Abstract (incomplete)

BinSpell – commitment to a symbol is withheld until the appropriate level of evidence has been reached. We present 2 interface designs which are possible to control given low bandwidth and noisy channels.

Introduction

Many of the devices designed to make communication possible for individuals with motor disabilities usually require some vestige of neuro-muscular control. Brain computer interfaces (BCI) affords individuals with severe neuro-muscular impairments, direct neural control of external devices. There are a variety of different methods for obtaining brain signals which are suitable for BCI control. We choose here to focus on electroencephalography (EEG) as it is currently the most widely used modality[citation], and the most commercially viable[i] [ii]. One of the primary focuses of BCI research has been to enable motor-limited individuals to control communication devices. Thus, much research has gone into providing BCI-controlled spelling devices, or virtual keyboards, to these individuals.

Control of external devices requires the BCI system be able to differentiate between different patterns in continuous brain signals[iii]. But amplitudes of scalp recorded data are quite small and difficult to distinguish from artifacts[iv]. Muscle contractions, eye movements, and even the heartbeat are a few sources of artifacts which lend to a noisy environment[v]. In addition, EEG has pretty poor spatial resolution, and the tissue which makes up the brain conducts electrical signals from many different neuronal sources. Add the fact that the desired signals are only a small subscale of all brain activity, and you have very low signal to noise ratio.

The number of distinguishable patterns—*brain states*—partially hinges upon classification accuracy[vi]. BCIs using binary classifiers in general reach an accuracy of around 90%[vii], although there are classifiers who perform well below this value. Because BCI bandwidth is already so low, low classification error is critical for acceptable performance of BCI-controlled communication devices. Information transfer rates decrease as accuracy decreases, increasing the user's frustration and undermining the viability of BCI as a communication modality.

Currently, there remains a significant imbalance in bandwidths between a BCI system and its user: very little useful information is successfully extracted from an acquired neural signal for reasons explained briefly above, while feedback to the user can present a great deal of information. Thus in order to enable effective communication in such a noisy, low bandwidth environment, communication aids designed for this environment have to include ways of handling noise and error, while optimizing the use of the available bandwidth.

One important design consideration is the compression of the most amount of information from the user, into the least number of bits. Statistical properties of language can aid in this compression by providing context for compression algorithms. While much work has gone into the improvement of classification methods, statistical properties of language, if properly exploited, can greatly improve effective rates of BCI enabled communication.

We present a virtual keyboard, BinSpell, which is designed for low-accuracy binary classifiers whose bandwidth is additionally reduced by very noisy channels. We use Huffman encoding

techniques to efficiently represent the user's intent as a series of binary choices. We implement a language model based on a trigram character model, and a bigram word model in order to increase the accuracy of predictions we make, by inferring the user's intended output. Redundancy is used to counter the error rate of the classifier, thus also improving the accuracy of the predictions.

A second keyboard design, Circ-O-Spell, presented is an extension of an existing keyboard, Hex-o-spell [viii]. With Circ-O-Spell, we present an alternative method of making selections, which we argue requires fewer steps on average. We also employ a Hidden Markov Model and borrow from Huffman coding to reduce the number of steps to a correct selection, and hence increase communication speed.

Background

Brain Imaging Techniques

There are a variety approaches to the recording of brain activity, which can also serve as direct control and communication pathways from the brain to external devices. Among them are electroencephalography (EEG), magnetoencephalography (MEG), positron emission tomography (PET), functional magnetic resonance imaging (fMRI), and near infrared spectroscopy (NIRS)[ix]. EEG appears to be the most practical, commercially viable option thus far, due to its portability, and comparatively reasonable cost[x]. The limitations of the remaining approaches are due mostly to the fact that they are technically demanding and expensive[xi].

EEG

Electroencephalography (EEG) uses electrodes which sit on the scalp, to measure voltage fluctuations in the brain. The number of sensors used can range from 25 to 128 sensors. Typically, a small area of the scalp is lightly abraded right before the application of a sensor. This is done to reduce the impedance caused by dead skin cells. A conductive gel is applied with each sensor, and each sensor is connected to a wire which feeds into an amplifier. Scalp EEG is widely used in diagnosing many medical conditions such as epilepsy, strokes, and sleep disorders[xii]. But translating those signals into meaningful information is very difficult. One of the largest obstacles is the noise.

Single neurons are more complex than even the most sophisticated neural networks, and brain processes often involve their interactions at multiple scales[xiii]. EEG has inherently low spatial resolution, so the signals picked up from EEG sensors reflect the activity of millions of neurons[xiv]. In addition to brain activity, electrical signals from the scalp can also come from sources such as eye blinks or tongue movement[xv]. Electrical impulses from the heart can actually produce scalp potentials greater than EEG amplitudes[xvi], thus making recorded signals even more difficult to discriminate. Additionally, the electrical and geometrical properties of the brain itself, as well as the skull and scalp, all lend to the quality of recorded signals[xvii]. All of these factors contribute to the extremely low signal to noise ratios in recorded signals.

BCI

A brain computer interface (BCI) is a communication channel between the brain and a computer (fig#). It records changes in electrical activity in the brain, and translates them into control signals. Acquired signals are first amplified, then digitized for analysis. Unwanted artifacts, such as those produced by a heartbeat, and noise, are filtered out, and the target signal

	Hex-O-Spell[4]	P300[5]	P300[6]	Dasher[7]
Classifier (accuracy> 90%)	LDA	Stepwise LDA	SVM	Classification Matrix
Average spelling rate (chars/min)	4.24	~4	11.1	~5.6
Achieved spelling rates (chars/min)	2.3 – 7.6	2.8 – 7.8 (offline testing)	6.67 - 16.7	3.2 - 14.4 (Avg.)

feature is extracted. Built in classifiers characterize brain state by correlating specific mental activities with the target features. The interpreted brain state is commuted into control signals, and sent to the device being operated.

BCI's can be either invasive, such as single-unit recording or electrocorticography (EcoG), or non-invasive, such as EEG or MEG. Invasive techniques provide the most information per channel, but carry the greatest risk, as they require surgery to provide openings in the skull. Most BCI systems are non-invasive, and use EEG signals recorded from the scalp [xviii].

Virtual Keyboards (incomplete)

Generally, virtual keyboards are devices or software, which emulate physical keyboards, but do not have physical sensing buttons. They can be controlled by pointing devices, such as mice, touch pads, photo-electric sensing devices, or active finger tracking methods [xix]. Virtual keyboard key layout can be dynamic, and can readily include words, phrases, or pictures.

Virtual keyboards designed for BCI control are on-screen keyboards. Although the area of software keyboards is limited by the size of the screen on which it is displayed, they can be more adaptable to compact mobile devices.

Related Work (Incomplete – need dasher and images)

P300 Speller

The P300 Speller is an synchronous EEG controlled spelling system. It relies on the P300 event related potentials (ERPs) that result when users respond to specific stimuli generated by the BCI system [xx].

Hex-O-Spell

Hex-O-Spell[xxi],[xxii] is controlled by the Berlin Brain Computer Interface (BBCI). The BBCI is an EEG based system which decodes motor imagery without subject training[xxiii]. Its use of machine learning techniques and its ability to operate at high decision speeds, affords high quality feedback to the user. [xxiv] reports 98% accuracy at an average speed of 1 decision every 2.1s.

Hex-O-Spell keyboard layout is comprised of 6 hexagonal fields surrounding a circle, each containing 5 characters. The center of the circle contains an arrow used to make selections. A language model determines how the letters are arranged within each hexagon.

Hex-O-Spell is mentally controlled by imagined right hand and foot movements. Imagined right hand movement effects a clockwise rotation of the arrow in the center of the circle. Imagined right foot movement ceases rotation, and causes the arrow to extend towards a hexagon. Selection of a hexagon is caused by sustained right foot motor imagery, and causes all other hexagons to be cleared. The 5 characters in the chosen hexagon are then redistributed among the empty hexagons, and the arrow is reset to its minimal length. Parameters such as the arrow turning and growing speeds, can be tailored to the user.

Hex-O-Spell was tested with two subjects who had no training on the BBCI, and who had very little experience with the keyboard. The achieved typing speeds were between 2.3 and 5 chars/ min for one subject, and between 4.6 and 7.6 char/min for the other, for error free completed phrases.

Dasher

Our Approach

Circ-O-Spell

This keyboard is an extension of the Hex-O-Spell keyboard[xxv]. The hexagonal fields are replaced by circular fields, containing sets of symbols (fig#). Circ-O-Spell is intended to be controlled by the output of a binary classifier. State-0 brings about rotation and state-1 achieves selection. When a circle containing a set of symbols is selected, a new layout is presented containing only the symbols in the selected set (fig#). In this new layout, each symbol from the selected set gets its own circle, and symbols for 'back-space' and delete are also presented. Delete removes the most recently output symbol from the text box, exits the new layout, and returns the keyboard to the state prior to the selection of that symbol. The back-space key exits the new layout, and returns the user to the layout immediately preceding entrance into the new layout. If the selected circle contains only one symbol, that symbol is added to the text box at the top of the window, and the user is returned to a layout where all the letters are presented(fig#).

Letter and word frequencies are an important consideration when designing a system for the purpose of enabling communication. They can be used to calculate probabilities which help capture important properties of language, and allow for prediction of a letter, word, or phrase in a speech sequence. N-grams are groups of letters, words, or phrases of size n, which serve as the basis for many probabilistic

language models[xxvi]. N-grams do have shortcomings in that they can only capture dependencies within their own length[xxvii]. They cannot explicitly represent any long range dependencies and are thus unable to distinguish them from noise[xxviii].

Higher order n-grams can provide more information, as they reflect the frequencies of larger units ([xxix]) but tend to introduce more sparsity[xxx]. This sparsity can be resolved, however, with large corpora, smoothing techniques, and the incorporation of contextual information[xxxi xxxii xxxiii]. [xxxiv] argues that most letter level bigram frequency tables reflect unrealistic counts because they do not use the positions of those bigrams within words as contextual information. They demonstrate a marked difference in letter bigram frequencies, depending on their positions within words.

Markov models are models of probabilistic temporal processes, which satisfy the Markov assumption – that is, the current state depends only on a finite history of previous states[xxxv]. The order of a Markov process defines the number of previous states on which the current state is Dependant. The simplest is the first-order Markov process, which assume that the current state is dependant only on the previous state, not any other earlier states[xxxvi]. First-order Markov processes are characterized by their prior probabilities, transition models, and emission models. Borrowing notation from [xxxvii], where X_t denotes a state variable at time t, prior probabilities $P(X_0)$, define the probabilities over the states at the first time step. The transition model is a conditional distribution $P(X_t|X_{t-1})$ describing how the state evolves over time. The observation model is also a conditional distribution $P(E_t|X_t)$ which describes the probabilities on the evidence variables E_t , the output, at every time step. Markov models often make up the formalism behind N-gram models used in natural language processing [xxxviii, xxxix].

Circ-O-Spell uses a first-order Markov Model to attempt to reduce the number of steps needed to make a selection. We describe our calculations below and use the following notation:

X_t	State variable at time t
$C_1(.)$	Count of bigrams found solely at the beginning of a word
$C_2(.)$	Count of bigrams found any other position in a word
x_i	Letter of the alphabet

$$P(X_0 = x_i) = \frac{\sum_{j=1}^{k} C_1(x_i x_j)}{\sum_{i=1}^{k}\sum_{j=1}^{k} C_1(x_i x_j)} \quad (1)$$

$$P\left(X_1 = x_j \mid X_0 = x_i\right) = \frac{C_1\left(x_i x_j\right)}{\sum_{i=1}^{k}\sum_{j=1}^{k} C_1\left(x_i x_j\right)} \quad (2)$$

$$P\left(X_t = x_j \mid X_{t-1} = x_i\right) = \frac{C_2\left(x_i x_j\right)}{\sum_{i=1}^{k}\sum_{j=1}^{k} C_2\left(x_i x_j\right)} \quad (3)$$

The default letter layout is determined by the order of the probabilities calculated by Equation (1). When a circle-set containing the letter of choice is selected, the letter with the highest probability among that set, is highlighted to facilitate faster selection. The arrangement of the letters after a single letter is output into the text box, is determined by Equation (2). Equation (3) describes the conditional probabilities for any states which follow. The frequency tables represent bigram frequencies per 1000 words, of words occurring at least once per million in normal usage, and of more than three letters[xl].

BinSpell

BinSpell fig(#) is also designed to be controlled by the output of a binary classifier. State 0 causes selection of the box on the left and state 1 selects the box on the right. The user is intended to select the box with the symbol of their choice. Symbols offered include single characters, space, delete, and whole words. The symbols for space and delete are not offered, however, until at least one other symbol has been typed. The size and color of symbols are determined by their probabilities, the computation of which we explain later. Symbols with the lowest probabilities are small and blue, the next largest are black. The most likely symbol is the largest and is displayed in red. Whole words are provided in cyan to help distinguish them from single letters. Selection of a box causes all of the symbols to be reordered and redistributed between the boxes (fig#). This selection process is repeated until one of the probabilities surpasses a threshold, and the associated symbol is output into the text box at the top (fig#).

Huffman coding is an algorithm originally designed for the binary encoding of sequences of ASCII symbols. It uses each symbol's frequency of occurrence to build up an optimal way of representing that character as a binary string [xli]. It works by creating a binary tree of nodes, each of which represents a symbol and it's associated probability. The way in which the tree is walked produces codes that represent the symbols. Thus the least number of bits encode the most commonly used symbols. It is most commonly used within lossless compression schemes, such as JPEG, or MP3 [xlii].

We use Huffman coding to determine the members of a box-set, reducing the number of steps it would take to access the most likely symbols. Thus, the left and right boxes reflect loosely the left and right halves of a Huffman tree. Individual letters are alphabetized, however, to facilitate visual scanning.

When a selection is made, the probabilities of all symbols within that box are weighted by the accuracy of the classifier (80% in our case). The probabilities of the symbols in the box that was not chosen are weighted by one minus the accuracy. All of the probabilities are then re-normalized, and a new Huffman tree is generated and used to determine the new layout. The flowchart representing this process can be seen in (fig#)

Selections also cause the background of the selected box to be shaded in orange, while the background

color of the non-selected box becomes white (fig#). If a box is selected in succession, the border of the box is intensified for a few seconds, in order to indicate its selection. Selection of a word outputs the word into the text box. As a policy, a space character is automatically appended to a whole word selected. The selection of delete following an incorrectly selected word causes the entire word to be erased. Then both the text box and the keyboard are returned to the state prior to the incorrect selection.

BinSpell implements a 2nd order, character-level Markov process in parallel with a 1st order, word-level Markov process, as its language model. First-order Markov processes are explained in the previous section. Second-order Markov processes assume the current state depends only on the previous two states, and no other history. We calculate prior probabilities, transition probabilities, and output probabilities for both processes and describe them below. We add to the previous notation:

$C_{1^w}(.)$	Word level unigram counts
$C_{2^w}(.)$	Word level bigram counts
W_t	State variable in word-level Markov process
w_i	A single word
$C_{3^x}(.)$	Letter-level trigram counts

$$P(W_0 = w_i) = \frac{C_{1^w}(w_i)}{\sum_{j=1}^{n} C_{1^w}(w_j)} \tag{4}$$

$$P(W_t = w_j \mid W_{t-1} = w_i) = \frac{C_{2^w}(w_i w_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n} C_{2^w}(w_i w_j)} \tag{5}$$

$$P(X_t = x_k \mid X_{t-1} = x_j, X_{t-2} = x_i) = \frac{C_{3^x}(x_i x_j x_k)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} C_{3^x}(x_i x_j x_k)} \tag{6}$$

Priors at the letter-level are calculated as in Equation (1), utilizing the same bigram frequency counts used in Circ-O-Spell. Word-level unigram counts [citation] determine prior probabilities for each word as in Equation (4). In designing BinSpell, we had to make a choice about how to weigh letter-level and word-level probabilities relative to one another. For lack of information on how to optimally make this choice, we assigned them equal weight, and put off further investigation of this question, to a future date. We chose to restrict the number of words presented to the user to 3, and we explain their selection below. Next, the priors of all the letters and words presented are renormalized. Finally, Huffman coding determines in which box they will be displayed conditional probabilities which determine the state transitions at both levels are calculated after a single symbol has been output. At the letter level, the distribution over the state is determined by the length of the string of text which has

been typed. If a single letter has been typed, the probabilities for the next letter are calculated according to Equation (2). If more than one letter has been typed, Equation (6) describes the conditional probabilities associated with letter-level state transitions.

After a whole word has been typed out, the probabilities for the next word are determined by Equation (5). The last whole word typed is kept track of, with the space character used as a word delimiter. We should mention here that we adopt a policy of calculating the probability for the delete option dynamically with every state transition. For lack of access to key logging data, and thus frequency counts for the delete key, we chose to implement a strategy which is intended to facilitate its access, while not lending to a increase in accidental deletions. Thus, the probability for delete is calculated as the average of the top three probabilities among the symbols to be presented.

The weighting which results with each box selection, as described above, determines the emission probabilities at any given time step. When an emission probability has surpassed a fixed threshold, the symbol associated with this probability is typed out.

We use prefix matching to infer which word the user is most likely attempting to type, given the string which has already been typed. This matching is done against a small dictionary of words, which is initially read in from a text file. The top three probabilities among the closest matches are offered to the user.

Methods

Script Testing

We began by attempting to determine the most effective combination of weighting and thresholding levels. For this purpose, we wrote a python script to automate the typing out of text strings. The script first parsed the string into individual words, using space as delimiter. Then each word was parsed into characters. Both boxes of symbols were tested to see which one contained either the character or word of choice; the word being searched for before the character. called the update(.) function of BinSpell. of 5 phrases, 100 times each. This was done for a simulated classifier accuracy of both 80% and 90%.

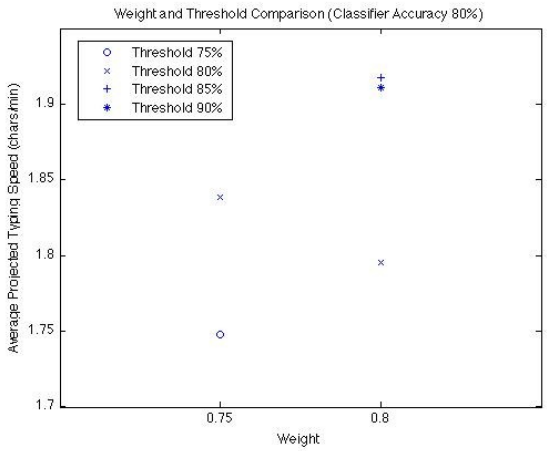
The same 10 phrases were typed out 100 times each, by a script which called BinSpell. The calculated typing speeds assumed 3s per bit.

Experimental Setup

5 subjects(3 females, 2 males, ages 22-28) participated in testing the virtual keyboard. It ran on an Apple MacBook laptop, running Mac OSX. They were seated in a comfortable desk chair.

Training consisted of a 5-10 minute period, consisting of a trial run in which they were asked to type out their names, while directions for using the keyboard were explained to them.

For BinSpell, each participant was informed that they should first scan for their word of choice. If that word had not yet been suggested, they should scan for the next single character of choice. It was also explained that during the shuffling, if a box did not contain the symbol they were attempting to output, they should choose the other box, instead of continuing their attempt



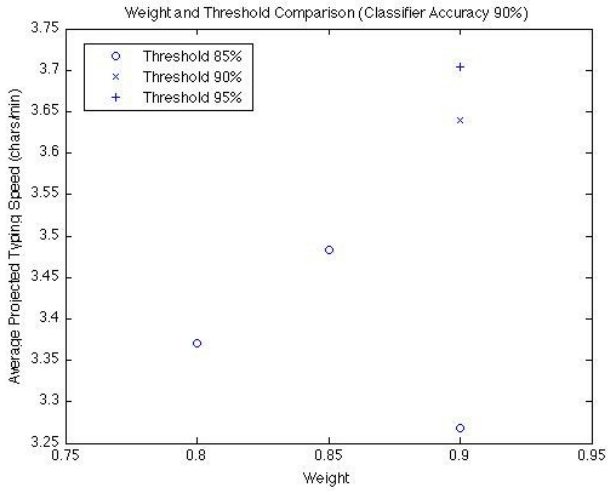


Figure 4b:

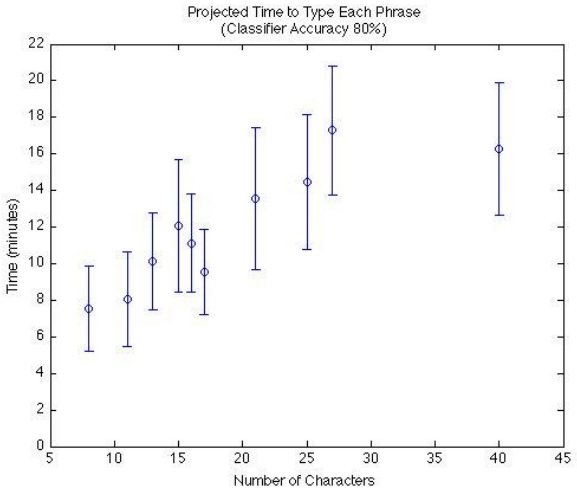


Figure 5a:

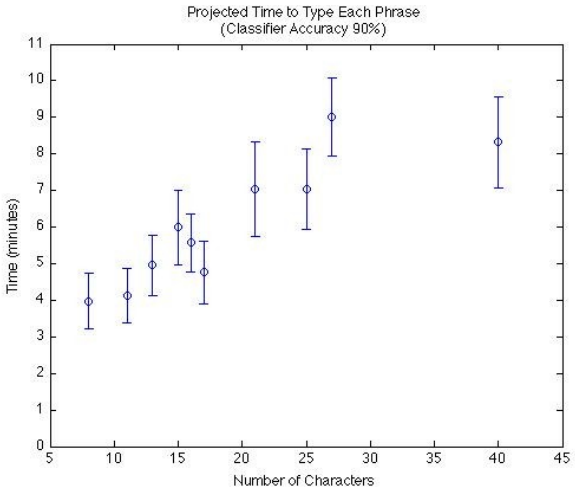


Figure 5b:

to visually locate the letter.

For Circ-O-Spell, each participant was informed that in the case of an incorrect symbol selection, they should simply select the first available circle, in order to get to the delete option.

Participants were provided with 10 phrases selected from among the top 10 Google search queries, for the U.S. on Nov. 13th, 2009[xliii]. These were:

1. tony alamo
2. harrison barnes
3. what drives edward phase
4. palladia
5. we look forward to a world founded upon

6. walmart black friday deals
7. wizard of oz hanging
8. water on the moon
9. david banner
10. bold fresh tour

There were two trials. The entire phrase set was typed once per trial. The first trial simulated classifier accuracy of 80%. The second simulated 90% classifier accuracy. Typing speed was measured for complete error-free phrases. If an error occurred, the delete option had to be selected in order to correct it. The total number of characters typed per subject per trial was 193.

Results

The results of the threshold tests, shown in Figure(3) demonstrated that in both cases, the best performance, in terms of typing speed, was obtained when the classifier accuracy determined the weighting, and the threshold was 5% above the accuracy. The typing speeds which resulted from the script tests can be seen in Figure(4).

The typing speeds achieved by the subjects are shown in (fig#). The typing speeds achieved when the classification error rate was 20% were between .74 and 3.7 chars/min, with an average of 1.5 chars/min. With the error rate of 10%, the typing speeds were between 1.9 and 5.5 chars/min, with an average of 3.1 chars/min.

Discussion

One of the aims of research into BCI typing designs, is to reduce the cognitive load on the user. [xliv] Dasher's zooming interface requires greater intense sustained attention and control for holding a mouse pointer steady than is required by most BCI target acquisition tasks.

The rapid rate of row/column intensification in the 6x6 matrix of the P300 speller presents a difficulty to users[xlv]. It has been suggested that involuntary eye movements in conjunction with the number of symbols presented, may make it difficult for a user to focus on a particular location on the matrix[xlvi].

The added load to the user these features present in a relatively noise-free environment, would be amplified in a noisy environment. In contrast, BinSpell is designed for abbreviated cognitive states and does not require sustained inputs. Although it is a dynamic keyboard, the symbols are presented at a moderate presentation rate. These differences afford less frustration to a user communicating with BinSpell in a sub-optimal environment.

There is evidence to suggest that, given a selection task, using standard keyboard presented less frustration and less visual strain than the use of standard mouse input[xlviixlviii]. Generally,

cursors are controlled using several bytes/s. Thus, using only a fraction of a bit per second, effective control of cursor presents even more challenge to a user. Addition of an extremely noisy environment would make the cursor unmanageable. In the same way, control of key-selection elements which required a continuous control process would be difficult to impractical if that control was effected over a low-bandwidth noisy channel. Thus, BinSpell's lack of reliance on continuous control makes it better suited then keyboards such as Dasher and Hex-O-Spell for such an environment.

Dynamic vs. Static keyboard designs

Keyboards with static layouts do not incur a penalty (due to scanning time) on the ITR or typing speed. Static layouts allow users to develop strategies which facilitate their learning the keyboard, thus increasing the overall communication speed.

The rapid rate of row/column intensification in the 6x6 matrix of the P300 speller presents a difficulty to users[xlix]. It has been suggested that involuntary eye movements in conjunction with the number of symbols presented, may make it difficult for a user to focus on a particular location on the matrix[l].

Research suggests that disabled individuals may not be as successful in BCI controlled tasks as able-bodied counterparts[li].

Although advances in storage have allowed for greater memory capacity on mobile devices, the number of applications available for portable devices has also increased. Thus it is still important to consider code size when writing for portable devices. One way to avoid leaving large footprints is to interpret a scripting language instead of executing compiled object code, which tend to be much larger than the source[lii].

Conclusion (incomplete)

We have presented a virtual keyboard designed for effective control in a severely noisy and bandwidth-limited environment. Our results suggest that it is possible to enable communication rates over extremely low bandwidth channels, comparable to those of existing BCI-controlled spellers. We argue that our design is better suited for noisy, bandwidth-limited communication than existing virtual keyboards.

BCI systems promise to deliver applications that will one day allow both patient and non patient populations to bypass conventional motor control pathways enabling direct neural communication with devices.

We borrow from machine learning, information theory, in order to minimize the adverse effects of a noisy channel, thus allowing communication.

Future Work (incomplete)

- letter vs owrd weights
- higher order ngrams
- error-correction
- static design
- test with BCI

i "Next Little Thing 2010 - Psychic video games (4) - Small Business."
ii "url.pdf."
iii Lotte et al., "A review of classification algorithms for eeg-based brain-computer interfaces."
iv Nunez and Srinivasan, *Electric Fields of the Brain*.
v Ibid.
vi Lotte et al., "A review of classification algorithms for eeg-based brain-computer interfaces."
vii del R. Millan and Mourino, "Asynchronous bci and local neural classifiers: an overview of the adaptive brain interface project."
viii Blankertz et al., "Advanced Human-Computer Interaction with the Berlin Brain-Computer Interface."
ix Cincotti et al., "Interacting with the Environment through Non-invasive Brain-Computer Interfaces."
x "Next Little Thing 2010 - Psychic video games (4) - Small Business."
xi Wolpaw et al., "Brain–computer interfaces for communication and control."
xii Ibid.p.9
xiii Ibid.
xiv Ibid.
xv Ibid.
xvi Ibid.
xvii Ibid.
xviii Guger et al., "Design of an EEG-based Brain-Computer Interface (BCI) from Standard Components running in Real-time under Windows-Entwurf eines EEG-basierten Brain-Computer Interfaces (BCI) mit Standardkomponenten, das unter Windows in Echtzeit arbeitet."
xix Kölsch and Turk, "Keyboards without keyboards."
xx Rakotomamonjy and Guigue, "BCI competition III."
xxi Blankertz et al., "Advanced Human-Computer Interaction with the Berlin Brain-Computer Interface."
xxii Blankertz et al., "The Berlin brain-computer interface presents the novel mental typewriter hex-o-spell."
xxiii Blankertz et al., "The Berlin brain-computer interface presents the novel mental typewriter hex-o-spell."Blankertz et al., "The Berlin Brain-Computer Interface."
xxiv Blankertz et al., "The non-invasive Berlin brain–computer interface."
xxv Blankertz et al., "Advanced Human-Computer Interaction with the Berlin Brain-Computer Interface."
xxvi "n-gram - Wikipedia, the free encyclopedia."
xxvii "ngrams.pdf."
xxviii "n-gram - Wikipedia, the free encyclopedia."
xxix Jones and Mewhort, "Case-sensitive letter and bigram frequency counts from large-scale English corpora."
xxx "ngrams.pdf."
xxxi Rawlinson, "Bigram frequency counts and anagram lists."
xxxii Jones and Mewhort, "Case-sensitive letter and bigram frequency counts from large-scale English corpora."
xxxiii Curran and Osborne, "A very very large corpus doesn't always yield reliable estimates."
xxxiv Rawlinson, "Bigram frequency counts and anagram lists."
xxxv Russell and Norvig, *Artificial Intelligence*.
xxxvi Ibid.
xxxvii Ibid.
xxxviii Brown et al., "Class-based n-gram models of natural language."
xxxix Makhoul and Schwartz, "State of the art in continuous speech recognition."
xl Ibid.
xli Cormen et al., *Introduction to Algorithms, Second Edition*.
xlii "Huffman coding - Wikipedia, the free encyclopedia."
xliii "Google Trends: Nov 13, 2009."
xliv "felton_07.pdf."
xlv Sellers and Donchin, "A P300-based brain–computer interface."
xlvi Sellers and Donchin, "A P300-based brain–computer interface."
xlvii "ousevskeyboard.pdf."
xlviii Karat, McDonald, and Anderson, "A comparison of menu selection techniques."
xlix Sellers and Donchin, "A P300-based brain–computer interface."
l Sellers and Donchin, "A P300-based brain–computer interface."
li Ibid.
lii "download.pdf."