1 Thumb, 4 Buttons, 20 Words Per Minute: Design and Evaluation of H4-Writer

I. Scott MacKenzie[1], *R. William Soukoreff*[2], *Joanna Helga*[1]

[1]Dept. of Computer Science and Engineering
York University
4700 Keele Street
Toronto, Ontario, M3J 1P3 Canada
+1 416 736-2100
{ mack, joanna }@cse.yorku.ca

[2]Dept. of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario, M5S 3G4 Canada
+1 416 978-3619
will@dgp.toronto.edu

ABSTRACT
We present what we believe is the most efficient and quickest four-key text entry method available. *H4-Writer* uses Huffman coding to assign minimized key sequences to letters, with full access to error correction, punctuation, digits, modes, etc. The key sequences are learned quickly, and support eyes-free entry. With *KSPC* = 2.321, the effort to enter text is comparable to multitap on a mobile phone keypad; yet multitap requires nine keys. In a longitudinal study with six participants, an average text entry speed of 20.4 wpm was observed in the 10th session. Error rates were under 1%. To improve external validity, an extended session was included that required input of punctuation and other symbols. Entry speed dropped only by about 3 wpm, suggesting participants quickly leveraged their acquired skill with *H4-Writer* to access advanced features.

ACM Classification: H.5.2 [Information Interfaces and Presentation]: User Interfaces – input devices and strategies (e.g., mouse, touchscreen)

General Terms: Performance, Design, Experimentation, Human Factors

Keywords: Text entry, Huffman coding, mobile text entry, small devices

INTRODUCTION

New methods of text entry appear with great frequency these days. This is due in large part to a heightened interest in small hand-held devices. While texting on mobile phones is the most obvious example, small or reduced form-factor devices are of interest in other domains, such as ubiquitous computing, wearable computing, gaming, and accessible computing. These environments often require an input mechanism that is physically constrained and that requires only a few keys or a small set of input primitives.

Text entry methods generally fall within one of two categories: methods using discrete actions on a keyed device (virtual or physical) and methods using continuous actions on a digitizing surface. The focus in this paper is the former: text entry using keys or buttons.

In the following sections, we present a new method for designing optimized small keyboards. The result is a family of reduced-key keyboards. We present and analyse the design space for keyboards with 2 to 27 keys. A 4-key design is chosen for detailed analysis and empirical evaluation.

Symbols, Codes, and Keys

Most previous research in designing optimized keyboards takes the approach of assigning letters to keys, typically by exploiting the statistical properties of a language [e.g., 1, 3-5, 7, 16, 19, 22]. We take the opposite approach: assigning keys to letters.

The keys-to-letters idea is somewhat like assigning codes to symbols, as, for example, assigning binary codes to letters in the UTF-8 character encoding system. Figure 1a shows the code corresponding to "a". Since the encoding is binary, the code consists of 0s and 1s only. We could, however, interpret the 0s and 1s as key presses on a 2-key keyboard, in which case entering "a" would require 8 key presses, as depicted in Figure 1b.

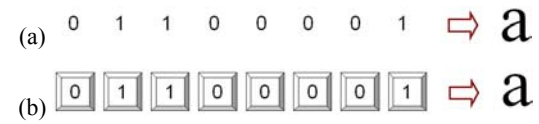


Figure 1. (a) Code-to-symbol assignment for "a" in UTF-8. (b) Equivalent key sequence.

Clearly, this exercise has little practical value, since input requires numerous key presses for each letter of input. However, it is noteworthy that only two keys are required. A simple question is this: Is there a more efficient coding

UIST'11, October 16–19, 2011, Santa Barbara, CA, USA.

system to reduce the number of key presses? The answer is equally simple: yes.

Huffman Codes

In 1952, Huffman described an algorithm to generate a "minimum-redundancy code" such that "the average number of digits per message is minimized" [6, p. 1098]. The algorithm builds a code tree from a set of symbols and their relative probabilities or frequencies in a language. Frequent symbols get shorter codes; infrequent symbols get longer codes. As an example, the codes generated by Huffman's algorithm for the letters of the English alphabet, with the addition of a SPACE character, are given in Figure 2. The symbol frequencies are from a version of the British National Corpus[1] containing 67 million words [20].

Symbol	Frequency	Huffman Code
[space]	67962112	111
e	37907119	010
t	28691274	1101
a	24373121	1011
o	23215532	1001
i	21820970	1000
n	21402466	0111
s	19059775	0011
h	18058207	0010
r	17897352	0001
l	11730498	10101
d	10805580	01101
c	8982417	00001
u	8022379	00000
f	7486889	110011
m	7391366	110010
w	6505294	110001
y	5910495	101001
p	5719422	101000
g	5143059	011001
b	4762938	011000
v	2835696	1100000
k	1720909	11000011
x	562732	110000100
j	474021	1100001011
q	297237	11000010101
z	93172	11000010100

Figure 2. Huffman codes built from the frequencies and letter symbols (including SPACE) in English.

The lengths of the codes vary from 3 to 11 bits (or "digits", in Huffman's terms). The mean code length, weighted by frequency, is 4.117 bits. The code length is equivalent to keystrokes per character (*KSPC*) if the digits are viewed as keystrokes, as per the example in Figure 1b. *KSPC* is a common metric for text entry research [9]. It is the average number of keystrokes required to produce each character of text in a given language using a given input method. While *KSPC* = 4.117 is still quite high, it is only about half the *KSPC* of the UTF-8 example above, and only two keys are required.

The codes in a Huffman tree are called *prefix codes*, since they are at leaf nodes in the tree. This means, for example, if the code 010 represents a symbol, no other symbol has a code beginning with 010 (see "e" in Figure 2). The importance for text entry is that delimiting actions are not necessary to uniquely identify the meaning of a sequence of input actions.

[1] ftp://ftp.itri.bton.ac.uk/

Design Space

Huffman's algorithm is most-commonly used for binary, or base-2, codes for electronic communications systems. Although rarely done, the algorithm can also generate codes using other bases.

To explore the design space of optimized small keyboards using Huffman code-to-symbol assignments, Figure 3 shows the mean code length, or *KSPC*, for Huffman code trees built for bases 2 through 27 supporting the 27 symbols in Figure 2. (Other points in the design space are plotted for comparison, as discussed later.) As expected, there is a relationship between the base of the codes and the mean code length. The lower the base, the longer the mean code length; the higher the base, the shorter the mean code length. This is relevant for text entry. A lower base means more keystrokes but fewer keys. A higher base means fewer keystrokes but more keys.

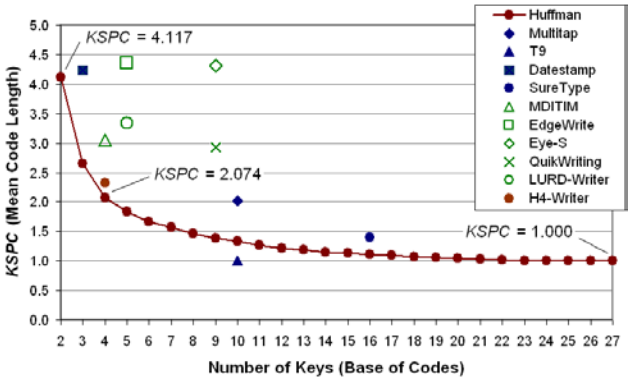


Figure 3. *KSPC* (mean code length) vs. number of keys (base of codes) for Huffman codes using the 27 symbols in Figure 2.

The right-most point in Figure 3 represents the Huffman code tree for base-27. The result is a conventional keyboard, with *KSPC* = 1. There are 27 keys but only one keystroke is required to produce each character. At the other extreme, we see the base-2 example discussed earlier, with *KSPC* = 4.117. These two points encompass the design space of interest in this paper.

The base-4 Huffman value has *KSPC* = 2.074 (see figure). This is close to *KSPC* = 2.032, the value for multi-tap on a mobile phone keyboard [9] (see ◆ in figure). This is interesting since multi-tap requires 10 keys: 9 keys for a-z and SPACE and a 10th key to delimit successive letters on the same key (e.g., "cab" = "222n2n22" where "n" is a designated "next" key).[2] So, the prospect of a 4-key keyboard producing English text in just over two keystrokes per character is provocative.

It is possible to have points below the line in Figure 3, but they require predictive techniques of some form. An

[2] A timeout may also serve as a delimiter. In a sense, the timeout is like a 10th key, since it produces a discrete event not available with the 9 keys used for text input.

example is T9 on a mobile phone, which has a very low *KSPC* = 1.007 [9] (see ▲ in figure). T9 requires a dictionary as well as a 10th key ("next") to cycle through alternate words when there are multiple matches for a key sequence. However, visual attention is imposed on the user who must view the display to determine if the correct word appears at the end of each key sequence [20]. In general, predictive techniques do not support eyes-free input.

Both multi-tap and T9 require additional keys and interaction methods to correct errors or to enter punctuation symbols, uppercase letters, digits, etc. As demonstrated later in this paper, a keyboard designed using Huffman codes can provide these same features without additional keys.

Besides the multi-tap and T9 points, there are additional points that could be added to Figure 3 for the many phone-like and other keypads and text entry methods proposed in the literature (see [11, 13] for reviews). A few points have been selected for discussion here.

Text entry is possible using three keys (base-3) using the "date stamp" method [10]. Letters appear on a virtual keyboard. Left and right cursor keys navigate the virtual keyboard and ENTER selects letters (see Figure 4a). *KSPC* = 4.230 using a fluctuating optimized letter arrangement (■ in Figure 3).

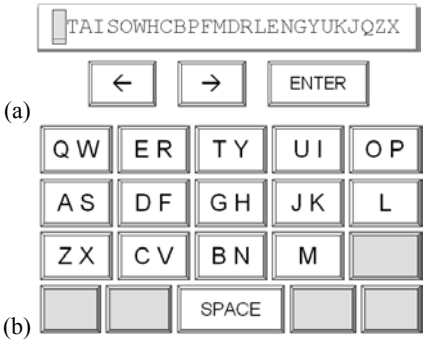


Figure 4. Example text entry methods. (a) 3-key date stamp method [10]. (b) SureType letter arrangement on RIM's *Blackberry 7100*.

The Qwerty-like keyboard on RIM's *Blackberry 7100* – called SureType – uses a five-column arrangement with two letters per key (see Figure 4b). This form factor allows for bigger keys but introduces ambiguity. In all, 16 keys are required: 14 letter keys, 1 SPACE key, and 1 "next" key (not shown). Using T9-like input, *KSPC* is close to 1. Using multi-tap, *KSPC* = 1.404 (● in Figure 3).

There are two points in Figure 3 for the base-9 condition. Although described as gesture systems by the authors, comparisons are relevant here since the gestures involve navigating between discrete regions on a display. *Eye-S* is shown in Figure 5a [18]. Used with an eye tracker, text entry involves fixating on targets in a sequence defined for each letter. The sequence for "c" (248) is shown in the figure.

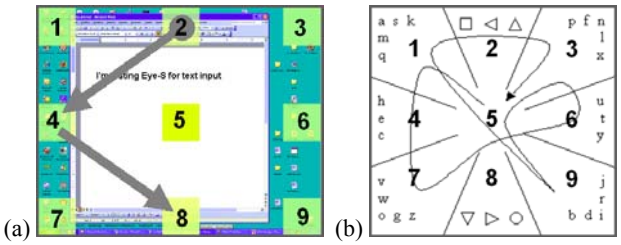


Figure 5. Base-9 text entry methods. (a) *Eye-S* [18]. (b) *Quikwriting* [17].

Quikwriting is a stylus input method [17]. As with *Eye-S*, it involves navigating around discrete regions on a display. Two gestures are necessary to enter a letter; the first selects a group of letters, and the second selects a letter within the group, with the center (region 5 in the figure) delimiting the letter gestures. Moving the stylus from the center into a perimeter region selects the respective group of letters. A letter within the group is selected by moving the stylus either back to the center region (selecting the middle letter of a group) or first through 1 or 2 perimeter regions (denoting the other letters in the group, respectively). The sequence for "quik" (517 563 59 513) is shown in the figure.

Eye-S and *Quikwriting* have specific target sequences for each letter, including SPACE, thus *KSPC* is easily calculated. The results are *KSPC* = 4.318 for Eye-S and *KSPC* = 2.929 for *Quikwriting* (◇ and ✕ in Figure 3, respectively). These values are substantially higher than *KSPC* = 1.391, the base-9 statistic using Huffman codes. Of course, any comparison must acknowledge the different design goals.

Eye-S was designed to create patterns that mimic Roman letters. For example, the target sequence in Figure 5a is similar in shape to "c". The result, although sub-optimal in terms of *KSPC*, is easier for users to learn.

A design goal for *Quikwriting* was to devise a stylus input method that avoided the thorny problem of handwriting recognition. Clearly, this goal was met. With *Quikwriting*, recognition is a simple matter of detecting stylus transitions through discrete zones (viz. targets). There is a one-to-one correspondence between target sequences and letters in the alphabet. Recognizing handwritten strokes, on the other hand, requires computing a set of shape features for a user-entered stroke and comparing the features to entries in a lookup table to find the best match – a much harder problem.

The Huffman design objective is simple: minimize the mean code length or *KSPC*. While this is a notable pursuit, there are many outstanding issues, such as implementation, ease of learning, and human performance. We will examine these issues shortly. But, first, we focus on the base-4 and base-5 conditions in Figure 3.

BASE 4 AND BASE 5

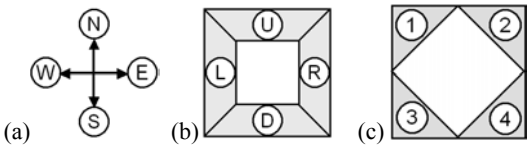
The codes and *KSPC* values for several text entry methods are shown in Figure 6. These methods were chosen since they all involve (about) four primitive input actions. Thus, they are reasonable points of comparison for the base-4 condition in Figure 3. The second column, labeled H4, gives the Huffman codes for the base-4 condition discussed above. *KSPC* = 2.074 (see Figure 3).

Symbol	H4	MDITIM	LURD-Writer[1]	Edge-Write[1]	H4-Writer
[space]	0	NE	L*	21*	DD
e	32	WES	LL*	2134*	UU
t	31	SNE	UU*	124*	RR
a	23	NSW	RR*	324*	RD
o	22	WSEN	LUUR*	21342*	RL
i	21	WNS	LU*	13*	UD
n	13	NSN	RD*	3142*	UR
s	12	ESE	UR*	2143*	DU
h	11	WSWS	URR*	1324*	UL
r	10	WSN	LUR*	213*	DRR
l	333	SNS	DLL*	134*	DLL
d	332	SWE	LUU*	2434*	DRU
c	331	ESW	LURR*	234*	DLD
u	330	SEN	UUR*	1342*	DLR
f	303	ESNE	DDL*	213*	DLU
m	302	WSWN	LLU*	31424*	DRD
w	301	WNWN	RDD*	13242*	RUD
y	203	SWSE	DLLU*	1424*	RUR
p	202	WNEN	URRD*	3123*	RUU
g	201	ESNS	URD*	21343*	RUL
b	200	SEW	RDL*	1343*	DRLD
v	3003	WNWS	DLU*	134*	DRLR
k	3002	WSWE	RRD*	13234*	DRLU
x	3000	SWSN	URDD*	1423*	DRLLD
j	30012	SESW	RDLL*	243*	DRLLR
q	30011	WSES	RDLU*	213424*	DRLLU
z	30010	SWSW	RDDL*	1234*	DRLLL
KSPC =	2.074	3.047	3.339	4.356	2.321

[1] The asterisk (*) is a delimiting action

Figure 6. Code-to-symbol assignments and *KSPC* for several base-4 or base-5 text input methods. (Figure 7 explains the labels).

Figure 7 shows the mapping between the labels used in Figure 6 and the buttons, stoke directions, or corners, used in the original research.



(a) (b) (c)

Figure 7. Labels for codes in Figure 6. (a) *MDITIM.* (b) *LURD-Writer* and *H4-Writer.* (c) *EdgeWrite.*

Minimal Device Independent Text Input Method

One example of base-4 prefix codes is Isokoski and Raisamo's Minimal Device Independent Text Input Method (*MDITIM*) [8]. Their system was designed both for keyed input and gesture input. Only four keys or zones were used, one for each compass direction (N, S, E, W). Rather than use an optimized code set, however, they selected codes based on a few design heuristics, primarily to support gesture input. For example, where possible, they choose sequences that were spatially similar to the intended letter, as shown in Figure 8a for the letter "b". Figure 8b shows the equivalent key sequence.

(a) (b)

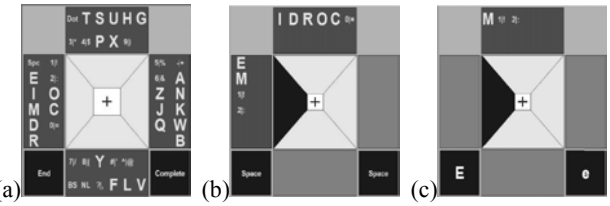
Figure 8. *MDITIM* for "b". (a) Gestures. (b) Keys [8].

As well, Isokoaki and Raisamo avoided codes with the same gesture used consecutively, such as WW, since it is hard to distinguish consecutive strokes in the same direction (without a delimiting action, such as pen up). This heuristic, while reasonable for gesture input, has the opposite effect for keyed input. Consecutive presses on the same key are fast, so repeating "digits" in the code have a positive effect for keyed input.

MDITIM includes additional codes for numerals and punctuation symbols (not shown). The *MDITIM* key-to-symbol assignments are shown in the third column in Figure 6. In the end, *MDITIM* yields *KSPC* = 3.047 (Δ in Figure 3). This is 47% higher than *KSPC* = 2.074 for the base-4 Huffman code-to-symbol assignments in the second column of Figure 6.

LURD-Writer

With Felzer and Nordmann's *LURD-Writer* [2], the user moves the mouse cursor left, up, right, or down to hit the edges of an input square. Although the mouse movements are like gestures, the edges are like soft keys, which are selected when contacted by the mouse cursor. The initial state of the input square is shown in Figure 9a. Each hit selects a group of characters, which are then rearranged along the edges. As edges are hit, characters are pre-selected and displayed in the lower left and right corners of the input square. The user may select a character with a left or right mouse button click, or hit another edge to further narrow in on the desired character. The input of "e" involves three actions: LEFT, LEFT, CLICK, as illustrated in Figure 9.



(a) (b) (c)

Figure 9. *LURD-Writer* input of "e" [2]. (a) Initial state. (b) After selecting left edge. (c) After selecting left edge again. Final selection of "e" occurs with a right mouse button click (left button for uppercase).

The LURD codes for a-z and SPACE are given in Figure 6, fourth column. The choice of codes was based on design heuristics supporting a hands-free mouse emulator for users with physical disabilities. Input used a headband and sensor that generated pulses in response to intentional muscle contractions in the user's brow. Single or double contractions evoked state changes in the "mouse". One

heuristic imposed was to only use clockwise changes in the movement direction. As seen in Figure 6, clockwise patterns, such as LU or LUR, are used, whereas counter-clockwise patterns, such as UL or RUL, are not.

For each code, a final asterisk (*) is added to represent the mouse button click for selection. Since *LURD-Writer* does not use prefix codes, a delimiter is needed to distinguish, for example, L (SPACE) from LL (e). The button click effectively adds 1 to the code length for each symbol, since the button click is akin to a 5th key. For this reason, we consider *LURD-Writer* a base-5 text entry method.

LURD-Writer includes codes for the entry of digits and punctuation symbols and also includes a word completion mode (not shown). For the core 27 symbols (a-z, SPACE), *KSPC* = 3.339 (○ in Figure 3) making *LURD-Writer* slightly less efficient than *MDITIM*.

EdgeWrite

Another text input method of interest is Wobbrock et al.'s *EdgeWrite* [24]. *EdgeWrite* is a gestural input method that uses patterns of movement between the four corners of an input square. A goal was to design each gesture to mimic the letter it represents, as this criterion is known to quicken users' learning of the gestures [14]. The gestures for the letters a-e are shown in Figure 10.

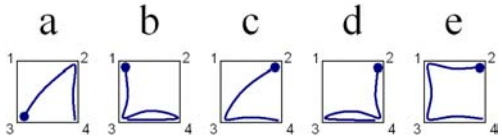


Figure 10. *EdgeWrite* gestures for a-e.

EdgeWrite has been implemented with a variety of devices and methods. One method involves "typing on four keys" [25]. In this case, each gesture is a series of key presses. Using the authors' labels for the corners (see Figure 10), inputting the "e" involves pressing in sequence the keys 2, 1, 3, and 4. The *EdgeWrite* codes for a-z and SPACE are given in Figure 6, fifth column.

As with *LURD-Writer*, the *EdgeWrite* codes are not prefix codes, so a delimiting action is needed to distinguish, for example, 21 (SPACE) from 2134 (e). The authors used an "adaptive timeout" as a delimiter. The duration of the timeout was between 1.2× and 2.0× the duration of key presses; thus, the timeout amounts to a discrete action taking at least as long as one key press. Again, this is akin to a 5th key and is represented in Figure 6 as an asterisk after each code. As with *LURD-Writer*, the need for a distinct delimiting action makes *EdgeWrite* a base-5 text entry method.

The four-key input mode for *EdgeWrite* has *KSPC* = 4.356 (□ in Figure 3). Although this is high, it is important to remember that *EdgeWrite* is primarily a gesture input method. Optimizing the codes for keyed entry was not a design criterion.[3]

Our proposed base-4 text entry method using Huffman coding is *H4-Writer*, shown in the right-most column in Figure 6. Before describing *H4-Writer*, the performance evaluations of *MDITIM*, *LURD-Writer*, and *EdgeWrite* are summarized.

Performance Comparison

Isokoski and Raisamo conducted a longitudinal study of *MDITIM* with five participants [8]. Entry speeds were about 2-3 wpm for the 1st session, representing about 30 minutes of practice. Speeds increased to about 7-8 wpm in the 10th session.

Torsten and Nordmann reported entry speeds for *LURD-Writer* of 1-3 wpm [2]. Testing involved a single physically-disabled participant. Input used pulses generated by a sensor connected to the user's forehead. The pulses controlled a state machine to generate L, U, R, and D commands. No doubt, higher entry speeds are possible using a manually operated mouse or keyboard.

Wobbrock et al. evaluated their 4-key implementation of *EdgeWrite* in a longitudinal study with five participants. Input used keys on the numeric keypad of a standard keyboard. Participants pressed keys using multiple fingers on their dominant hand. Entry speeds were about 6.5 wpm in the 1st session, increasing to 16.9 wpm in the 10th session.

Taking Stock

The H4 column in Figure 6, at *KSPC* = 2.074, gives what we believe are the most efficient code-to-symbol assignments for 4-key English text entry. There are several caveats, however. One is the tag "non-predictive". This is important because predictive text entry methods (e.g., T9) do no support eyes-free input: The user must attend to the on-going predictive process. The H4 assignments in Figure 3 are non-predictive, and therefore support eyes-free input. Of course, the code-to-symbol assignments must be learned, and this takes time.

Another issue is the limited reach of a model that reduces English to 27 symbols. English, or any other language, is much more than 26 letters and SPACE. Many other symbols are important for text entry, such as accented symbols, punctuation symbols, numerals, and so on. Furthermore, editing commands (e.g., BACKSPACE, DELETE) and mode shifts (e.g., SHIFT, CAPSLOCK) are required to support comprehensive text entry. However, it is difficult to include these in the model, since frequency data for symbols and commands are either

[3] Later refinements in the *EdgeWrite* gesture set by Wobbrock and Myers yield *KSPC* rates of 4.483 [23] and 4.258 wpm (see http://depts.washington.edu/ewrite/downloads/EwChart.pdf).

unavailable or domain specific. So, while the modeling process above is robust, it is limited. Clearly, converting the H4 code-to-symbol assignments in Figure 6 into a comprehensive text entry method requires further design. For this, we introduce *H4-Writer*.

Design of H4-Writer

The right-most column in Figure 6 shows the code-to-symbol assignments for *H4-Writer*. The same algorithm was used as for the H4 codes in the second column; however, a trick was used. Note the absence of any code beginning with "L" for *H4-Writer*. Thus, the top-level node in the tree has three branches (URD), instead of four. This was achieved by tricking the algorithm: A fake symbol with an arbitrarily high frequency was inserted in the symbol-frequency list. With this mutation, the algorithm assigned an entire top-level branch to the fake symbol. Of course, this reduces the efficiency of the codes for the core symbols (a-z, SPACE). The loss in efficiency appears as an increase in *KSPC*. As seen in Figure 6, *KSPC* = 2.321 for *H4-Writer*. Remarkably, this is only 12% higher than *KSPC* = 2.074 for the base-4 Huffman codes. This occurs simply because the top nine symbols for *H4-Writer* require only two codes (viz. keystrokes) yet encompass 72% of English (see frequencies in Figure 2). By comparison to H4, the *KSPC*s for the other methods in Figure 6 are higher by 47% (*MDITIM*), 61% (*LURD-Writer*), and 110% (*EdgeWrite*).

The L-branch for *H4-Writer* was hand-coded, with the goal of designing a comprehensive and consistent 4-key text entry method. A hand approach was used due to the absence of reliable frequency data for special symbols and commands, as noted above. Examples from the L-branch for *H4-Writer* are given in Figure 11. BKSP, ENTER, and SHIFT are assigned short two-key sequences, since it is known that these keys are among the most frequent of all keys used in desktop applications [21]. CAPS_LOCK, period, and comma are assigned three-key sequences.

Symbol	H4-Writer L Branch	Symbol	H4-Writer L Branch
[Bksp]	LL	-	LDDLD
[Enter]	LU	%	LDDUL
[Shift]	LR	$	LDDUU
[CapsLock]	LDL	[SymLock]	LDDUR
.	LDU	0	LDDDUL
,	LDR	1	LDDDUU
?	LDDLL	2	LDDDUR
;	LDDLU	3	LDDDUD
'	LDDLR	etc.	

Figure 11. Coding for *H4-Writer* commands and special symbols.

At *KSPC* = 2.321, *H4-Writer* is positioned very close to Huffman's "minimum redundancy line" in Figure 3 (see ● in figure). Although this is promising, it remains to be seen how well *H4-Writer* will perform for text entry. We are interested in the usual metrics for speed and accuracy, but also in the learning process and in the entry of uppercase letters, numerals, and punctuation symbols.

METHOD

The evaluation of *H4-Writer* proceeded in two parts. The first part was a longitudinal study with ten sessions. Users entered phrases of text selected at random from a set. The phrases included only lowercase letters and the SPACE character. Such a procedure is high in internal validity, but low in external validity, necessitating the second part of the study. The second part involved an additional session using a phrase set that included uppercase letters, digits, and punctuation symbols. This final session provided an opportunity to test *H4-Writer*'s viability as a general-purpose text entry method.

Participants

As is customary with longitudinal evaluations [e.g., 8, 15, 25], fewer participants were used but testing involved many sessions. Six participants were recruited from two local university communities. There were 4 males and 2 females with a mean age of 27 (*SD* = 11.0). All were experienced users of desktop computers, reporting 10-16 hours of daily usage. None had prior experience with *H4-Writer* or the other methods in Figure 6.

Apparatus

The apparatus included hardware and software. The software was written in Java and ran on a notebook PC running Microsoft *Windows XP*. Figure 12 shows a screen snap of the software midway through the input of a phrase of text. Four virtual keys (in red boxes) show the symbols accessible along each branch of the Huffman code tree. The contents of the virtual keys are updated with each key press, reflected the decent through the Huffman tree. The key renderings are important for novices, who visually follow the progression of symbols on the virtual keys during input. However, as learning

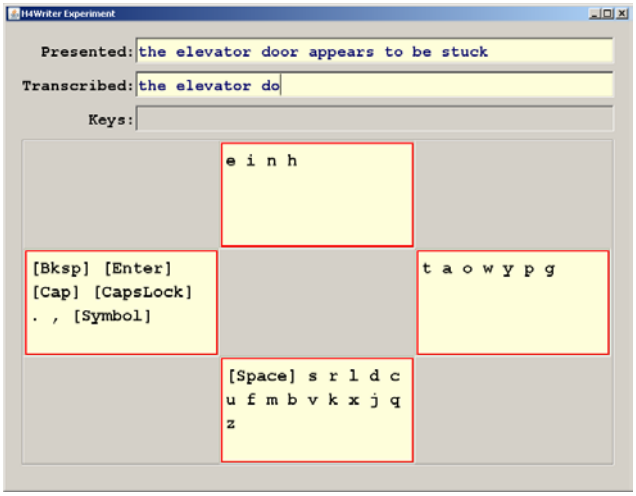


Figure 12. Screen snap of the *H4-Writer* software.

progresses, the need to view the virtual keys diminishes. For example, "the" requires RR (t), UL (h), UU (e), DD (SPACE), as given in Figure 6. These patterns are quickly learned. Words involving less common letters take more

time to learn. An important motivation for the experiment was to empirically measure the learning progression.

Mistakes, if noticed, were corrected by entering BKSP (LL) followed by the button presses for the correct character. At the end of the trial, the ENTER (LU) had to be typed. Following each trial, to help motivate participants, a pop-up window appeared showing the presented and transcribed text phrases, as well as the entry speed and error rate.

User input to *H4-Writer* requires four physical keys or buttons. For this, we used an Xtreme Gaming *Sector 7* game controller, as seen in Figure 13.

Figure 13. Xtreme Gaming *Sector 7* game controller. Physical input uses the four thumb-operated buttons on the right-hand side.

The game controller was held in the usual manner with both hands. Text was entered by the right thumb using the four buttons on the right side of the controller. The button assignments were as expected (e.g., 4 = L; see Figure 13).

The game controller interfaced to the host via a USB connection. Software from Xpadder (www.xpadder.com) provided the necessary mapping of game controller button presses to system key events, as required for the *H4-Writer* experiment software.

Procedure

Each participant was briefed on the goals of the experiment and on the operation of *H4-Writer*. The system was demonstrated by the experimenter, after which the participant entered a few warm-up phrases.

Sessions were scheduled with no more than two sessions on one day and no more that two intervening days between sessions. Each session was divided into 12 blocks with 5 phrases/block. The first two blocks were deemed warm-up and discarded. Rest breaks were allowed between phrases and blocks at the discretion of the participant. Sessions at the beginning of the experiment lasted about 50 minutes. Toward the end, sessions lasted about 35 minutes.

Overall there were 11 sessions of text entry. These were divided into two parts, described as follows.

Part I – Basic Test of H4-Writer

Sessions 1-10 used phrases drawn at random from a set of 500 phrases [12]. The phrases included only lowercase letters, with no punctuation. Part I constituted a longitudinal study and was intended as a basic test of *H4-Writer*, focusing on learning, and on input of the core 26 letters, as well as SPACE, BACKSPACE, and ENTER.

Part II – Extended Session

A single extended session was performed after completion of Part I to test the viability of *H4-Writer* as a comprehensive text entry method. The 11th session used a different phrase set containing 100 phrases. The phrases included a mixture of uppercase and lowercase letters, as well as special symbols. A few examples follow:

```
Can I skate with my sister today?
Beware the ides of March!
Don't say anything.
Play it again, Sam.
Yes - you are very smart!
```

The goal for Part II was to improve external validity by using *H4-Writer* in a context typical of expected usage. Input involved accessing the L-branch of the *H4-Writer* code tree to input special symbols (see Figure 11).

For both parts, participants were instructed to proceed as quickly and accurately as possible and to correct any errors they noticed as input proceeded. Error-free transcribed text was desired, but not essential.

Design

As the experiment was longitudinal and was intended as an initial test of *H4-Writer*, there was no alternative input method as a point of comparison. Session, however, served as an independent variable, primarily to gauge the progress of learning of the participants. The dependent variables were entry speed (words per minute), error rate (%), and keystrokes per character (*KSPC*).

Overall, the analyses were based on 6 participants × 11 sessions × 10 blocks/session × 5 phrases/block = 3,300 phrases of input.

RESULTS AND DISCUSSION

Part I – Basic Test of H4-Writer

Text entry speed in Part I started with a mean of 7.7 wpm in Session 1 and finished at 20.4 wpm in Session 10 – an increase of 165%. The improvement with practice is evident in Figure 14, which shows the progress overall and for each participant.

We consider the session-ten entry speed in Figure 14 the most important overall result of this experiment. Not only is 20.4 wpm a respectable text entry speed, it was achieved using one thumb on four buttons (see title of this paper).

Figure 14 also shows the power law of learning over the ten sessions:

$$\text{Entry Speed} = 10.0\, n^{0.33} \text{ wpm}$$

where n is a unit of about 40 minutes of practice. The model explains 96.3% of the variation in observations, according to R^2.

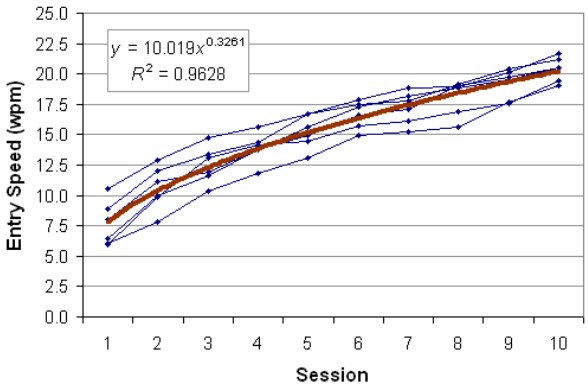


Figure 14. Entry speed (wpm) by Session, overall and for each participant.

The closest point of comparison to our result is in Wobbrock et al.'s test of *EdgeWrite* using four-keys on a numeric keypad [25]. Using 5 participants, they reported a mean of 16.9 wpm on session 10. However, entry involved the use of four fingers. For our experiment, by contrast, text entry involved one thumb only. Nevertheless, *H4-Writer* is about 20% faster. This is expected given the differences in the computed *KSPC* between *EdgeWrite* and *H4-Writer* noted earlier (see Figure 6). Simply put, in terms of key-stroking requirements, *H4-Writer* is more efficient than *EdgeWrite*. So, a higher entry speed is fully expected. Of course, it takes an experiment with users, such as reported here, to put an empirical perspective on this.

Entry speed alone does not provide a complete story. It is important also to consider accuracy or error rates. Over the 10 sessions, the mean error rate was very low: 0.69%. This figure is based on the mean of the minimum string distances (MSD), computed as character-wise comparisons between the presented and transcribed text phrases. Figure 15 shows the error rate result overall and by participant.

The result for participant P3 is highlighted in Figure 15. This participant seemed to have a preoccupation with speed and this surfaced as a swelling in error rates, with a Session-5 high of 5.2%. This was pointed out to the participant, along with a suggestion to relax and focus a bit more on accuracy. The effect was immediate. P3's error rates were in line with the other participants for Sessions 6-10. Although it might be reasonable to label P3 an outlier and exclude the data, the effect overall seemed minor, so the data stand.

Although the error rate results in Figure 15 look slightly more erratic than the entry speed results in Figure 14, this is typical. Errors are discrete events and tend to occur in clusters. In fact, of the 3,000 total phrases represented in Figure 15, 2,588 (86.3%) were error free; i.e., error rate = 0%. Errors when they did occur tended to produce single-phrase chunks.

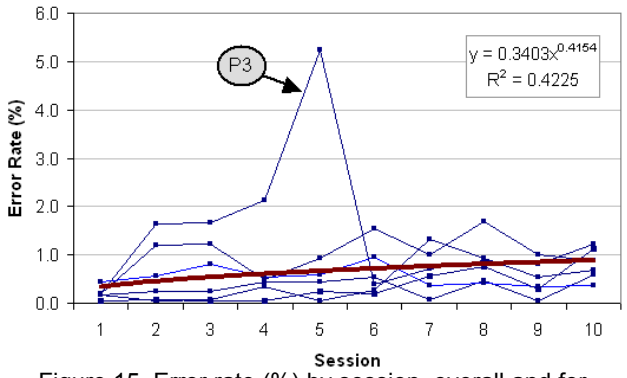


Figure 15. Error rate (%) by session, overall and for each participant. See text for discussion.

Keystrokes per character (*KSPC*) is not only a useful statistic to compare text entry methods a priori (see Figure 6), it is a useful dependent measure. If a participant enters text perfectly, then the observed *KSPC* equals the predicted (i.e., calculated) *KSPC*. However, if errors occur, the situation is different. Errors that remain are reflected in the MSD statistics, as just examined. Errors that are corrected require extra keystrokes and this yields an increase in *KSPC* above the predicted *KSPC*. The result for *KSPC* is given in Figure 16.

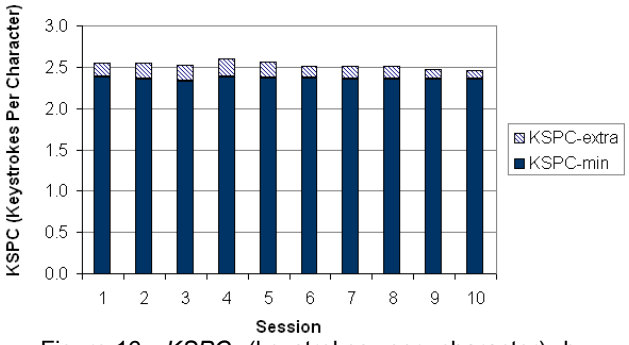


Figure 16. *KSPC* (keystrokes per character) by session.

The observed *KSPC* is divided into two parts: *KSPC-min* and *KSPC-extra*. *KSPC-min* is the predicted *KSPC* based on the actual phrases entered. The heights of the solid bars for *KSPC-min* are close to the value of *KSPC* = 2.321 given earlier (see *KSPC* value for *H4-Writer* in Figure 6). Slight deviations occur because the values are computed from the actual phrases entered, rather than for English in general.

Of more interest here is *KSPC-extra*, which quantifies the additional keystrokes to correct errors. These keystrokes are combinations of BKSP (LL) and the keystrokes for the correct characters. The added height to the bars in Figure 16 is small. The mean of the *KSPC-extra* values is 0.134, suggesting an overhead of approximately 1 keystroke every 7.4 keystrokes to correct errors. Our conclusion is simply that participants' behaviour overall was neither reckless nor untoward. Participants made a small number of errors and they tended to correct most of those that did occur.

Although we consider *KSPC* to provide a suitable perspective of error correction performance (because our design was directed at reducing *KSPC*), we also calculated the corrected error rate, which averaged 3.7%.

The on-screen keyboard was visible throughout the experiment. This was essential while learning *H4-Writer*; however, visual access of the on-screen keyboard diminished as expertise developed and entry speeds increased. Once entry speeds exceeded about 15 wpm, *H4-Writer* became fully eyes-free as the keying rate (3-4 keystrokes per second) was too high for any visual search to occur.

Part II – Extended Session

By the time participants reached the 10th session, they were well versed in the operation of *H4-Writer*, having reached an average entry speed just over 20 wpm. The 11th session was designed to test their ability to leverage existing skill and delve into the L-branch of the *H4-Writer* code tree to input special symbols.

Session 11 proceeded just like Sessions 1-10, except a different phrase set was used. Punctuation and other special symbols were required (examples given earlier). Clearly, performance would suffer, but how much so? The result for entry speed is shown in Figure 17.

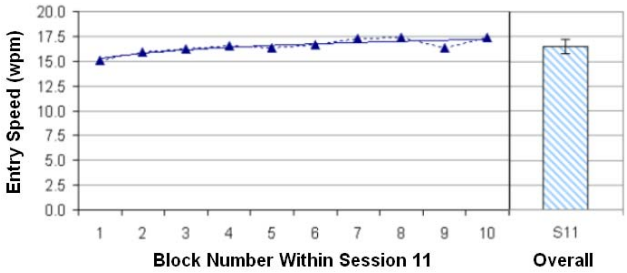


Figure 17. Session 11 entry speed (wpm) by block and overall.

The overall entry speed for Session 11 was 16.5 wpm, varying from 15.1 wpm in the 1st block to 17.4 wpm in the 10th block. Bear in mind that there were two warm-up blocks for which the data were discarded. (This was true for all sessions, as noted earlier.) Therefore, the initial performance while entering special characters was slightly worse than revealed in Figure 17. However, given that the total time for Session 11 was about 35 minutes, participants clearly adjusted quickly to the added demand of entering special symbols. By the end of Session 11, entry speeds were within 3 wpm of the speeds in Session 10 with the lowercase-only phrase set.

Error rates and *KSPC* results were also encouraging for Session 11. The error rate overall was <1% and *KSPC-extra* was similar to that observed in Sessions 1-10.

Participants liked *H4-Writer*, but found the experiment long and demanding. An alternative methodology for the longitudinal study might have worked better. For example, participants could have used *H4-Writer* daily in an unstructured manner, with formal testing occurring at designated intervals, such as one test per week over a few months of use.

CONCLUSIONS

We have presented the design and evaluation of *H4-Writer*, a text entry method that uses one thumb and four buttons. Given the small form factor, the observed entry speed of just over 20 words per minute after 10 sessions of practice (≈40 minutes each) is encouraging.

H4-Writer is one example in a family of keyed input methods using Huffman coding to minimize *KSPC* for non-predictive text entry. A variety of other implementations are being considered, for example, using stylus input, a joystick, a touchpad, head tracking, and eye tracking.

As demonstrated in our design analysis, the use of a Huffman code tree allows extra symbols and extra commands to be added without the need for additional buttons. And so, the method has potential for use in any situation where only a small number of input primitives are available yet comprehensive text entry is required.

ACKNOWLEDGEMENT

Thanks are offered to Torsten Felzer for providing the enhanced phrase set for the 11th session.

REFERENCES

1. Dunlop, M. D., Watch-top text-entry: Can phone-style predictive text entry work with only 5 buttons? *Proceedings of MobileHCI 2004*, (Heidelberg, Germany: Springer-Verlag, 2004), 342-346.

2. Felzer, T. and Nordmann, R., Alternative text entry using different input methods, *Proceedings of the ACM SIGACCESS Conference on Computers and Accessibility - ASSETS 2006*, (New York: ACM, 2006), 10-17.

3. Gong, J. and Tarasewich, P., Alphabetically constrained keypad designs for text entry on mobile phones, *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2005*, (New York: ACM, 2005), 211-220.

4. Green, N., Kruger, J., Faldu, C., and Amant, R. S., A reduced QWERTY keyboard for mobile text entry, *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2004*, (New York: ACM, 2004), 1429-1432.

5. Harbusch, K. and Kühn, M., Towards an adaptive communication aid with text input from ambiguous keyboards, *Proceedings of the European Association for Computational Linguistics - EACL 2003*, (Cambridge, MA: MIT Press, 2003), 207-210.

6. Huffman, D. A., A method for the construction of minimum redundancy codes, *Proceedings of the IRE*, *40*, 1952, 1098-1101.

7. Hwang, S. and Lee, G., Qwerty-like 3x4 keypad layouts for mobile phone, *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2005*, (New York: ACM, 2005), 1479-1482.
8. Isokoski, P. and Raisamo, R., Device independent text input: A rationale and an example, *Proceedings of the Working Conference on Advanced Visual Interfaces - AVI 2000*, (New York: ACM, 2000), 76-83.
9. MacKenzie, I. S., KSPC (keystrokes per character) as a characteristic of text entry techniques, *Proceedings of the Fourth International Symposium on Human-Computer Interaction with Mobile Devices - MobileHCI 2002*, (Berlin: Springer, 2002), 195-210.
10. MacKenzie, I. S., Mobile text entry using three keys, *Proceedings of the Second Nordic Conference on Human-Computer Interaction - NordiCHI 2002*, (New York: ACM, 2002), 27-34.
11. MacKenzie, I. S. and Soukoreff, R. W., Text entry for mobile computing: Models and methods, theory and practice, *Human-Computer Interaction*, *17*, 2002, 147-198.
12. MacKenzie, I. S. and Soukoreff, R. W., Phrase sets for evaluating text entry techniques, *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2003*, (New York: ACM, 2003), 754-755.
13. MacKenzie, I. S. and Tanaka-Ishii, K., Text entry with a small number of buttons, in *Text entry systems: Mobility, accessibility, universality*, (I. S. MacKenzie, and Tanaka-Ishii, K., Eds.). San Francisco, Morgan Kaufmann, 2007, 105-121.
14. MacKenzie, I. S. and Zhang, S. X., The immediate usability of Graffiti, *Proceedings of Graphics Interface '97*, (Toronto: Canadian Information Processing Society, 1997), 120-137.
15. MacKenzie, I. S., Zhang, S. X., and Soukoreff, R. W., Text entry using soft keyboards, *Behaviour & Information Technology*, *18*, 1999, 235-244.
16. Pavlovych, A. and Stuerzlinger, W., Less-Tap: A fast and easy-to-learn text input technique for phones, *Proceedings of Graphics Interface 2003*, (Toronto: Canadian Information Processing Society, 2003), 97-104.
17. Perlin, K., Quikwriting: Continuous stylus-based text entry, *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST '98*, (New York: ACM, 1998), 215-216.
18. Porta, M. and Turina, M., Eye-S: A full-screen input modality for pure eye-based communication, *Proceedings of the ACM Symposium on Eye Tracking Research and Applications - ETRA 2008*, (New York: ACM, 2008), 27-34.
19. Ryu, H. and Cruz, K., LetterEase: Improving text entry on a handheld device via letter reassignment, *Proceedings of the 19th Conference of the Computer-Human Interaction Special Interaction Group (CHISIG) of Australia - OZCHI 2005*, (New York: ACM, 2005), 1-10.
20. Silfverberg, M., MacKenzie, I. S., and Korhonen, P., Predicting text entry speed on mobile phones, *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2000*, (New York: ACM, 2000), 9-16.
21. Soukoreff, R. W. and MacKenzie, I. S., Input-based language modeling in the design of high performance input systems, *Proceedings of Graphics Interface 2003*, (Toronto: CIPS, 2003), 89-96.
22. Tanaka-Ishii, K., Inutsuka, Y., and Takeichi, M., Entering text with a four-button device, *Proceedings of the 19th International Conference on Computational Linguistics - Vol. 1*, (Morristown, NJ: Association for Computational Linguistics, 2002), 1-7.
23. Wobbrock, J. O. and Myers, B. A., Trackball text entry for people with motor impairments, *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2006*, (New York: ACM, 2006), 479-488.
24. Wobbrock, J. O., Myers, B. A., and Kembel, J. A., EdgeWrite: A stylus-based text entry method designed for high accuracy and stability of motion, *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST 2003*, (New York: ACM, 2003), 61-70.
25. Wobbrock, J. O., Myers, B. A., and Rothrock, B., Few-key text entry revisited: Mnemonic gestures on four keys, *Extended Abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems - CHI 2006*, (New York: ACM, 2006), 489-492.