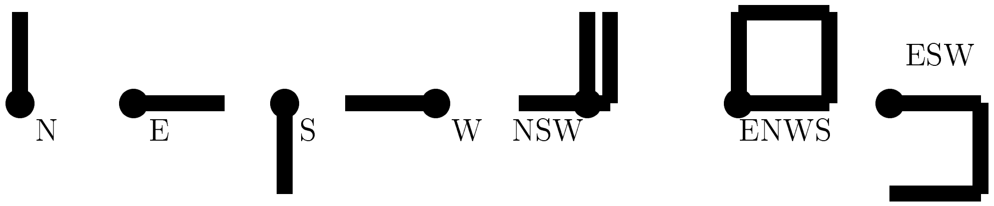

Figure 4.1 MDTIM visualizations

As seen in figure 4.1 a character begins with a dot. A thick line continues from the center of the dot to the first direction. Each subsequent direction continues from where the previous one ended except when two subsequent directions are opposites (NS, SN, EW, or WE). In this case the starting point of the last direction in the pair is offset slightly to right for NS or SN pairs and down for EW or WE pairs. The effect can be seen in the NSW character in figure 4.1. The only ambiguity problem in this visual representation is a four direction loop like ENWS which becomes indistinguishable from NESW. The ambiguity holds only for the static visualization. Dynamic visualization and the online recognition situation have a time component that clearly distinguishes between opposite four direction loops.

Now that we have a general understanding of the tools that are at our disposal, we can start working on the actual coding. Which direction string should match which character code? For speed and general economy in input we should find the most frequent character codes and map them to slightly shorter direction strings than the less frequent ones. But what exactly is a slightly shorter direction string and how long is a long direction string? To answer these questions we have to return to the tree representation of our input space.

Table 4.1 lists the maximum number of MDTIM codes for direction densities from 3 to 8 and code lengths from 1 direction to 5 directions. Because we are not allowed to repeat directions, two direction MDTIM is pretty useless as the code length is exactly the same as the tree height. More than eight directions is too much because the error rate grows unacceptable for text input (see section 3.3.1, [Kurtenbach and Buxton, 1993] or [Balakrishnan and Patel, 1998]). Although the decision about using four directions for MDTIM is based on different reasons (such as error rate, minimalism and skill transfer from compass and general human tendency to prefer right angles), the code length table shows that four directions is not such a bad choice. It gives a reasonable number of codes with relatively short code length.

	number of directions					
length	**3**	**4**	**5**	**6**	**7**	**8**
1	3	4	5	6	7	8
2	6	12	20	30	42	56
3	12	36	80	150	252	392
4	24	108	320	750	1512	2744
5	48	342	1280	3750	9072	19208

Table 4.1 Maximum number of codes as a product of number of directions and tree height.

Before we can say how many codes we actually need, we need to make some design decisions. First the handling of uppercase letters. In MDTIM the uppercase characters have the same codes as the lowercase characters. A modifier is used to choose between the two. With many input devices the modifier is a button which is pressed once during the drawing of a character to choose the uppercase character. The procedure is very similar to using the shift key on a QWERTY keyboard. Another conceivable modifier would be a special code such as the different "shift" gestures that are used in Graffiti.

Having the same code for upper and lowercase letters makes learning MDTIM easier because fewer codes need to be memorized. The Second choice is whether we want to use the modifiers for other purposes as well. In MDTIM we use this feature sparingly for character pairs that are strongly associated and thus hopefully easily remembered. These pairs include characters like parentheses, ' and ", - and +, and comma and semicolon. The goal is to keep the method minimal and having too many special functions makes the system complex rather than simple. Whether these pairs are easy enough to remember remains to be seen.

Table 4.1 shows that using four directions with code length of four we have a maximum of 108 codes to use for the lowercase alphabet and all other characters that we wish to use in our writing. Is this enough? The 102-key QWERTY keyboard has 56 different keys in the biggest keyblock that is used for most writing tasks. Some of these have two or more functions attached to them, but having the uppercase letters accessible with a modifier, we have enough codes for writing English language. Actually we have enough codes to give some characters shorter codes. Note, however, that one code with a length of two will consume a branch from the coding tree that would allow nine codes with length of four. Therefore, we must use the short codes very sparingly[1].

The character frequencies that we need can be extracted from the data discussed in chapter 2. The data is not representative of all languages and writing tasks. Therefore we do not claim that the resulting character codes would be the best possible. Rather, the code given in this study is an ad-hoc construction created to quickly evaluate the MDTIM concept. Especially the digram frequencies were given very little consideration. Nine of the top 35 digrams shown in figure 2.2 cause a word-level unistroke conflict (repeating direction) in both Kernel and Gutenberg data. Only seven of the 35 digrams listed for the Soukoreff data cause a conflict, but still better digram organization could be possible. We decided that it was more important to get characters like space, a, r, and n forms that resemble the Latin alphabet than to make rapid multi-word unistrokes possible.

However, some ideas that are important parts of a universal text input method are represented in the code. Firstly, we reserve space for language specific characters because we do not want to have a completely different code for different languages. We do not want separate language specific character sets because we need a good international walk-up performance so that MDTIM can be embedded, in objects and interfaces that must be immediately usable by all people. Clearly, this approach overlooks languages that do not use the Latin alphabet. However, as rude as it may sound today, we believe that in the future languages such as Chinese, Japanese and Korean that use very many characters will sooner or later start using the Latin alphabet or some slight modification as an official writing method. Already today every student of English language in those coun-

[1] A special version of Huffman's algorithm or some other algorithm for generating optimal prefix codes would do this automatically. However, because we have rejected automatic code generation, we must manually keep the code close to optimum.

tries has to learn the Latin alphabet anyway. As the use of English language increases, the question of abandoning the problematic native character set must rise sooner or later.

Most of the variation within languages that use the basic Latin alphabet amended with special characters such as the French á, â, ç, ë, é, è, ê, ï, ô, œ, ü, ù, and û can be accommodated either by including a special language specific code area or by having operations that allow the construction of the special characters of components. In our code characters å and ä, which are two of the special characters used in Finnish, both occupy a level three branch as place holders for more numerous language specific characters in other languages.

Table 4.2 lists the direction strings for the lowercase characters of the Finnish alphabet and the basic control characters. See appendix A for a more complete listing of the MDTIM codes used in the tests discussed in chapter 5.

Character	Direction string	Character	Direction String
a	NSW	p	WNEN
b	SEW	q	WSES
c	ESW	r	WSN
d	SWE	s	ESE
e	WES	t	SNE
f	ESNE	u	SEN
g	ESNS	v	WNWS
h	WSWS	w	WNWN
i	WNS	x	SWSN
j	SESW	y	SWSE
k	WSWE	z	WSWS
l	SNS	å	WEN
m	WSWN	ä	NSE
n	NSN	ö	WNES
o	WSEN		
return	SEN	backspace	NW
space	NE		

Table 4.2 MDTIM direction strings for a core set of characters.

We see that most characters were coded with four directions, few with three, and two with only two. The justification for the two short codes is as follows.

Table 2.1 shows that space is the most common character. Therefore if we will give a very short code for some characters, space should be included. The number of backspaces is clearly underrepresented in Table 2.1 because backspace does not appear at all in regular texts. However, backspace is used a lot in writing. With keyboards most corrections are done using it. While MDTIM is not a keyboard, the backspace concept is too good not to be used in MDTIM too. A 6 percent error rate would bring backspace to the second place in the ranking in Table 2.1. While 6 percent error rate is unacceptably high for expert performance, it may not be uncommon among beginners. If corrections are difficult to perform, the users may judge MDTIM difficult and unappealing to use. Therefore, we want to give backspace a short and fast code.

4.6 Compatible input devices

Now that we have described the method in general, we will show how it can be fitted for use with some popular input devices. The devices we will discuss are mouse, touchpad, trackball, joystick and keyboard. This list reflects the functionality that we implemented for the test software and is more due to easy availability and simple implementation than a result of careful consideration. However, one of these devices can be fitted into almost any imaginable computing device and therefore if MDTIM proves to be functional with all these devices, it is a good candidate for a universal text input method.

4.6.1 Mouse

Mouse and all devices capable of generating mouse-like input are handled in the same way. This simplifies the implementation (the standard mouse drivers can be used) and gives a chance to test the hypothesis about reaching device-independence through minimalism. Which, of course, has already been proven partially true because all devices have already been fitted together with the mouse driver.

Dragging has been found to be more difficult than merely pointing with the mouse [MacKenzie *et al.*, 1991]. For this reason we do not want to require the user to use the mouse buttons more often than is necessary. Therefore the segmentation needed for producing repeating directions is done with a timer that fires after 70 ms of inactivity. This means that when the user wishes to repeat a direction, she needs to stop inputting directions for 70 ms and then input the last direction again. With a mouse this means stopping the mouse for a short period of time and then continuing the interrupted move. A delay of 70 ms was chosen with trial and error. It felt right for the author, who at that time was relatively untrained in MDTIM. The sample rate of the mouse dictates the minimum value for the timeout. Usually the sample rate is around 30 samples/second and thus the timeout should be longer than 34ms. Some hardware, such as the CH Track-Ball Pro model for PS/2 connector that was used in our experiment have a faster sample rate and thus allow shorter timeout which may be useful for fast writers.

When the MDTIM recognition algorithm receives a timer signal, it resets the flag that instructs it to ignore repeating directions. When the movement continues the recognition engine will receive more coordinates and recognize the direction. It will then append the direction to the current direction string. If the direction is a repeat, the algorithm will fail to recognize the direction string as either legal partial character or legal complete character and reset the current direction string thus accomplishing mid-character undo.

4.6.2 Touchpads, touch-screens, pressure sensitive pens and digitizing tablets

Touchpads, touch-screens, pressure sensitive pens and digitizing tablets usually offer an operating mode in which they can be used as mouse substitutes. Thus they appear to MDTIM as mice. Repeating direction is generated with the timer method described in section 4.6.1.

While many devices offer an operating mode in which they produce a three degree of freedom input (x, y, pressure), we chose to ignore the pressure coordinate. There are reasons for this besides the simplicity of implementation.

1. Minimalism suggests to use as simple input as possible.

2. Some other input devices have two degrees of freedom. Thus user interface compatibility requires that we only use two degrees of freedom.

3. In some conditions, such as under vibration in a moving car, or with half-frozen fingers in the winter, the pressure component will be too inaccurate to be useful.

As has been demonstrated [Goldberg and Richardson, 1993, Venolia and Neiberg, 1994], unistroke characters can be explicitly segmented using the pressure component of the digitizing tablets. Therefore ignoring the information on whether the pen, finger, or other appendage used for writing is touching the surface or not means giving up on information that some researchers have considered very significant for the success of a text input method. Examples of systems built around the touch information are all unistroke character sets and some virtual keyboards.

We hope that the word-level unistroke capability and device-independence will compensate for the potential weakness introduced by ignoring significant components of input on some devices.

4.6.3 Trackball

A big trackball should be very much like touchpad as an input device for MDTIM except for the inertia of the ball. I.e., the only difference is that instead of sliding the finger on a surface, the surface (the ball) moves along with the finger. The posture of the hand and the movements themselves are very similar. A small trackball on the other hand behaves like a small joystick except that the ball is not self-centering and thus the writer needs to lift her finger when it hits the ball casing. Thus, it seems that trackballs can be used for input with MDTIM. They exhibit only slightly different interface and it is interesting to see whether the differences make trackball better or worse than other similar devices. The handling of the input is done as described in section 4.6.1.

4.6.4 Joysticks

A joystick is a device with a stick upon which the user may exert some force. The force or in many cases the position of the spring loaded stick is measured and translated into a digital input. Digital joysticks usually offer the capability to feed eight different inputs to the system. Their analog counterparts can be used to input a point in an n x n matrix with a realistic resolution for n well above 10.

For the purposes of this text input system we reduced (in software) all types of joysticks to simple four-switch devices capable of inputting the four directions. Figure 4.2 shows the input matrix of an analog joystick. The areas that cause a direction to be generated are marked with the initial of the direction (N, E, S and W). A direction is generated when the stick is tilted over the area. A new direction cannot be generated before the stick has visited the shaded area in the center.

The shaded square (c) in the middle and the slices extending to the corners of the area (d) are dead zones inside which the input is ignored. Both dead zones exist to stabilize the input. Typical joysticks are not very accurate devices. The input may jump around up to several percent of the whole range. Therefore when the stick is in the middle or tilted to approximately in the direction of the diagonal the actual input generated may jump around on both sides of the diagonal. This would produce a rapid burst of directions which would ruin all but the most skilled attempts of inputting anything sensible.

A joystick offers a very natural way for explicit segmentation between directions. Most joysticks are self centering and therefore returning to the center position between directions is almost automatic.

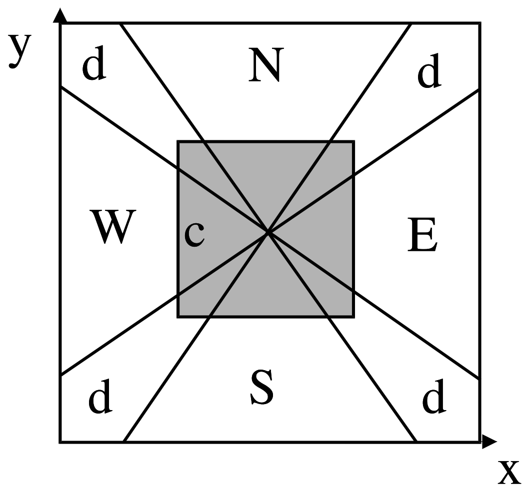

Figure 4.2 The input space of an analog joystick.

4.6.5 Keyboard

A keyboard with a minimum of five keys can be used as a MDTIM input device. Using only four keys of a standard PC keyboard for MDTIM input is probably not a very smart move outside laboratory experiments. Instead one might imagine using a small keyboard with fewer keys. Such keyboards can be found in pagers, remote control devices and mobile telephones. Some of these special-purpose keyboards already have a direction control device similar to the direction control hat found in gamepads or four separate keys in a similar arrangement. The output of these keyboards, however, is keyboard-like. The keys produce unique keycodes when pressed or depressed or both. Four of these keycodes can be mapped directly to the four directions. There are no ambiguous diagonal or center areas and no need for filtering of minute movements. Repeating a direction means simply pressing a key twice.

4.7 Learnability

As the developers of existing input methods have pointed out [Venolia and Neiberg, 1994, Kurtenbach and Buxton, 1993], a method that wishes to allow speedy input should offer the user an easy way to get started and room for learning seamlessly as she writes. A classic example of this is the learning curve with a QWERTY keyboard. A new user can "hunt and peck" and with experience move to two-handed operation and with some more formal training to touch typing.

MDTIM like Unistrokes does not allow as easy start as one would wish. The alphabet has to be learned in order to start writing. However, the growth path should be there. For example, with a touchpad a user will probably start by writing one stroke at a time, lifting her finger between every stroke while planning for the next one. As she grows more confident with the new character set, she will probably start drawing one character with each stroke. With even more practice word-level unistrokes may become feasible.

MDTIM is very forgiving with the timing of the strokes. At any point the user may take an indefinite amount of time to ponder what to do next. Due to using only four directions, MDTIM is also forgiving with the direction of the strokes. The figures for response time and error rate should follow those measured by Kurtenbach and Buxton for marking menus and thus be in favor of four directions (see section 3.3.1 or [Kurtenbach and Buxton, 1993] for details). In MDTIM the length of the strokes beyond the required minimum is completely irrelevant. Together with the easy mid-character correction capability these qualities should make the user at ease while learning MDTIM and allow the development of a personal "handwriting" style that can deviate rather far from the model characters without degrading the recognition accuracy.

Figure 4.3 shows a set of strokes that would be interpreted as NSW when fed to MDTIM. The large difference in the size of the different stroke patterns may be due to amplification generated by the mouse driver. This is, however, of no consequence since only small movements are filtered out. After the movement reaches the size (speed) that passes the filter, further growing does not hurt.

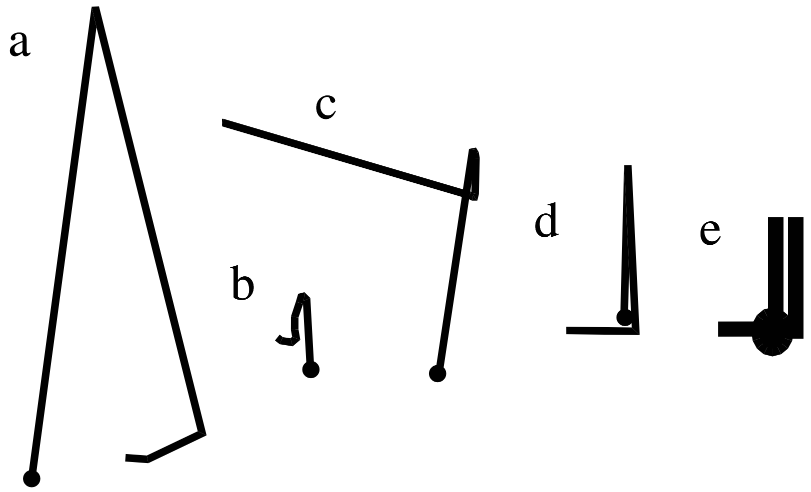

Figure 4.3 Four strokes interpreted as NSW (a-d) and the canonical representation (e).

4.8 Summary

We have described a device-independent text input method that utilises four principal directions to construct prefix codes that have been associated with characters. The method gives the writer a lot of freedom. There is no time pressure on the writer. She can draw the directions separately or a character at a time and often a string of characters with a single stroke. The speed and size of the characters may be freely chosen as long as a minimum threshold is exceeded. Unfortunately the users have to learn the prefix codes in order to use the method with any kind of speed.

We chose to use only a very limited input that can be generated with very many different input devices. This has the obvious drawback that almost all devices have more advanced features that might allow much faster, easier or less error prone ways to write. The advantage of our approach is the ability to use the same writing method with all devices.

5 Design of the Tests

In this chapter we will describe the experiment that was conducted to evaluate MDTIM. The goal of the experiment was to generate some hard data on the speed, error rate and typical learning rate of MDTIM. We also tested various input devices to see whether MDTIM writing skill gained on one device transfers to the others.

5.1 Input devices

Five different input devices were used. For most of the experiment we used Cirque EasyCat touchpad. To see whether MDTIM writing skill is device-independent we used four other input devices. These were Gravis PC Gamepad, Microsoft IntelliMouse, CH Products Trackball pro trackball and a regular QWERTY keyboard.

Next we give more details on the input devices and the way they were used in the experiment.

5.1.1 EasyCat touchpad

The Cirque EasyCat touchpad is shown in figure 5.1. The EasyCat was chosen because it is small enough (6,75 x 8,5 x 1,25 cm) to be conveniently held in one hand while the fingers of the other hand are used for input.

The EasyCat has two buttons that can be used as mouse buttons, but it can also emulate the mouse buttons through tapping the surface. Tap on the brighter colored area in the upper right corner of the active surface emulates the second mouse button. A tap anywhere else emulates the first mouse button. Dragging with button emulation happens through a double tap where the finger is left down on the second tap. EasyCat acts as though the first button was pressed until the finger is lifted again. If the finger drifts close to the edge of the touchpad during a drag, the finger may be lifted and placed elsewhere on the pad without breaking the dragging mode. These standard features of the EasyCat work conveniently with MDTIM and thus we did not bother to disable them. The dragging mode allows the user to utilize a kind of CapsLock feature. This is convenient if the user holds the touchpad in an orientation that does not allow pressing the buttons easily.

EasyCat is designed to be used while resting on a desk. In our test, however, the touchpad was held in the non-dominant hand to simulate using a small portable device.

Figure 5.1 Cirque EasyCat Touchpad [Cirque, 1999].

5.1.2 Gravis PC gamepad

Gravis PC Gamepad is a small handheld game controller with four buttons and one tilting eight-direction control hat with a detachable stick. The Gamepad can be turned around and has a switch that reverses the controls for left handed users. In our test the stick was attached to the direction control hat. And thus the gamepad was actually a joystick.

Like the touchpad, the gamepad was also held in the hands of the user. The most common style adopted by the users was to envelope the gamepad in both hands and use the dominant hand to control the joystick while the thumb of the non-dominant hand was held over the buttons.

5.1.3 Microsoft IntelliMouse

Microsoft IntelliMouse has two buttons and the scroll-wheel that has recently become a standard feature in most new mice. We however needed only the old standard mouse functionality with one button.

The dominant hand was used to operate the mouse. A mouse does not suit well for use without a desktop and thus it was used on a desk.

5.1.4 CH Products Trackball Pro

As seen in figure 5.2 the Trackball Pro is a typical trackball construction with a big ball mounted on a box with buttons. The buttons are located symmetrically and their functions can be reversed so that the trackball is equally suited for left and right handed operation.

Figure 5.2 CH Products Trackball Pro [CH Products, 1999].

While some trackballs are small enough for desk-free operation, the Trackball Pro is not one of them. Thus the trackball experiments were done while the trackball rested on a desktop. Again, the dominant hand was used for operating the trackball.

5.1.5 QWERTY keyboard

The only decision with the standard PC keyboard was which five keys to use for MDTIM. We chose the arrow keys that are in the inverted-T formation between the main key block and the numeric keypad. For the modifier key we used the right shift.

The keyboard was yet another desk-bound device. Like with the other devices, the dominant hand was used for operating the keyboard.

5.2 Software

For gathering data on the performance of users writing with MDTIM we wrote a program that implements MDTIM and logs the relevant events so that they can be analyzed later. The test program was written in C++ and it uses the Win32 APIs directly. We used the mouse driver supplied with the Microsoft IntelliMouse for all mouse-like devices. For the remaining two devices we used the drivers included in the Windows-95 distribution.

Figure 5.3 shows the interface that was visible to the test subjects during the test sessions. The largest window within the main window is a "working space" where the software displays the current MDTIM character during the input. Above the large child window we have two smaller windows. The upper of these displays the phrase that the user is supposed to write using MDTIM. The lower window shows what the user has written. The tall window on the right shows the tail of the data written to the log file.

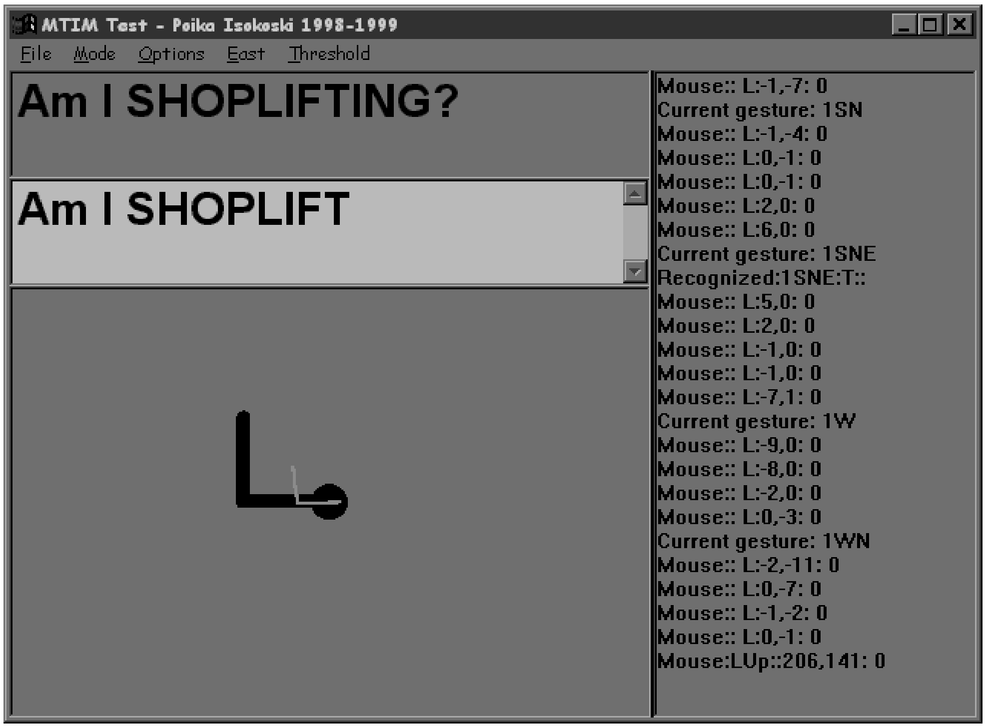

Figure 5.3 The MDTIM test program.

As can be seen in figure 5.3 the user can see both the actual input and MDTIM interpretation of that input. The actual input is the thin line and the MDTIM interpretation is the thick black direction sequence that we are already familiar with from chapter 4.

Although the idea was to test MDTIM in a form as simple as possible without any fancy signal processing, dictionary support for recognition, or other external means of improving the performance, some measures were necessary to make it usable at all. First the joystick data is gathered with some filtering as discussed in chapter 4. Second, the mouse data needed some filtering too. A minimum threshold was set for interpreting the difference between two consecutive coordinates to be a direction. Without filtering out the tiny movements the result would be an uncontrolled stream of directions as soon as the user touches the touchpad or mouse. When the acceleration was set close to minimum in the mouse driver settings, a good threshold seemed to be around three pixels. This filtered out most of the small unintentional movements while reacting soon enough when the user actually tried to do something.

While MDTIM may allow eyes-free operation with some input devices, this is not the case for novice users using a touchpad. Thus, as described above, the software gives visual feedback. While the benefits of auditory feedback may not be as clear as they are with the visual feedback, the software also emits a click through the Windows' sound system when a character is recognized.

5.3 Procedure

5.3.1 Goals

The experiment had two main goals. First, it was a demonstration of the MDTIM. Secondly, it was a longitudinal study that had two goals. Firstly, the rate of learning typical to MDTIM was be determined. Secondly, we attempted to gain a good estimate on expert speed and accuracy.

5.3.2 Test subjects

Because the experiment had a longitudinal nature, the number of sessions per subject was rather large. Given limited resources this means that the number of subjects had to be rather small. Acquiring unlimited resources for this project would not have been a good idea either given the inexperience of the experimenter. The result might have been a big mess instead of a controlled experiment.

Five volunteers were chosen from the staff of the Department of mathematics, statistics and philosophy in the University of Tampere. All were between 23 and

29 years of age. Our sample was clearly biased towards the best young brains in town at least in terms of mathematical ability. Clearly a sample like this cannot be claimed to represent the population in general. However, we did not have much choice. Ten sessions, preferably on different days during a two-week period, is too much to ask from a complete stranger without any compensation. Thus we chose to accept the warped sample to get the test done at all. Because the test subjects worked in the same building, the test sessions at the beginning or end of the day were easy to arrange.

5.3.3 Sessions

The first and last sessions lasted approximately one hour and the eight sessions in between lasted slightly over 30 minutes. All sessions had a 30 minute MDTIM practice session using a Cirque EasyCat touchpad. During the first session typing and handwriting speeds for all subjects were measured. During the last session MDTIM skill transfer test was done using Gravis Gamepad joystick, Microsoft Intellimouse mouse, arrow keys of a QWERTY-keyboard, and a CH Products Trackball pro trackball. Each input device was used for five minutes without any formal familiarization.

We tried to schedule the sessions for consecutive days. However to accommodate the test subject's summer holiday schedule we sometimes conducted two sessions each day with a minimum of five hours between the sessions. Weekends caused a two day break. Few one day breaks were also allowed if the test subjects had more important things to do.

5.3.4 Session 1

The test subjects were told that the goal of the experiment is to evaluate the writing method, not to test the subjects. They were also told that they may stop the experiment at any time if they for any reason wish to do so.

The first measured task was the typing and handwriting speed test. All subjects were presented with the beginning of the test phrase file in one window and were asked to type as much of it as possible into another window in five minutes. They were asked to avoid or correct all errors while writing as fast as they could.

The handwriting speed was measured using the same data, paper, and a ballpoint pen. This time the subjects were asked to write as fast as possible while keeping the text legible by most people. They were especially reminded to write all characters including punctuation.

Two subjects started with the typing test and three with the handwriting test. The purpose of these tests was to get a rough idea of the writing speed of the subject using two familiar methods for comparison with MDTIM.

Before starting the first 30-minute MDTIM practice session, the subjects were told and asked to try out the following features of the touchpad-MDTIM combination.

1. Tapping causes the next recognized character to be upper case.
2. Double-tap with finger left down on the second tap causes a Caps-lock effect.
3. The left button on the touchpad can be used for the upper-case function too.
4. MDTIM recognizes movements only if they are large and fast enough. Slowly moving the finger on the pad does not cause MDTIM direction input.
5. Beyond the minimum threshold the length or speed of the strokes do not matter.
6. For best performance fairly large and fast strokes should be used.
7. MDTIM characters never have repeating directions.
8. Repeating directions can be used for mid-character correction by waiting for the 70 ms timeout to occur before repeating the direction.
9. The backspace character can be used to erase text.

Additionally the visual representation of the characters in the reference chart was explained until the subjects seemed to understand it.

The rest of the session 1 was just like sessions 2-9.

5.3.5 Sessions 2-9

The subjects were given the MDTIM reference chart (appendix B) and asked to start writing. They were also told that they should complete each phrase as fast and with as few errors as possible and that they should only rest between phrases. All questions were answered, but the test supervisor did not interfere in any other way. Each training session lasted roughly 30 minutes including all rest periods between phrases.

5.3.6 Session 10

Session 10 began with the already familiar 30 minute training session. After that the users went through four five minute sessions using different input devices. The order of the devices was balanced to avoid bias due to fatigue or undesired skill transfer between the new devices.

5.3.7 Test data

For the MDTIM practice session all 451 phrases shorter than 30 characters were extracted from the fortune cookie databases delivered with RedHat Linux 5.2. These phrases were presented to the subjects in random order. The phrases contained both upper and lower case characters of the Latin alphabet along with numbers, most types of parentheses, and other characters found on standard keyboards.

Often researchers are satisfied with a small subset of characters such as lower or upper case alphabet and space (for an example see MacKenzie's paper [1999]). While this approach does give a good basis for comparing the input methods, the input speeds and especially the learning rates are unrealistically good. The non-alphabet characters may be rare, but if it takes the user a minute to figure how to input the @ character (or to deduce that it cannot be written at all), the average input speed may be reduced significantly. The more important result of a larger character set may become evident in the speed of learning and ability to retain the learned character sets. MacKenzie and Zhang recognize this fact in their 1997 paper on Graffiti [1997]. For these reasons we chose not to artificially limit the size of the character set. The test data is *exactly* what we found in the fortune databases.

Using a wider set of characters is expected to make the results look bad in comparison to figures measured for other writing method using fewer characters. However, having many characters including many relatively rare characters probably brings the measured speeds and learning rates closer to what we can expect in real-world situations. Therefore, we consider the wider character set justified even at the cost of some difficulty in comparing the results with figures measured for other writing methods.

5.3.8 Recorded data

During the MDTIM practice sessions most relevant events such as mouse, keyboard, and joystick input, recognized MDTIM directions, completed test phrases, etc. were timestamped and recorded into a file. The timer used has a resolution of one millisecond. However, because the events were timestamped when Windows 95's message passing architecture had delivered the messages to the logging procedure rather than when they were generated, the accuracy of the timer is not the limiting factor. Queuing delays greater than 50 ms should be rather rare and thus the events happened between 0 and 50 ms before the recorded time. These log files allow us to extract almost any features of the writing activity.

In addition to the automatic recording of events, we used a notepad and a pen to record user remarks and behavior during the tests.

6 Results

In this chapter we present the results from the experiment that was described in chapter 5. We begin with the speed and accuracy that was measured for our test subjects. We then give a prediction for expert speed and accuracy. Our experience with five different input devices is also discussed. After a short note on the correlation between handwriting, typing and MDTIM writing speeds, we will finish with an evaluation of the ad-hoc MDTIM character set that was used in the experiment.

The software used in the experiment timestamps and saves events such as mouse movements, key presses, direction and character recognitions, and phrase completions. The resulting files can then be examined after the experiment. We used several Perl scripts to condense the information before feeding it to statistics and spreadsheet software which produced the numbers that are discussed below.

6.1 Measured speed

6.1.1 Procedure for computing the speed

For speed computations the first character in each phrase was excluded because the test subjects were allowed to rest between phrases and thus the mental preparation time for the first character cannot be measured. Time measured for each character begins when the previous character is recognized and ends when the character in question is recognized. Thus it includes both the time spent in cognitive work of fetching the character from memory and the time spent on actually drawing the character.

All speeds are given in words per minute. One word per minute is equal to five characters. Space, backspace and in general all recognized characters are included except where something else is explicitly stated.

6.1.2 Average speed

Figure 6.1 shows the mean speed computed over the ten half-hour sessions excluding rest periods between the phrases. We have two different measures for the speed. The upper curve in figure 6.1 shows the speed that includes all characters that were used to correct errors. The lower curve gives the productive speed, which is the number of words per minute that the test subjects considered to be correct enough without further corrections. An error includes a wrong character and a backspace to correct it. Thus the difference between the curves in figure

6.1 divided by two gives one measure for the error rate. The errors are discussed in more detail later.

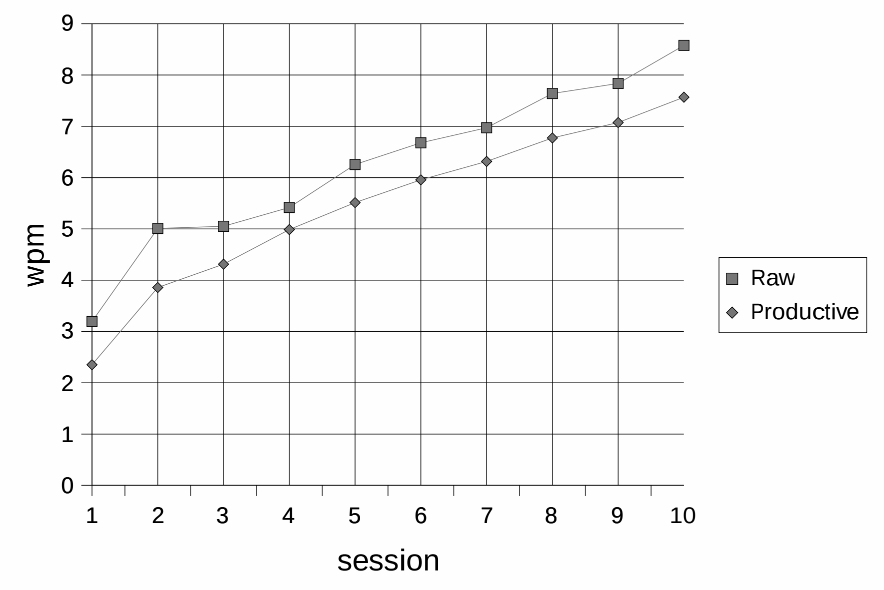

Figure 6.1 Mean raw and productive speeds.

The speed curves in figure 6.1 show two features that seem common to most people. During the first two sessions learning is very fast. This fast initial learning happened because people simultaneously got familiar with the touchpad and learned the character locations on the reference chart. After the second session the writers started to rely more and more on their memory and the learning continued at a near linear rate of about 1/2 wpm per session with slight degradation towards the end of the experiment.

Figure 6.2 shows the learning curves for each test subject. The variation between the subjects is rather large. By the last session the two fastest ones (1 and 3) were more than twice as fast as the slowest one (4).

Subject four had huge difficulties with the touchpad. Either she had very clumsy hands, or the shape of her fingertip was such that the pad could not accurately trace its movements. Whatever the reason, subject 4 never got familiar with the touchpad and thus could not concentrate on memorizing the characters. It is unclear how large portion of the population has similar difficulties and whether there are ways to avoid these difficulties with training or different construction of the pad.

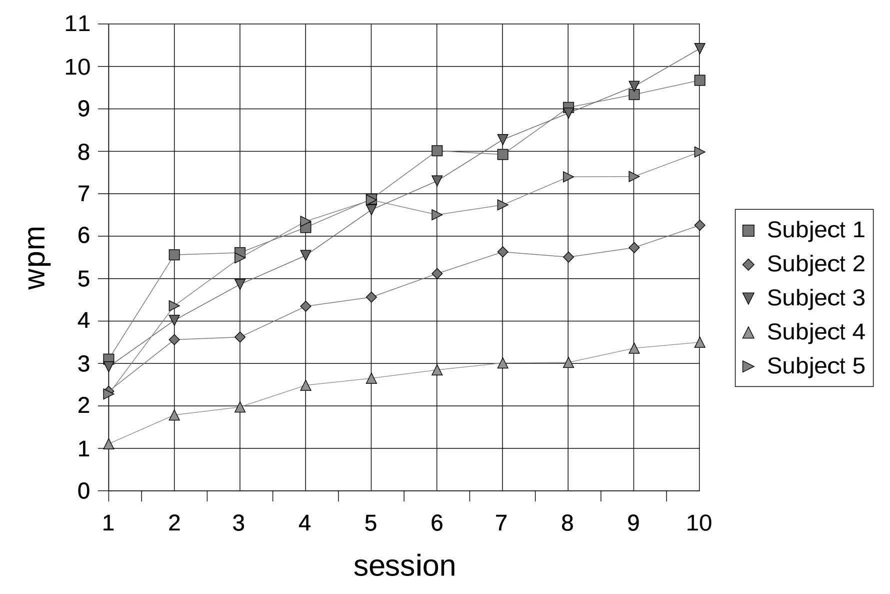

Figure 6.2 Productive wpm rate for each test subject.

In addition to finger-touchpad incompatibility, motivation may be another major factor in MDTIM learning speed. The two fastest writers in our test were also clearly enthusiastic about their task. They took it as a challenge and wanted to write fast. Although each test subject was, for the duration of the experiment, kept unaware of the performance of others, these two considered themselves to be in competitive situation. They attacked the test software with the kind of concentration and competitiveness that can sometimes be observed in computer gaming sessions. This kind of behavior was not encouraged in any way except for listing speed of learning as one of the variables that would be measured.

The other test subjects were in no way reluctant to work through all of the 10 sessions, but they did not exhibit the kind of dedication that was evident in subjects 1 and 3.

The writing speeds shown in figures 6.1 and 6.2 are mean speeds computed over all of the active writing time during the 30 minute session. During this time the writer may have very fast and very slow periods depending on her alertness and the character mix in the text. To give an idea of this variation, we computed the speed within one minute window every six seconds over the entire experiment. Figure 6.3 shows the result of this computation for subjects 3 and 4. The experiment seems to have lasted longer for subject 4. One reason for this is that the writers were allowed to finish phrases that had been started before the 30-minute session ended. For slower writers finishing the phrase takes longer.

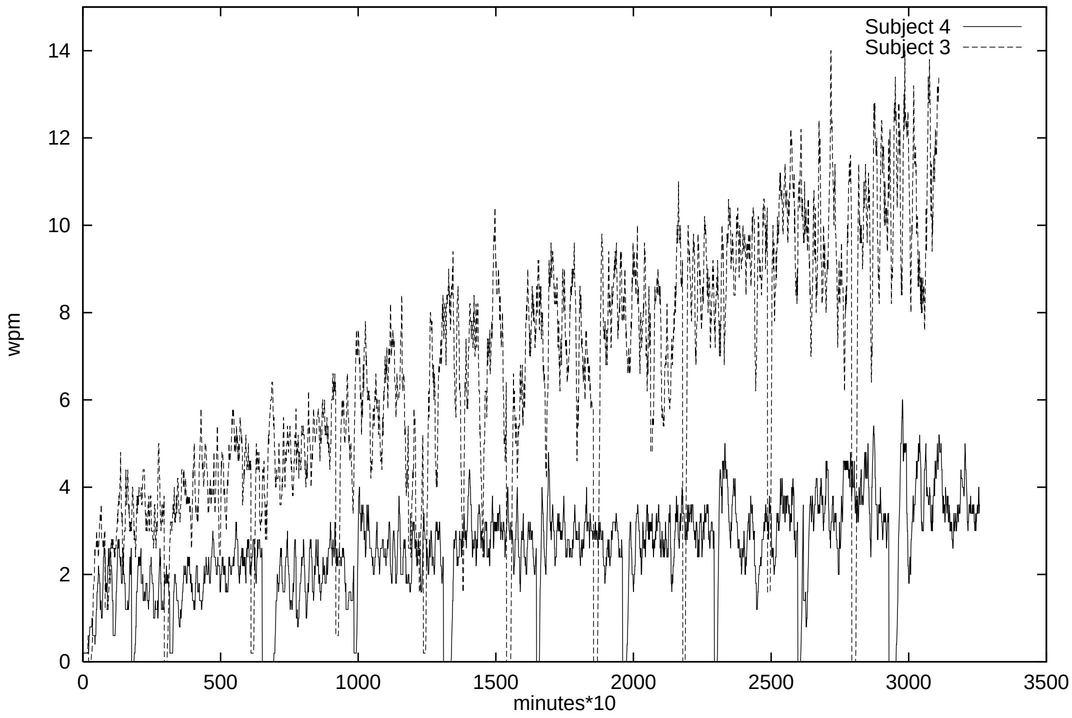

Figure 6.3 Wpm rates for subjects 3 and 4 throughout the experiment.

We can see that the speed for each writer varies mostly within 3 wpm of the average for a given session. The slow periods at the beginning and end of each session are the biggest exception reaching all the way down to 0 wpm. One minute window is long enough to hide most short breaks and exceptionally quick sections of the text. Shorter window will give more variation, spikes up to 20 wpm can be observed with a 10 second window. The curve shown in figure 6.3 tells us that most speed measurements that last longer than a minute will give results that are relatively close to the average speed measured over longer periods of time. This gives some credibility to the results in the 5 minute tests with different input devices that we will give later.

6.2 Measured error rate

Earlier we gave the difference between the raw and productive input speeds divided by two as an estimator for the error rate. While this number does tell something about the amount of work that goes into correcting the errors, it is not exactly the number of errors. The most notable source of inaccuracy are situations where an error goes unnoticed for a while and correct text is entered after the wrong input. Our software allows corrections only by erasing all the text between the end of the written string and the error. Thus one error may cause several characters to be erased.

To get a more accurate estimator for the error frequency we counted all errors manually from the log files. We tried to judge what exactly was the error and which inputs were used to correct it. The situation gets somewhat complicated when an error is made in correcting another error and possibly a third one while trying to correct them both. Thus we were not always able to count the errors exactly right. The result was, however, a number that is closer to the real error rate than our earlier estimate. Figure 6.4 shows the mean of the error rates of our five writers.

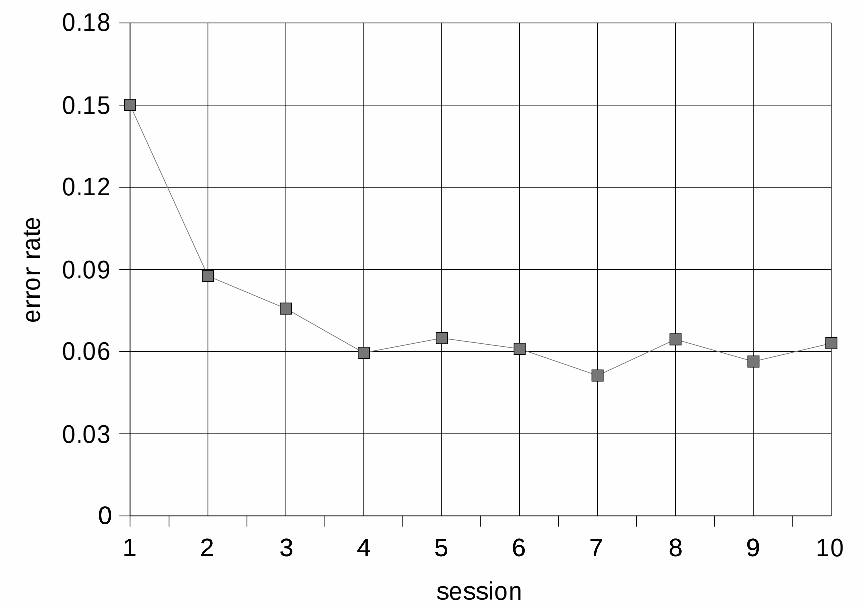

Figure 6.4 Mean error rate through the experiment.

As with the speed of text entry, the first sessions show the greatest change in error rate too. Familiarization with the touchpad and direction input explain the steep decrease in the error rate. After the third session the error rate stays between 5 and 6.5 percent with no apparent trend. This is probably the real level of MDTIM error rate when using a handheld Circue EasyCat touchpad and under stress for speed.

The error rate is high by most standards. A 6.5% error rate means that almost every 14th character needs to be erased and corrected. In her 1994 article LaLomia gives 3% as the minimum acceptable error rate for handwriting recognizers. This is probably true for unistroke recognizers such as Unistrokes, Graffiti and MDTIM. Therefore we can conclude that without improvements in the error rate MDTIM coupled with a touchpad is not good enough for general use.

6.3 Predicted expert speed and accuracy

Above we have described the measured user performance with MDTIM. One of our goals for this experiment was to predict expert performance with MDTIM. We will now discuss the prediction that can be derived from the data gathered during the experiment.

As seen in figure 6.1, the experiment was too short to allow meaningful prediction based on the learning curve. The curve is close to linear and we had too few test subjects to give reliable prediction. All we can say based on the curve in figure 6.1 is that experts will on average write faster than our test subjects during the last test.

For better prediction for expert performance we must use other means. Our data allows the computation of one simple estimate for expert input speed.

The prediction is based on the fastest character times that we measured. This means that we find the fastest writing time for each character and then assume that all instances of the character could be written equally fast by an expert. This computation gives an upper bound for the top speed for a given writer with a given skill level. That is, if a user manages to write all instances of the characters with her best speed, the resulting mean speed will match the estimate.

The writing times for a given character are not normally distributed. Figure 6.5 shows the distribution of writing times for the 1543 space characters entered during the last two sessions. The fitted normal distribution curve illustrates how poorly normal distribution estimates the distribution. The writing speeds are heavily clustered around 1000 ms, with very few instances below 250 ms, but many over 1750 ms. Intuitively it is clear that the writing speeds can never reach zero, but can get values infinitely far along the positive time axis in figure 6.5. Skewness and kurtosis figures computed for the other characters showed that with very few insignificant exceptions all writing time distributions follow the same general pattern of a tight cluster close to zero and a long right tail.

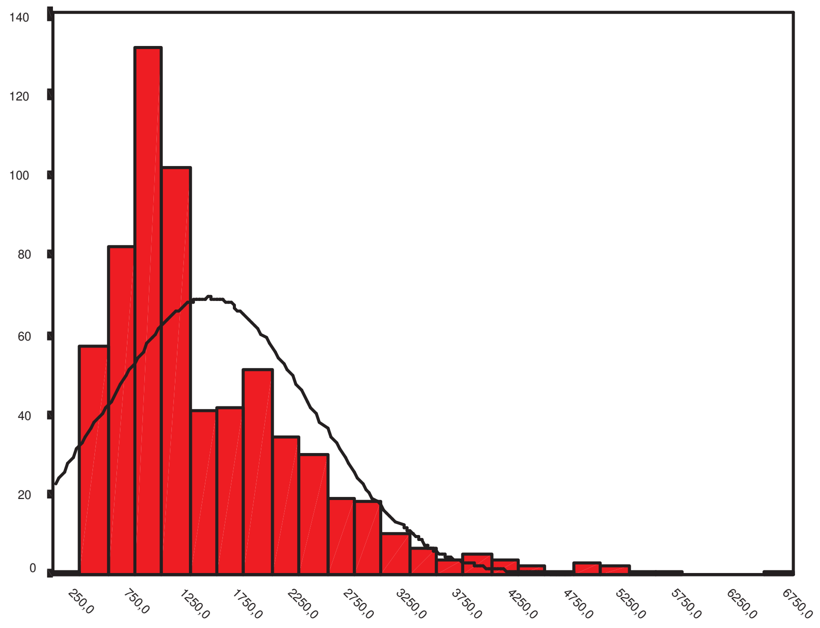

Figure 6.5 Distribution of the writing speed of the space character.

This means that the expert speed estimate described above will always be greater than the mean speed, but it is entirely possible to write several words at speeds close to the speed that our estimate gives (though maintaining this peak speed for even a minute is rather unlikely as we learned from figure 6.3). Thus our estimate is good for predicting error free peak performance for a writer that generated the data from which the estimate is computed.

Figure 6.6 shows the raw input speed that we measured and the upper bound prediction. The prediction is the average of the predictions computed individually for each user from her fastest character times. We see that the fastest character times were still decreasing at the end of the experiment. This suggests that our test users were not even close to their maximum speed yet.

Expert error rate is likely to be close to what we measured for the last sessions (see figure 6.4) because error rate typically does not decrease with practice after certain skill is achieved. In addition to our 10 session study, this fact has been observed in text input studies at least by MacKenzie and Chang [1999] and Frankish et al. [1995].

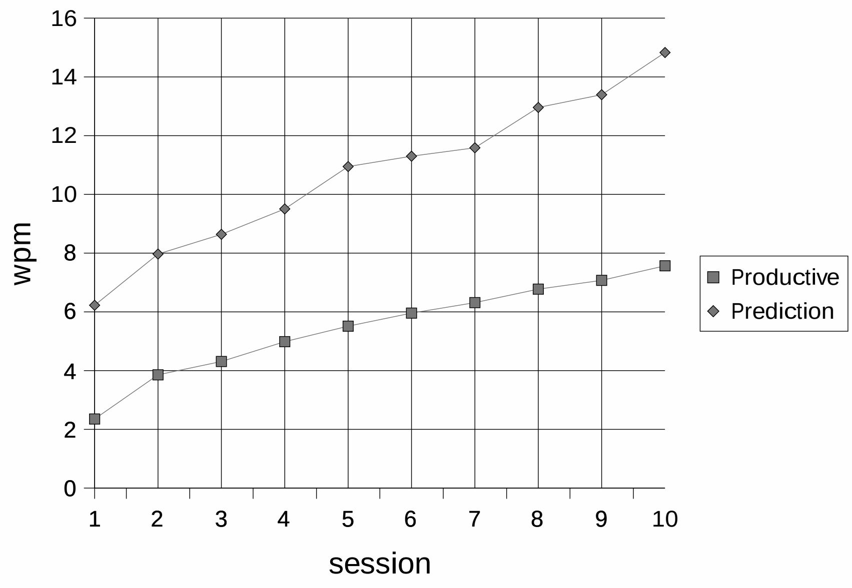

Figure 6.6 Upper bound prediction for writing speed.

6.4 Device (in)dependence

Above we have listed results on MDTIM speed and error rate when writing is done using a touchpad. Next we will give results for four other input devices. The users did not get to train writing with the other devices like they did with the touchpad. Therefore, the numbers for the touchpad represent performance after some training and the numbers for other devices show initial performance with a new device.

Figure 6.7 shows the productive wpm figures for each of our five test subjects with touchpad, trackball, mouse, joystick, and keyboard. Figures for the last four devices are computed from five minute tests. The touchpad figure is the same that was shown for session 10 in figure 6.2.

We see that the variation within subjects is sometimes as great as between subjects. The purpose of the experiment was to see whether MDTIM writing skill can be transferred from a device to another without significant speed penalty. It seems that this is indeed the case. Some test subjects were faster with other devices than with the familiar touchpad. We should not, however, pay too much attention to the individual five minute tests, because one exceptionally easy or difficult phrase can affect a short test rather much. The mean speeds are slightly more reliable and show that the touchpad was fastest with the mouse and

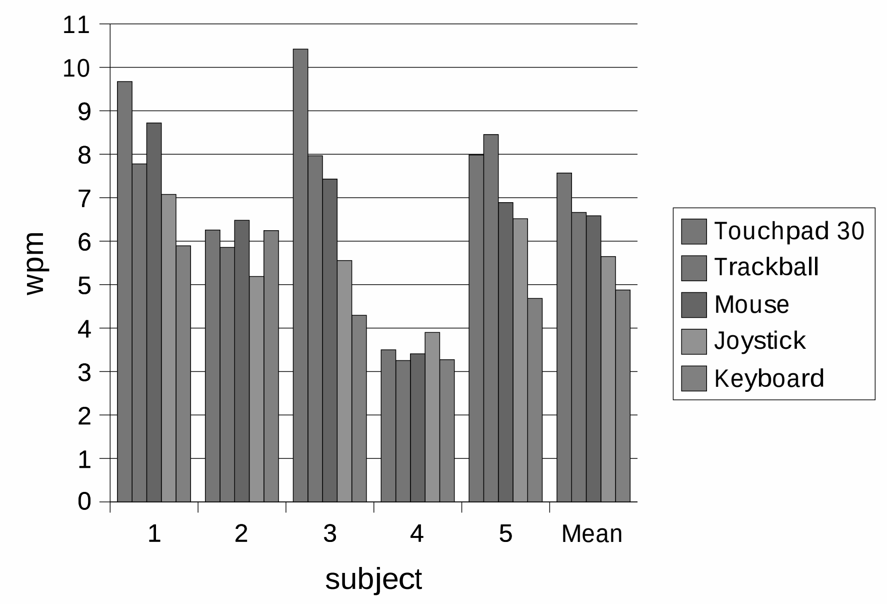

Figure 6.7 MDTIM speed on various devices.

trackball about one wpm behind and joystick and keyboard another wpm behind them.

This result is easy to explain. The touchpad should be fastest because the writers had just practiced on it for five hours. Mouse and trackball allow several directions to be drawn with single circular movement and thus with lower motor overhead. Joystick and keyboard both require a discrete finger movement for each direction and therefore they are the slowest. Joystick and keyboard are also more unlike the touchpad in the sense that they require different hand posture and thus more motor adaptation. Keyboard in particular is operated with two or three independently moving fingers whereas the operation of the other devices can be reduced to moving one finger on a two-dimensional plane.

Overall the speed differences do not seem to be very significant at this skill level. With more practice on one of the devices the situation might change.

Figure 6.8 shows the backspace counts in the same experiments. The most notable result is that the average error rate on joystick was slightly below 3% which would make it acceptable according to LaLomia's 1994 results. Keyboard is close to being acceptable, having an average error rate of 3.1%.

The fact that part of the devices were handheld, while others laid on a desk, does not seem to have an effect that could be seen through the variation caused

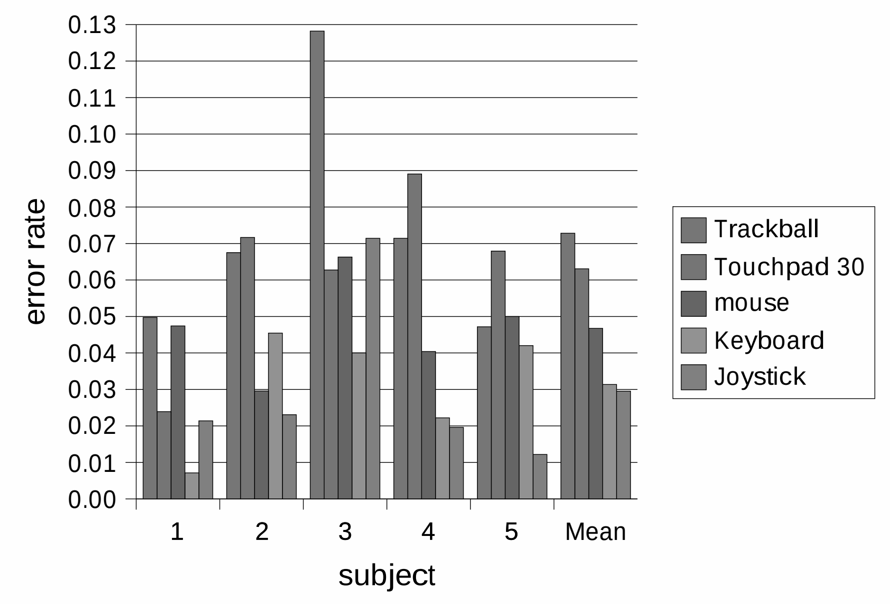

Figure 6.8 Error rates for all subjects and devices.

by other differences in the devices. The touchpad was held in the non-dominant hand and thus it could be operated by either moving the pad under the finger or by moving the finger over the pad. Similarly the joystick could have been operated by tilting the whole device and keeping the stick stationary or vice versa. Our experience and our observations during the experiment suggest that at least initially coordinating two hands is more difficult than coordinating one. The writers seemed to write faster and with fewer errors when the non-dominant hand with the pad was steadied against a knee or handrest than when the hand had no support below the elbow.

6.5 Handwriting, typing and MDTIM speeds

Although the sample is too small to give a reliable statistical proof of the correlation between handwriting, typing and MDTIM writing speeds, the data does suggest that such correlation may exist. Thus a note on this is in order.

Only one of the test subjects had learned touch-typing formally in school. Not surprisingly she was the fastest typist in our sample. None of the test subjects had used any pen-based text input method for longer than a couple of minutes. They were not familiar with Graffiti, Unistrokes or any form of shorthand writing. All used a QWERTY-keyboard daily.

Table 6.1 summarizes the measured typing, handwriting and MDTIM writing speeds. Typing and handwriting have poor correlation with correlation coefficient r=0.41. The three first writers type clearly faster than they can write by hand. The two last test subjects are slightly faster with pen and paper than they are with a QWERTY keyboard. The correlation between handwriting and MDTIM writing speed is stronger with r=0.65. Typing and MDTIM writing speeds have the best correlation in our data with r=0.87.

	writing speeds		
subject	**QWERTY-typing**	**handwriting**	**MDTIM productive**
1	35.40	17.96	9.67
2	29.16	11.40	6.26
3	39.52	28.72	10.42
4	14.04	16.52	3.50
5	21.12	21.88	7.98

Table 6.1 Typing, handwriting and MDTIM speeds in words per minute.

Overall it seems that fast writers tend to be fast with all methods and slow writers slow with all methods. There are at least two possible explanations. The first one has to do with cognitive capability to produce the stream of characters that need to be written. The second explanation places the bottleneck in the motor system. Our experiment does not reveal which, if either, is true.

6.6 The character set

The characters for the character set used in the experiment were chosen both to minimize the number of directions that need to be input and to give as many of the common characters a form that somehow resembles Latin handwriting characters. As described in chapter 4 we did not know the character frequencies well enough to make the character set optimal. Similarly we ran out of creativity in our attempt to make all characters resemble their Latin counterparts. Thus the result was a compromise regarding both our major goals. We have somewhat optimized direction consumption and a mixture of familiar and new character forms. This character set allows us to extract some factors that may help future character set designers.

We used the data collected during the two last training sessions with the touchpad. The data consists of timings for 10824 characters. Unfortunately as described in chapter 2 the character frequencies are very uneven. Of the 86 characters that appeared in our sample 40 appeared less than 20 times. This means that, in addition to being statistically unreliable, blindly comparing the mean writing speeds of the characters tells more about learning through repetition than about the goodness of the characters.

Generally speaking we are dealing with a bundle of interconnected factors that all contribute to the writing speed. The most frequent characters were given the shortest forms that resemble the Latin character set and are thus easier to learn. These same characters appeared most frequently in the writing practice thus giving the test subjects more practice on the characters that were easiest to learn and draw.

We will now identify three factors that correlate with the writing speed. Although correlation does not imply causation, these factors do have theoretical foundation and thus we can rely on that together they are responsible for a significant part of the difference in the observed writing times. The first factor is the number of directions needed for writing the character. The second factor is the degree to which the character resembles the Latin character with the same meaning. The last factor is the amount of practice that the writer has had with a character.

6.6.1 Number of directions

We constructed the character set with the assumption that the most frequent characters should be shortest to make them fastest to write. Table 6.2 shows that on the average shortest characters were indeed the fastest.

directions	**mean**	**N**
2	1216	2193
3	1489	5582
4	1827	3049

Table 6.2 Writing speeds (in milliseconds) for characters grouped by number of directions.

We gave space and backspace the shortest forms with only two directions. In

Table 6.2 the two direction group, however, has a surprisingly long mean writing time. The explanation for this is the fact that backspaces are relatively frequent (N=650) and slow (1549 ms). The long mean writing time for backspace can be explained with the long mental preparation before actually writing the character. The writer has to spot the error and decide to erase before she can start fetching the form of backspace from her memory and finally move her finger to draw the character. Space was clearly faster with mean writing time of 1076 ms.

Randomly assigning the 2 two-direction forms, 24 three-direction forms and 60 four-direction forms for the 86 characters that were present in our data would on average result in 3.67 directions to be used per character. The writers used only 3.07 directions per character during the last two sessions. This means that our attempt to minimize the number of directions needed for writing was somewhat successful.

The time required for invoking the “shift” effect needed for accessing the second meaning of a character can be computed from the data. We compared the mean writing time for lowercase alphabet (Latin a-z) and upper case alphabet (Latin A-Z). The mean writing time for the lowercase alphabet was 1510 ms (N=7008) and the mean for uppercase alphabet was 2007 ms (N=836). This gives a difference of 497 ms. This seems like a long time to press one button. The conclusion is, therefore, that button pressing, or the tap that some subjects used on the touchpad, requires a lot of mental preparation and therefore may not be the right way to handle uppercase characters. However, we should also notice that the uppercase characters were much more infrequent and therefore it does not really matter if writing them takes a little longer.

6.6.2 Similarity with a Latin character

To see the effect of similarity with Latin character set, we divided the characters into three groups according to subjectively perceived similarity. The first group holds the characters that can be drawn both as Latin and MDTIM characters with the same meaning (n, o, u). The second group consists of characters that have some similarity to their Latin counterparts, but require rotation or omittance and/or addition of some features to be recognized as MDTIM characters (backspace, space, *, a, b, c, d, r, s, t, w, ä, ö). The last group holds all characters that do not belong to either one of the first two groups. The mean writing times for the character groups are shown in Table 6.3.

The average number of directions per character in the group is also shown in

group	mean	directions	N
same	1214	3.4	1611
similar	1422	2.6	5019
dissimilar	1769	3.5	4194

Table 6.3 Writing speeds (in milliseconds) for characters grouped by similarity to their Latin counterparts.

Table 6.3. It seems that having the familiar Latin form for the character helps in making the character fast. However, the good time/direction ratio for the first group is largely due to letter o, which has four directions (WSEN), but is extremely easy to write because it can be drawn as a circle.

The case of "o" illustrates the difficulty of finding the significant factors. It has the best time/direction ratio of all characters and it has four directions. "O", however, is a frequent character making over 5% of all characters in our sample. Therefore, o should also be faster than most characters simply because it is so frequent that the test subjects had a lot of o-writing practice by the 9th session. Frequent practice, similarity with Latin "o", and motorically easy form are enough to make "o" the third fastest character in our sample even though it has four directions.

6.6.3 Amount of practice

The number of times a given character is written and the time that it takes to write it have a negative correlation. Figure 6.9 shows a scatter plot with the character count on the horizontal axis and mean writing time on the vertical axis. The plot does not show all characters. Some infrequent characters have very long mean writing times because the test subject has for some reason kept a long pause between characters, either to search the character in the reference chart or for some other reason. The very fast and very infrequent characters can be explained as errors. The two characters with mean writing speeds of less than 1000 ms are ä and cr (ASCII code 13) neither of which was needed for accomplishing the writing task the test subjects were given. The writers entered these characters in error and happened to do it very fast a small number of times (2 times for ä and 4 times for cr).

We can see that the negative correlation gets stronger if we eliminate the

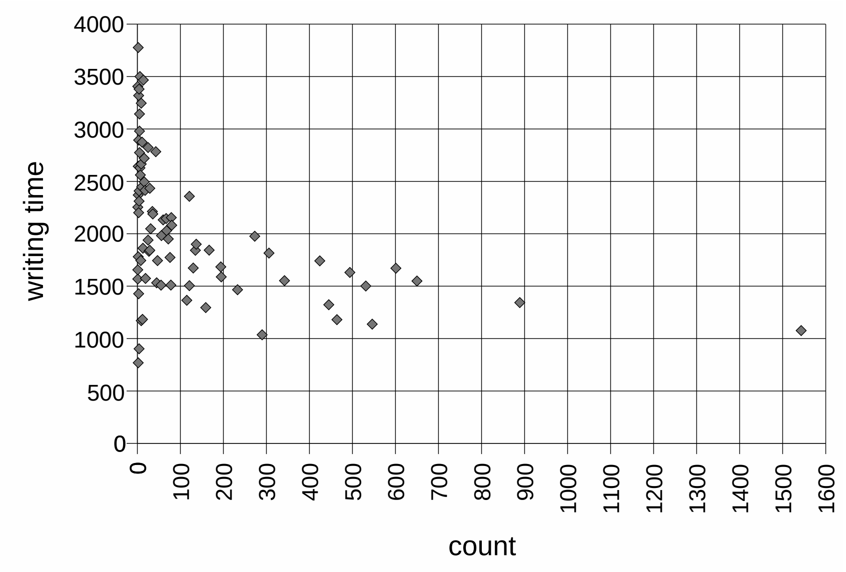

Figure 6.9 Writing time versus character count.

characters that have been entered enough times to give the mean some credibility. If we remove the characters that have not been entered at least five times (i.e. on average at least once by each writer) we get a correlation coefficient of r=-0.43 as opposed to the initial coefficient of r=-0.28.

6.7 Summary

The experiment did not reveal how fast experts could write using MDTIM because our test group did not finish their learning during the 5-hour experiment. At the end our test group using a touchpad wrote at a rate of slightly more than 7.5 wpm with an error rate close to 6%. Our estimate for expert writing speed with a touchpad is over 15 wpm.

The writing device does not seem all that important. Our test group wrote roughly at the same speed using a touchpad, trackball, mouse, joystick and keyboard the relative speeds being in this order. MDTIM writing speed on a keyboard was about 2.5 wpm slower than on the touchpad. The error rates on the different devices varied between 13% and 1% depending on the writer and on the device.

The number of directions per character, amount of practice per character, and the degree to which the character resembles the same character in the Latin character set all seem to affect the writing speed. However, speed differences between writers are also significant. A person's writing speeds with different writing methods seem to correlate.

7 Conclusions

We have examined a set of existing text input methods and concluded that while functional, they are often very dependent on certain physical devices. This observation led us to construct a minimal device-independent text input method. We proceeded to test the new method and found out that it works, but is very slow in comparison to other text input methods. In this chapter we evaluate our current construction and cast a look into the possible future development of our writing method.

As a handwriting character set MDTIM has both good and bad features according to Karlsdottir's criteria [1997]. It has a large number of acute angles and on a pen interface the pen has to move in all directions instead of a regular up-down pattern which has been found to be good for speed. On the other hand MDTIM completely eliminates all connecting strokes between characters. Unfortunately we could not continue the experiment long enough to give the test group the same amount of practice in MDTIM as they had in traditional handwriting and QWERTY-typing.

To be fair one should compare MDTIM only to methods with similarly constrained input devices. That is, systems working with a maximum of five keys or other input tokens. However, to be useful a universal text input method needs to perform at the level of all pen-based text input methods and most portable keyboards. Currently this is not true for MDTIM.

The results described in chapter 6 suggest that MDTIM is not fast enough. The initial performance is very poor. Even after five hours of practice MDTIM is clearly slower than most virtual keyboards and portable keyboards. We expect that with more practice MDTIM writing speed increases until it just slightly passes the performance of handwriting recognizers working with discrete characters. MDTIM error rates should also compare well against handwriting recognizers given that the same input device is used for both. Thus, highly experienced writers will find MDTIM usable, but since we do not know how much practice is needed to attain this kind of experience, we cannot make any recommendations on the use of MDTIM.

However, we did find that MDTIM writing skill seems indeed to be device-independent. This encourages us in our search for other, possibly faster, device-independent text input methods.

As mentioned earlier, MDTIM is not supposed to be the final word in text input. Rather it is an ad-hoc prototype for testing one method for achieving device-independence in text input. More research is needed to learn more about MDTIM and to improve it into a production quality text input method.

Below we describe topics that we plan to work on next. The list includes some immediate extensions to MDTIM as well as less obvious implications of the ideas upon which it is based.

Our tests do not reveal much on whether MDTIM is a viable option for eyes-free text input. Possibly it requires more auditory or tactile feedback to be usable. In our experiment the users saw the MDTIM characters forming on the screen as they wrote. We need to conduct a test without visual feedback before we can say how well users can keep track on what is happening in the recognition algorithm without seeing it.

The input space for MDTIM is two dimensional. With a 3D-input device and increased number of branches in the code tree we get a richer code which allows shorter characters or more characters with equal length. More research is needed in order to know whether a text input method like this is usable or not.

Computers, especially when connected to the Internet, can open a whole new and meaningful life to people with severe limitations in limb mobility. Unfortunately, when the hands cannot be used properly, keyboards and mice become difficult to use. MDTIM should be easy to use with a joystick regardless of which appendage is used to operate the joystick. Fingers, toes, chin, and tongue [Salem and Zhai, 1997] should all work well.

For years now researchers have tried to adapt the computer to the human ways of interaction. This work has certainly been successful, but the limits of our current ability to program computers are already showing in some areas. Computers do not yet understand natural language input. If they do, they only understand one language and tend to be speaker or writer dependent. Text input methods such as Unistrokes, T-Cube, Quikwriting and MDTIM are systems that try to utilize the strengths of both humans and computers. They offer a visual and tactile environment that is well suited for our hands and visual system while retaining a structure that allows easy algorithmic interpretation of the input.

The notion of creating input methods that lie in this middle ground between human and machine communications should offer an easier way to develop new interaction techniques than trying to make the computers interpret input that untrained humans create.

One area possibly worth investigating is a middle ground language for spoken communication. The language should utilize the parts of natural phonology that are easy for computers to recognize and it should have a simple well defined structure that still allows inventive sentence structures that are typical for human speech. The rather megalomaniac aim of this project would be to teach the language to everybody in the world and thus allow the speaker independent use of all computers on the planet for all people once they have learned the language.

So far our findings do not touch the matters of universal spoken languages. Instead, we have demonstrated that device-independence is an option in manually operated text input methods. Our method may not be fast enough or easy enough to learn to be the only text input method in future computing devices. However, the undeniable utility of a universal text input method should be significant enough to persuade one to contemplate the option of trying to design one if a text input method is to be designed anyway in a given PDA-project.

References

[Adams, 1999] Russ Adams. *Bar Code 1*, chapter Bar Code Specifications. Adams Communications, http://www.adams1.com/, 1999.

[Balakrishnan and Patel, 1998] Ravin Balakrishnan and Pranay Patel. The padmouse: Facilitating selection and spatial positioning for the non-dominant hand. In *Conference proceedings on Human factors in computing systems*, pages 9 – 16. ACM, 1998.

[Barber, 1997] Christopher Barber. *Beyond the Desktop*. Academic Press, 24-28 Oval Road, London NW1 7DX, UK, 1997.

[Bellman and MacKenzie, 1998] Tom Bellman and I. Scott MacKenzie. A probabilistic character layout strategy for mobile text entry. In *Proceedings of Graphics Interface*, pages 168 – 176, Toronto, 1998. Canadian Information Processing Society.

[Burnett and Gottfried, 1998] Margaret M. Burnett and Herkimer J. Gottfried. Graphical definitions: expanding spreadsheet languages through direct manipulation and gestures. *ACM Transactions on Computer-Human Interaction*, 5(1):1 – 33, 1998.

[Campbell, 1999] Duncan Campbell. *Interception Capabilities 2000*, chapter Technical annexe. European Parliament Directorate General for Research Directorate A The STOA Programma, http://www.nrc.nl/W2/Lab/Echelon/interccapabilities2000.html, 1999.

[Card *et al.*, 1983] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates Inc., Hillsdate, New Jersey, USA, 1983.

[CH Products, 1999] CH Products. *Trackball Pro*. CH Products, http://www.chproducts.com/pcgear/tbpro.html, 1999.

[Chatty and Lecoanet, 1996] Stéphane Chatty and Patric Lecoanet. Pen computing for air traffic control. In *Conference proceedings on Human factors in computing systems*, pages 87 – 94, New York, 1996. ACM.

[Cirque, 1999] Cirque. *Easy Cat.* Cirque Corporation, http://www.cirque.com/, 1999.

[Frankish *et al.*, 1995] Clive Frankish, Richard Hull, and Pam Morgan. Recognition accuracy and user acceptance of pen interfaces. In *CHI '95. Conference proceedings on Human factors in computing systems*, pages 503 – 510, New York, 1995. ACM.

[Goldberg and Goodisman, 1991] David Goldberg and Aaron Goodisman. Stylus user interfaces for manipulating text. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 127 – 135. ACM, 1991.

[Goldberg and Richardson, 1993] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Conference proceedings on Human factors in computing systems*, pages 80 – 87. ACM, 1993.

[Gopher and Raij, 1988] D. Gopher and D. Raij. Typing on a two-handed chord keyboard: Will QWERTY become obsolete? *IEEE Transactions on Systems, Man, and Cybernetics*, 18:601 – 609, 1988.

[Gutenberg, 1999] Project Gutenberg. *Project Gutenberg website.* http://promo.net/pg/index.html, 1999.

[Huerta, 1999] Marcos Huerta. *Dvorak Keyboard.* http://ccwf.cc.utexas.edu/~marcosh/dvorak.html, 1999.

[Isokoski, 1998] Poika Isokoski. Tekstin syöttö kynäkäyttöliittymissä [text input in pen-based user interfaces]. In Pertti Järvinen, editor, *Pieniä ATK-alan tutkimuksia - syksy 1998*, pages 33 – 53. Computer Science Department, University of Tampere, Tampere, Finland, 1998.

[John, 1995] Bonnie John. Why GOMS? *Interactions*, 2(4):80 – 89, 1995.

[Karlsdottir, 1997] Ragnheidur Karlsdottir. Comparison of cursive models for handwriting instruction. *Perceptual and Motor Skills*, 85(3):1171 – 1184, 1997.

[Kurtenbach and Buxton, 1993] Gordon Kurtenbach and William Buxton. The limits of expert performance using hierarchic marking menus. In *Conference proceedings on Human factors in computing systems*, pages 482 – 487. ACM, 1993.

[LaLomia, 1994] Mary J. LaLomia. User acceptance of handwritten recognition accuracy. In *Conference on Human factors in computing systems Conference companion*, page 107. ACM, 1994.

[Lintern, 1991] Gavan Lintern. An informational perspective on skill transfer in human-machine systems. *Human Factors*, 33(3):251 – 266, 1991.

[Lukatela and Turvey, 1998] Georgije Lukatela and M. T. Turvey. Reading in two alphabets. *American Psychologist*, 53(9):1057 – 1073, 1998.

[MacKenzie and Chang, 1999] I. Scott MacKenzie and L. Chang. A performance comparison of two handwriting recognizers. *Interacting with Computers*, 11:283 – 297, 1999.

[MacKenzie and Zhang, 1997] I. Scott MacKenzie and Shawn X. Zhang. The immediate usability of graffiti. In *Proceedings of Graphics Interface* [1998], pages 129–137.

[MacKenzie and Zhang, 1999] I. Scott MacKenzie and Shawn X. Zhang. The design and evaluation of a high performance soft keyboard. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, New York, 1999. ACM.

[MacKenzie *et al.*, 1991] I. Scott MacKenzie, Abigail Sellen, and William Buxton. A comparison of input devices in elemental pointing and dragging tasks. In *Conference Proceedings on Human Factors in Computing Systems*, pages 161 – 166. ACM, 1991.

[MacKenzie *et al.*, 1994] I. Scott MacKenzie, Blair Nonnecke, Stan Riddersma, Craig McQueen, and Malcolm Meltz. Alphanumeric entry on pen-based computers. *International Journal of Human-Computer Studies*, 41:775 – 792, 1994.

[MacKenzie, 1999] I. Scott MacKenzie. Text entry using soft keyboards. *Behaviour and Information Technology (to appear)*, 1999.

[Mankoff and Abowd, 1998] Jennifer Mankoff and Gregory D. Abowd. Cirrin: a word-level unistroke keyboard for pen input. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 213 – 214. ACM, 1998.

[Masui, 1998] Toshiyuki Masui. An efficient text input method for pen-based computers. In *Conference proceedings on Human factors in computing systems*, pages 328 – 335. ACM, 1998.

[Matias *et al.*, 1993] Edgar Matias, I. Scott MacKenzie, and William Buxton. Half-QWERTY: a one-handed keyboard facilitating skill transfer from QWERTY. In *Conference proceedings on Human factors in computing systems*, pages 88 – 94. ACM, 1993.

[Matias *et al.*, 1996] Edgar Matias, Scott I. MacKenzie, and William Buxton. One-handed touch-typing on a qwerty keyboard. *Human-Computer Interaction*, 11:1 – 27, 1996.

[Mayzner and Tresselt, 1965] M. S. Mayzner and M. E. Tresselt. Tables of single-letter and digram frequency counts for various word-length and letter-position combination. *Psychonomic Monograph Supplements*, 1:13 – 32, 1965.

[Perlin, 1998] Ken Perlin. Quikwriting: continuous stylus-based text entry. In *Proceedings of the 11th annual ACM symposium on User interface software and technology* [1998], pages 215 – 216.

[Perlin, 1999] Ken Perlin. Email Communication: quikwrite-talk@cat.nyu.edu, 1999.

[Red Hat, 1998] Red Hat. *Red Hat Linux 5.2.* http://www.redhat.com or ftp://ftp.funet.fi/pub/Linux/images/RedHat/redhat-5.2/i386/, 1998.

[Sagawa *et al.*, 1997] Hirohiko Sagawa, Masaru Takeuchi, and Masaru Ochi. Description and recognition methods for sign languge based on gesture components. In *Proceedings of the1997 international conference on Intelligent user interfaces*, pages 97 – 104. ACM, 1997.

[Salem and Zhai, 1997] Chris Salem and Shumin Zhai. An isometric tongue pointing device. In *Proceedings of the CHI '97 Conference on Human Factors in Computing Systems*, pages 538 – 539, New York, 1997. ACM.

[Soukoreff and MacKenzie, 1995] R. William Soukoreff and I. Scott MacKenzie. Theoretical upper and lower bounds on typing speed using a stylus and soft keyboard. *Behaviour and Information Technology*, 14(6):370 – 379, 1995.

[Sugimoto and Takahashi, 1996] Masakatsu Sugimoto and Kimiyo Takahashi. SHK single hand key card for mobile devices. In *Conference on Human factors in computing systems Conference Compation*, pages 7 – 8, New York, 1996. ACM.

[Tappert *et al.*, 1994] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Artificial Intelligence*, 12(8):787 – 808, 1994.

[Textware Solutions, 1998] Textware Solutions. *The Fitaly One Finger Keyboard*. http://www.twsolutions.com/fitaly/fitaly.htm, 1998.

[Venolia and Neiberg, 1994] Dan Venolia and Forrest Neiberg. T-cube: A fast, self-disclosing pen-based alphabet. In *Conference proceedings on Human factors in computing systems*, pages 263 – 270. ACM, 1994.

[Watson, 1993] Richard Watson. A survey of gesture recognition techniques. Technical Report TCD-CS-93-11, Department of Computer Science, Trinity College, Dublin 2, 1993.

A MDTIM direction string dictionary

string	character	with modifier
NSW	a	A
SEW	b	B
ESW	c	C
SWE	d	D
WES	e	E
ESNE	f	F
ESNS	g	G
WSWS	h	H
WNS	i	I
SESW	j	J
WSWE	k	K
SNS	l	L
WSWN	m	M
NSN	n	N
WSEN	o	O
WNEN	p	P
WSES	q	Q
WSN	r	R
ESE	s	S
SNE	t	T
SEN	u	U
WNWS	v	V
WNWN	w	W
SWSN	x	X
SWSE	y	Y
SWSW	z	Z
WEN	å	Å
NSE	ä	Ä
WNES	ö	Ö
ENEN	1	1
ENES	2	2

ENEW	3	3
ENSN	4	4
ENSE	5	5
ENSW	6	6
ENWN	7	7
ENWE	8	8
ENWS	9	9
ESNW	0	0
EWEN	.	:
EWES	,	;
EWEW	?	?
EWSN	!	!
EWNW	(	)
EWSW	{	}
EWSE	<	>
EWNE	-	+
EWNS	*	/
SNWE	[	]
SNWS	'	"
SNWN	@	@
SESN	|	|
WNWE	=	=
WNEW	$	$
WEWE	&	&
NE	space	space
NW	backspace	bacspace
SWN	return	return

B MDTIM reference chart