Proceedings of the Fourth IASTED International Conference
Telehealth and Assistive Technologies (Telehealth/AT 2008)
April 16-18, 2008 Baltimore, Maryland, USA

AUTOMATIC AND SELF-PACED SCANNING FOR ALTERNATIVE TEXT ENTRY

Torsten Felzer, Bruno Strah, and Rainer Nordmann
Department of Mechatronics in Mechanical Engineering
Darmstadt University of Technology
Petersenstr. 30, D-64287 Darmstadt, Germany
email: {felzer,strah,nordmann}@mim.tu-darmstadt.de

ABSTRACT
This paper introduces a novel software for alternative text entry implemented as part of the framework of the HAnds-free Mouse COntrol System HaMCoS. The application lets the user 'write' individual characters as well as completions of unfinished words by iteratively selecting one of multiple options. In addition to representing an instance of conventional *scanning* (also called *indirect* text entry), where a selection marker is automatically advanced to the next available option after a certain *dwell period* has elapsed, the program may be configured to have the user explicitly trigger any marker changes. One main purpose of this paper is to evaluate the two scanning techniques, *automatic* and *self-paced*, not only regarding the achievable entry rate, but also concerning usability. With this in mind, several comparative experiments which involve transcribing a specific text fragment have been conducted. The results demonstrate that both variants have advantages *and* disadvantages. In fact, it turned out that, although being more stressful for the user, automatic scanning allowed faster operation. Besides, it can be shown that the presented software application indeed is a promising alternative for someone who is unable to use any form of *direct* text entry.

KEY WORDS
Human-computer interaction, Bioelectric sensors, Word completion, Morse code.

1 Introduction

Communication is the key to getting into contact with other people and having a social life, and the primary medium of communication is natural language, spoken or written. There are – of course – exceptions, e.g., involving pictures, sign language, or any other form of non-verbal communication, but those are beyond the scope of this paper.

When concentrating on *written language*, the computer can be used as an aid which provides, among other things, a means to exchange emails or to access the Internet. Producing written language is therefore of tremendous importance for operating a PC (also regarding the Internet, e.g., to get to Web pages that *cannot* be reached by simply clicking on a symbolic link).

The problem is that written language is overwhelmingly powerful, yet consists of numerous strings of many different signs or symbols (the *characters*). The standard tool used for text entry in a computer, a manual keyboard, which assigns one or a combination of two actuators (*keys*) to every such symbol, seems to be the perfect instrument for the job[1].

Unfortunately, not every potential user is able to operate a standard keyboard. In particular, persons with severe physical disabilities – who may have to confine themselves to a very small number of keys (one or two), or who may not be able to use their hands at all – often are in need of alternatives, if they want to employ a computer.

This paper introduces such an alternative – called Scan_LURD (the reason for the name will become clear in section 4) – in thorough detail. The tool was developed to interface with the HAnds-free Mouse COntrol System HaMCoS [4], and its structure was inspired by common one-key solutions.

After a closer look at the pros and cons of *scanning* applications known so far, the HaMCoS software and its input interface are briefly described, finally followed by a characterization of the Scan_LURD application in section 4. Section 5 contains an outline of the transcription experiments conducted to investigate the usefulness of the tool which is followed by a discussion of the results. The conclusion wraps the paper up by furnishing a summary and some thoughts on future work.

2 Background

Pressing a key on a standard keyboard or clicking on a button of a software keyboard means *directly* initiating the desired result, e.g., the corresponding character appearing in some edit field. On the other hand, *scanning* applications *suggest* characters (or sets of characters) to the user who may select them, that is why this type of language production is called *indirect text entry*.

To select characters from an alphabet A containing n characters using scanning works as follows: the n characters are split across p_1 (usually disjoint) subsets

[1] *Word completion* [1] or *T9® solutions* [2] (see also [3]) show that there still is some redundancy, but unless the computer is to be used in a mobile environment, it does not hurt to have a 'full-blown' keyboard.

$A_1, A_2, ..., A_{p_1}$, which are presented to the user (mostly in some *visual* form). The subsets might also be called *selectable options*, since they are cyclically highlighted for a *dwell period* τ, one after the other. The highlighted option can then be selected by a specific input signal (e.g., the actuation of a certain switch).

If the subset corresponding to the selected option (say, A_k) only contains one single character, that character is copied to some output pane. Otherwise, scanning continues by *descending* one level and splitting the n_k characters of subset A_k across p_2 new subsets $A_{k1}, A_{k2}, ..., A_{kp_2}$. They are highlighted again, one by one, and let's say the user this time selects the $l-$th subset A_{kl}.

Again, for an *atomic* A_{kl} this results in outputting the corresponding character, whereas, for $n_{kl} > 1$, the process cycles over p_3 new subsets ($\subsetneq A_{kl}$) $A_{kl1}, A_{kl2}, ..., A_{klp_3}$. All this is repeated until finally an atomic subset is selected, and the entire process starts anew at the top level.

The (maximum) number of levels and the number of subsets at each level decide on the average time m to 'write' a character and thus on the quality of the individual scanning implementation. Furthermore, it must be said that the dwell period τ directly influences m, but there is a trade-off: the shorter τ is chosen, the faster the options are cycled (which potentially decreases m), but also, the higher the probability becomes that the user misses an option and has to wait for an entire cycle.

2.1 Linear Scanning

The simplest (yet slowest) scanning technique is called *linear scanning*. Here, the n characters of the alphabet are presented one by one to the user, i.e., each of the 'subsets' $A_1, A_2, ..., A_{p_1}$ contains only one character, and $p_1 = n$.

It is very easy to calculate the average time m to 'write' any character (assuming that each one is equally important):

$$m(\tau) = \frac{\tau \sum_{i=1}^{n} i}{n} = \frac{\tau}{2}(n+1).$$

2.2 Row-Column Scanning

A very popular technique often applied in the context of on-screen keyboards is *row-column scanning*. Let A be visualized by means of a rectangular grid with p rows and $q = n/p$ columns. Row-column scanning first highlights the rows (cyclically), until the user selects one of them. Following that, the n/p characters in the selected row are highlighted one by one.

The number of levels needed to 'type' a character is always 2, and the two levels are scanned linearly, so the average time needed is (assuming equality of all characters):

$$m(\tau, p) = \frac{\tau}{2}(p+1) + \frac{\tau}{2}(n/p+1) = \frac{\tau}{2}(p + n/p + 2).$$

In order to minimize m, it has to be differentiated:

$$\frac{dm}{dp} = \frac{\tau}{2}(1 - n/p^2).$$

Setting the derivative to 0 yields the minimum for $p = \sqrt{n}$. In other words, given that every character is counted with equal weight when computing m, it follows that m is minimal, if the on-screen keyboard is (approximately) arranged quadratically (which is not really surprising).

However, for 'normal' languages (at least for the English language), the 'equality assumption' is wrong, as the characters are used at different frequencies. In this respect, the actual *layout* of the on-screen keyboard also affects m. Common layouts comprise the ('historic') QWERTY-layout, arrangements in alphabetical order, or character sets based on Morse code (i.e., frequency-related) – see also [5].

2.3 Binary Scanning

In *binary scanning*, the number of subsets at each level is two, so selection only requires deciding for one of two alternatives. For example, the scanning application could always cut the remaining (sub)set in half and have the user decide which of the two halves contains the desired character. Repeating this procedure until the remaining subset only contains a single character, would result in an equal number of levels necessary to choose any character (at least if n is a power of 2). It follows for the number L of levels:

$$n = 2^e \Longrightarrow L = e = \text{ ld } n,$$

so the number of levels equals the *binary logarithm* of n.

Let A be the set of the 26 Roman letters. Choosing the letter 'H' amounts to the following 5 binary decisions ($\lceil \text{ld}(26) \rceil = 5$):

$$\{\text{A, ..., M}\} \vee \{\text{N, ..., Z}\} \Longrightarrow \{\text{A, ..., F}\} \vee \{\text{G, ..., M}\} \Longrightarrow$$

$$\{\text{G, H, I}\} \vee \{\text{J, K, L, M}\} \Longrightarrow \{\text{G}\} \vee \{\text{H, I}\} \Longrightarrow \{\text{H}\} \vee \{\text{I}\}$$

Another possibility for binary scanning is *not* to cut the remaining subset in half *on purpose*, thereby favoring the (possibly more frequently used) characters in the smaller subset over the other ones. A theoretical discussion of how to build an optimal binary spelling interface would exceed the scope of this paper – a detailed investigation can be found in [6].

A spelling device based on the same idea as binary scanning is described in [7]. However, the user does not operate the device with a specific *switch* as in 'conventional' scanning. Rather, she or he decides between the two alternatives with the help of a *mental* yes/no-answer (detected by inspecting the user's EEG signal). Anyway, that is basically the same thing – only that EEG processing is awfully slow and terribly sensitive to noise (e.g., [8]).

3 Contraction-based Control

A person with a severe physical disability is often not capable of reliably operating the *usual* input devices, such as standard keyboard or manual mouse, for controlling a computer. In order to take advantage of a computer anyway – e.g., as a communicative tool – a *bio-signal interface* can be used instead.

The idea behind such a system is to monitor the time series belonging to a suitable biological function, e.g., brain-waves, with the goal to find recurring patterns in the stream of input signals. The input stream is therefore chopped up into small 'chunks' (or *samples*) which are analyzed, e.g., with a neural network classifier. If such an input sample matches one of a limited number of predefined 'target' samples, the interface issues a control command, so the user can trigger commands by willfully altering the monitored bio-signal.

As a consequence, the main requirement as to the underlying biological function is that the user has to be able to influence it at will. Examples include EEG signals – establishing a basis for so-called *brain-computer interfaces* (e.g., [9]) – or eye movements – e.g., exploited in *eye-typing systems* (e.g., [10]).

The interface implemented in HaMCoS has been proven to yield a favorable solution in the past, particularly concerning noise insensitivity and ease of use. The system looks at the muscular activity of a single muscle of choice, e.g., the brow muscle, and tries to detect intentional muscle contractions.

The above mentioned samples in that case are simply one-dimensional values representing the amplitude a of the muscular activity at the current point in time $t = t_0$, and the amplitude values are 'classified' (by plain comparison) into the two 'classes' above and below an adjustable threshold d. An *intentional contraction* is simply defined as the *rising edge* (from 'below' to 'above') at a specific point in time t_1:

$$a(t_1 - 1) < d \wedge a(t_1) \geq d.$$

Furthermore, the contraction events are divided into *single contractions* (SC's) and *double contractions* (DC's), where the latter ones consist of two contractions with an interval I (between the falling edge of the first and the rising edge of the second) which is shorter than a certain *contraction period* τ_1 (see illustration in fig. 1).

The stream of input signals is thus translated into a sequence of SC's and DC's, which may be further processed, for example, to control the mouse cursor of a PC (as briefly described in the next subsection) or to navigate an electrically powered wheelchair [11]. The only thing required from the user is to, e.g., raise the eyebrow[2]. He or she does not even have to care about *exactly how strong* to do that – it merely has to be *strong enough* to exceed the threshold.

[2]This is just an example; any arbitrary muscle can be chosen here.

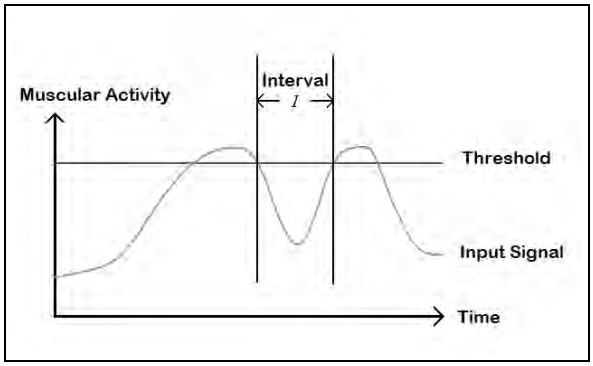

Figure 1. Two intentional contractions with interval I in between

3.1 The HAnds-free Mouse COntrol System

The previously introduced HAnds-free Mouse COntrol System HaMCoS processes the contraction sequence to fully control the mouse cursor on a computer screen under a GUI-based operating system[3].

HaMCoS makes use of the temporal succession of the contractions by defining an *internal state* S which determines the movement direction of the mouse cursor. S is updated each time an intentional contraction occurs and is one of the following: STOP, LEFT, UP, RIGHT, and DOWN. Initially, the STOP state is active (S = STOP), which also means that the mouse does not move. The transitions among the five states (which depend on the respective contraction type, SC or DC) are depicted in fig. 2.

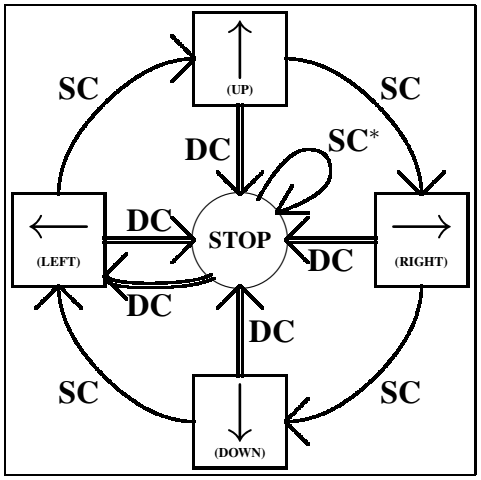

Figure 2. Transition diagram

In short, the transition diagram says that the mouse cursor can be started or stopped with a DC, and that, once the cursor has been set in motion, the user can cycle through the four 'direction states' by issuing SC's.

[3]Currently, HaMCoS exclusively runs under various versions of Windows®, but there are considerations porting it to other platforms.

By applying this technique, the user can move the mouse cursor to any position on the screen completely hands-free. In addition, HaMCoS generates a mouse click at the current screen position whenever the cursor is halted (i.e., S reverts to STOP) – the user is prompted for the *type* of the click (left or right button, single or double, drag start or end, ...) prior to generation.

So in total, HaMCoS is intended to perfectly emulate a two-button mouse, and it requires no more than intentional contractions of a single dedicated muscle.

3.2 The LURD-Writer

HaMCoS comes with a comprehensive framework consisting of several auxiliary applications for text entry, process management, or game playing. The framework also interfaces with standard applications (e.g., Web-browsers) by providing a software keyboard and easy scrolling. The idea behind this is to extend the computer's operating system and to enable someone who cannot use the hands to operate a PC using HaMCoS only.

One of the applications dealing with text entry is the so-called LURD-Writer (see [12]). This auxiliary tool makes entering text somewhat similar to composing Morse code, with the main difference that the selectable characters correspond to sequences of *four* 'bits' instead of just two (dots and dashes).

The selection principle is based on a small square input area with the mouse cursor being placed inside. When set in motion, the mouse cursor 'hits' one of the edges of the input square, unless the user changes cursor direction in time. Each time an edge is 'hit' the cursor is repositioned to the center of the input square and the edge (in form of a *LURD-bit* 'l', 'u', 'r', or 'd') is memorized in a running *LURD-sequence*. The character associated with the LURD-sequence stored at the time, when S reverts to STOP, is finally selected.

The associations between the selectable characters and the LURD-sequences are loosely based on Morse code (at least for the letter characters), in that the production of the LURD-sequence associated with a letter l_1 tends to take longer than that belonging to a different letter l_2, if the Morse code of l_2 is shorter than that of l_1.

The disadvantage of the LURD-Writer is that it represents a whole new way of 'writing' and thus requires some practice. On the other hand, it could be shown that this 'Morse-like' tool may considerably increase the achievable entry rate, compared to a mere on-screen keyboard.

4 Scanning Application Scan_LURD

In principle, HaMCoS is nothing else but a *single-switch* system – the only difference is that it relies on muscle-related input signals instead of a single key of the computer keyboard or simply a physical switch. The idea to realize *the* text entry method for single-switch devices (i.e., *scanning*) as a HaMCoS framework application (i.e., relying on muscle-based input) therefore truly suggested itself.

The outcome of this idea was the Scan_LURD application which is introduced in this section. Like any scanning tool, it lets the user repeatedly select among multiple options – each representing a subset of characters which constantly gets smaller – with the help of intentional muscle contractions. The selection scheme of the tool (in [13] called *containment hierarchy*) is depicted in fig. 3. It matches the encoding of the LURD-sequences mentioned above, which explains the application's name.

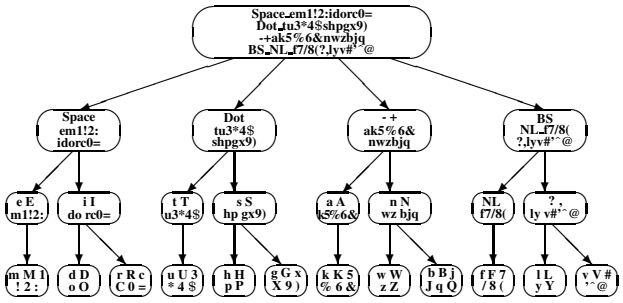

Figure 3. Selection scheme used in Scan_LURD

Initially, the program is in 'automatic scan mode', and the user may select boxes which are (as usual) cyclically highlighted for a *dwell period* τ_2 each[4]. While τ_2 is elapsing, the highlighted box is overlaid with a diminishing selection marker and can be preselected by the user issuing an intentional contraction. The selection is *finalized* if confirmed with another contraction within an additional τ_2 period.

When $\tau_2 \leq \tau_1$, this actually means that highlighted boxes can be selected with DC's, while SC's merely delay the selection marker. This makes the application more fault-tolerant, in that the user may even issue erroneous SC's without causing too much harm.

As already said above, when the selected box is labeled with a non-trivial set of characters, the process *descends* one level, and the characters are split across new boxes. Finally selecting a box containing only one character results in that character being copied to an output pane.

At the beginning of the selection process (i.e., at the *top* or *root level*), the user is presented with a total of five options: four character subsets and one special *menu option*.

4.1 Multiple Input Methods

Scan_LURD is part of the HaMCoS framework – the principal input method is therefore concerned with intentional muscle contractions. For example, in case of the brow muscle, the input signals can be picked up with a piezo-based sensor, attached to the user's forehead with the help of an elastic headband.

[4] τ_2 can be configured *independently* of τ_1.

However, the tool also accepts pressing the 'SPACE' key instead of an intentional contraction (i.e., simulating sensor signals with keystrokes).

4.2 Automatic Versus Self-paced Scanning

In addition to the default '*automatic* scan mode', the program can be switched over to '*self-paced* scan mode'. In the latter mode, the selection marker (or the highlight, respectively) is *not* automatically cycled to the next option after the dwell period τ_2. Rather, the user *actively* has to trigger that with an intentional contraction (so the user can take all the time she or he wants, before deciding whether to select the highlighted option).

After an intentional contraction, the next box (in cyclical order) is marked with a different color, and at the *falling edge*, the contraction period τ_1 starts running (visualized on screen). If the user issues another contraction *before* τ_1 has elapsed, the originally highlighted box is selected – otherwise, the highlight advances to the next box.

In other words, the user can cycle through the available options with SC's and finalizes a selection with a DC. The same principle is, by the way, used in HaMCoS when prompting for the type of mouse click to be issued after the cursor is stopped (see above), or in the menu-driven part of the wheelchair control system introduced in [14].

5 Practical Experiment

In order to evaluate the Scan_LURD application in an everyday life environment, a sufficiently voluminous text fragment was chosen for transcription using various input methods and different text entry applications in several program modes (in the context of Scan_LURD, this particularly relates to the two scanning techniques: 'automatic' and 'self-paced'). It was decided to take the first five verses from the first book of the bible (King James Version).

The (male) subject who executed the transcription task was 36 years old, and he is using a wheelchair since 1988 (because of *Friedreich's Ataxia*). He does have considerable motor problems (which must be kept in mind when looking at the numerical results), but is still able to operate a manual mouse or a standard computer keyboard[5].

The subject (who was well familiar with the HaMCoS input method *and* the entire range of framework applications) was asked to enter the text as fast as possible, while avoiding (or correcting) any 'typing' errors. A maximum amount of word completion assistance was applied in every single trial (which actually meant that each word longer than two characters was not spelled out completely).

6 Numerical Results

The times needed by the subject to complete the transcription task using various combinations of input methods and computer applications are presented in table 1. The subject was granted a pause of one minute between two verses – those *resting minutes* are *not* included in the table.

The values for the contraction period τ_1 ($700ms$) and the dwell period τ_2 ($900ms$) were found empirically.

Table 1. Times needed to enter 462 'characters' (including spaces, newline characters, verse numbering and immediate error correction). 'Manual' refers to the *standard* input devices (conventional mouse or 'SPACE'-key).

Input Method	Text Entry Application	Program Mode	Time Needed
Manual	On-screen keyboard	n/a	35 min.
HaMCoS	On-screen keyboard	n/a	114 min.
HaMCoS	LURD-Writer	n/a	68 min.
Manual	Scan_LURD	'automatic'	82 min.
Manual	Scan_LURD	'self-paced'	90 min.
HaMCoS	Scan_LURD	'automatic'	59 min.
HaMCoS	Scan_LURD	'self-paced'	72 min.

The table basically shows three things: first, manual operation of an on-screen keyboard is about twice as fast as HaMCoS-based scanning; second, the LURD-Writer and Scan_LURD yield similar results for HaMCoS-input (which is no surprise, since they rely on the exact same idea); and third, HaMCoS-operation of Scan_LURD *beats* manual operation for corresponding scanning techniques.

The third point might, however, be specific to the particular subject performing the transcription, as for him, raising the eyebrow is *much* easier than pressing a key.

7 Discussion

When the subject noticed that *automatic scanning* indeed turned out to be faster than *self-paced scanning*, he was quite amazed as he expected it to be the other way round: according to his experiences, 'automatic' scanning caused much more stress, thereby provoking erroneous selections (that had to be corrected aftertwards) and 'highlight misses' (resulting in an unnecessary 'wait cycle').

In the 'self-paced' scanning case on the other hand, the subject always felt to be in total control of what was going on. He could use his own pace, which allowed him to cycle as fast as (he thought) he can without any stress – and this produced much fewer mistakes.

The actual results show that faster cycling *with* errors to be corrected still may require less time than slower cycling without (or at least with *fewer*) errors.

Besides, the subject did not really like the 'confirmation feature' of the 'automatic scan mode'. He stated that constantly having to generate DC's was very exhausting (which actually counteracted the idea of fault-tolerance). Therefore, it was decided to repeat the two experiments using 'automatic' scanning, but this time doing without the confirmation contraction.

[5] It is true that the fact that only one subject participated in this study challenges the statistical validity of the results – anyway, additional experiments are planned for the immediate future.

Table 2. Additional results with *modified* 'automatic scan mode'.

Input Method	Text Entry Application	Program Mode	Time Needed
Manual	Scan_LURD	'single-impulse'	65 min.
HaMCoS	Scan_LURD	'single-impulse'	46 min.

The results of those 'single-impulse scanning' trials is shown in table 2. They reveal a dramatic increase concerning entry rate, and it should be noted that the HaMCoS-trial now really comes close to manual 'on-screen input' – the latter one still is faster, but this time merely by about 30%.

8 Conclusion

A computer program implementing a technique for indirect text entry has been presented. The application has been realized as a framework tool of the HAnds-free Mouse COntrol System HaMCoS and may be operated using intentional contractions of a single muscle of choice (e.g., the brow muscle). Experiments involving the transcription of a moderately long text fragment have been conducted, in order to be able to compare various input methods.

The software runs in two modes: 'automatic' and 'self-paced' scanning. In the former mode, the marker highlighting the selectable options automatically advances to the next option (in cyclical order), each time a certain dwell period has elapsed, while the user *actively* has to trigger cycling in the latter mode. It was found that 'automatic' scanning – although putting more pressure on the user, and thus provoking a larger number of erroneous selections (causing considerable 'correction overhead') – allowed higher entry rates compared to 'self-paced' scanning.

Moreover, the experiments led to a refinement of the 'automatic scan mode' that is still faster and at the same time less exhausting. The final version is truly an alternative for someone who cannot use the hands, as it only adds a temporal overhead of about 30% to the entry rate achieved with a manual mouse and an on-screen keyboard (i.e., *direct* text entry).

The next step for the immediate future is to go to hospitals and rehabilitation centers and have potential users try the HaMCoS software. Furthermore, the authors intend to work on the interface between HaMCoS and their wheelchair control system (see [15]), which are both based on the same input principle

Acknowledgement

This work is supported by DFG grant FE 936/3-1 "The AID package – An Alternative Input Device based on intentional muscle contractions".

References

[1] J. Gong, P. Tarasewich, C. D. Hafner, & S. I. Mackenzie, Work-in-progress: Improving dictionary-based disambiguation text entry method accuracy, *Proc. SIGCHI Conf. on Human Factors in Computing Systems*, San Jose, CA, 2007, 2387–2392.

[2] J. Cardinal & S. Langerman, Designing small keyboards is hard, *Theor. Comput. Sci., 332*, 2005, 405–415.

[3] D. J. Higginbotham, Evaluation of keystroke savings across five assistive communication technologies, *Augmentative & Alternative Communication, 8*(4), 1992, 258–272.

[4] T. Felzer, R. Fischer, T. Grönsfelder, & R. Nordmann, Alternative control system for operating a PC using intentional muscle contractions only, *Proc. CSUN Conf. on Technol. & Persons with Disabil.*, Los Angeles, CA, 2005.

[5] H. Venkatagiri, Efficient keyboard layouts for sequential access in augmentative and alternative communication, *Augment. & Altern. Communic., 15*(2), 1999, 126–134.

[6] M. Tregoubov & N. Birbaumer, On the building of binary spelling interfaces for augmentative communication, *IEEE Trans. Biomed. Eng., 52*(2), 2005, 300–305.

[7] N. Birbaumer, T. Hinterberger, A. Kübler, & N. Neumann, The thought-translation device (TTD): Neurobehavioral mechanisms and clinical outcome, *IEEE Trans. Neural Syst. Rehabil. Eng., 11*(2), 2003, 120–123.

[8] D. J. McFarland, W. A. Sarnacki, T. M. Vaughan, & J. R. Wolpaw, Brain-computer interface (BCI) operation: Signal and noise during early training sessions, *Clinical Neurophysiology, 116*(1), 2005, 56–62.

[9] J. R. Wolpaw, N. Birbaumer, W. J. Heetderks, D. J. McFarland, P. H. Peckham, G. Schalk, E. Donchin, L. A. Quatrano, C. J. Robinson, & T. M. Vaughan, Brain–Computer Interface Technology: A Review of The First International Meeting, *IEEE Trans. Rehab. Eng., 8*(2), 2000, 164–173.

[10] J. P. Hansen, K. Tørning, A. S. Johansen, K. Itoh, & H. Aoki, Gaze typing compared with input by head and hand, *Proc. 2004 Symp. on Eye Tracking Research & Applications*, San Antonio, TX, 2004, 131–138.

[11] T. Felzer & B. Freisleben, HaWCoS: The "Hands-free" Wheelchair Control System, *Proc. 5th ACM SIGCAPH Conf. on Assistive Technologies*, Edinburgh, Scotland, 2002, 127–134.

[12] T. Felzer & R. Nordmann, Alternative text entry using different input methods, *Proc. 8th ACM SIGCAPH Conf. on Comp. and Accessibility*, Portland, OR, 2006, 10–17.

[13] M. Baljko & A. Tam, Motor input assistance: Indirect text entry using one or two keys, *Proc. 8th ACM SIGCAPH Conf. on Comp. and Accessib.*, Portland, OR, 2006, 18–25.

[14] T. Felzer & R. Nordmann, Alternative wheelchair control, *Proc. IEEE-BAIS Symp. on Research on Assistive Technologies*, Dayton, OH, 2007, 67–74.

[15] T. Felzer & R. Nordmann, Consolidating computer operation and wheelchair control, *Proc. 9th ACM SIGCAPH Conf. on Comp. and Accessib.*, Tempe, AZ, 2007, 239–240.