Design of a Virtual Keyboard for a Noisy Environment and Limited Bandwidth

Tamara Blain

December 20, 2009

Abstract

This master's report addresses the problem of communication given extremely low bandwidth and a noisy channel.

Brain computer interfaces (BCI) affords individuals with severe neuro-muscular impairments, direct neural control of external devices, with devices enabling communication being among the most critical. EEG appears to be the most practical, commercially viable mode of BCI control thus far, due to its portability, and comparatively reasonable cost [1].

Control of external devices requires that the BCI system be able to differentiate between different patterns in continuous brain signals [2]. But 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, while feedback to the user can present a great deal of information.

Amplitudes of EEG scalp recorded data are quite small and difficult to distinguish from artifacts [3]. Muscle contraction, eye movement, and even the heartbeat are a few sources of artifacts which lend to a noisy environment [4]. In addition, EEG has fairly 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 sub scale of all brain activity, and the result is a very low signal to noise ratio.

The number of distinguishable patterns—*brain states*—partially hinges upon classification accuracy [5]. 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. We consider a BCI system with the additional limitations imposed by an information transfer rate of 1 bit / 3s.

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.

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 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.

The addition of redundancy to an already low-bandwidth environment, greatly reduced our rate of information. But we use a language model to predict away as much of the redundancy inherent in language as we can, so that each choice the user makes provides the maximal amount of additional information.

1 Introduction

Many of the devices designed to make communication possible for individuals with motor disabilities 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 [6], [7]. One of the primary objectives 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 that the BCI system be able to differentiate between different patterns in continuous brain signals [8]. But amplitudes of scalp recorded data are quite small and difficult to distinguish from artifacts [9]. Muscle contraction, eye movement, and even the heartbeat are a few sources of artifacts which lend to a noisy environment [10]. In addition, EEG has fairly 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 sub scale of all brain activity, and the result is a very low signal to noise ratio.

The number of distinguishable patterns—*brain states*—partially hinges upon classification accuracy [11]. BCIs using binary classifiers in general reach an accuracy of around 90% [12], 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. We consider a BCI system with the additional limitations imposed by an information transfer rate of 1 bit /3s.

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 maximal amount of information from the user, into the minimal 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, if one properly exploits the statistical properties of language, then one should be able to 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 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 presented, Circ-O-Spell, is an extension of an existing keyboard, Hex-O-Spell [13]. 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 such as to reduce the number of steps to a correct selection, and hence increase communication speed. Circ-O-Spell provided us with the exploratory ideas which led to BinSpell.

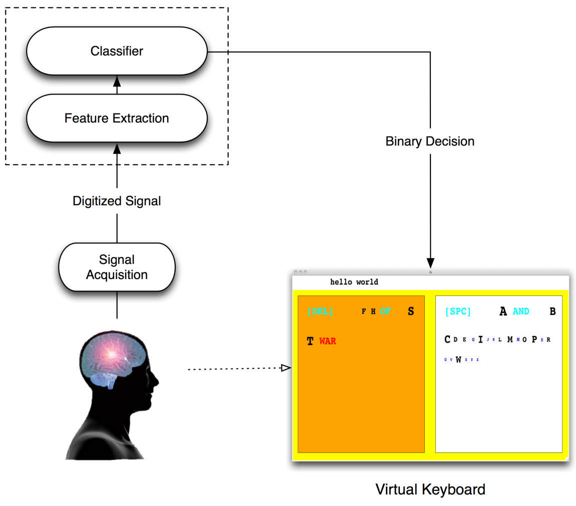


Figure 1: BCI

Thus we focus primarily on BinSpell.

2 Background

2.1 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), magnetoencephology (MEG), positron emission tomography (PET), functional magnetic resonance imaging (fMRI), and near infrared spectroscopy (NIRS) [14]. EEG appears to be the most practical, commercially viable option thus far, due to its portability, and comparatively reasonable cost [15]. The limitations of the remaining approaches are due mostly to the fact that they are technically demanding and expensive [16].

2.1.1 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 [17]. 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 [18]. EEG has inherently low spatial resolution, so the signals picked up from EEG sensors reflect the activity of millions of neurons [19]. In addition to brain activity, electrical signals from the scalp can also come from sources such as eye blinks or tongue movement [20]. Electrical impulses from the heart can actually produce scalp

Table 1: Comparison of typing speeds achieved by existing virtual keyboards.

	Hex-O-Spell	P300	P300	Dasher
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.)

potentials greater than EEG amplitudes [21], 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 [22]. All of these factors contribute to the extremely low signal to noise ratios in recorded signals.

2.2 BCI

A brain computer interface (BCI) is a communication channel between the brain and a computer (figure 1). 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 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 [23].

2.3 Virtual Keyboards

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 [24]. 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.

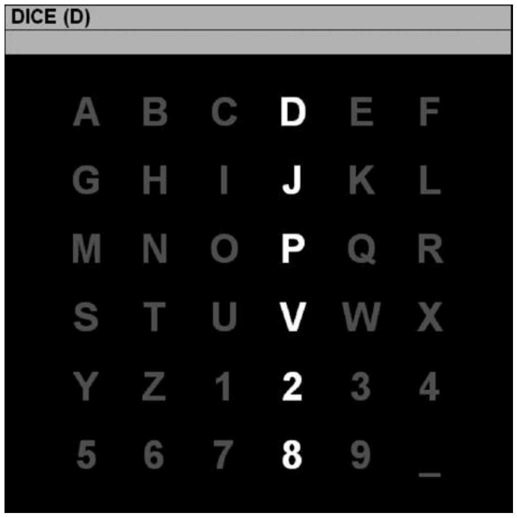


Figure 2: P300 Speller

3 Related Work

3.1 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 [25]. In this paradigm (shown in figure 2), users are presented with a 6×6 matrix of symbols, where each cell contains a symbol [1]. The user is asked to focus on their symbol of choice. The rows and columns are flashed randomly, until the appropriate response signal is detected [1]. The desired output is the intersection of the row and column which elicits the P300 signal [1].

Although it requires the use of signal averaging, as the random sequence stimuli have to be presented multiple times, research into improving P300 classification rates has resulted in reported typing rates between 2.8 and 7.8 chars/min [2]. Some classification techniques promise substantial increases in P300 typing speed, of up to 16.67 chars/min[3]. Improvements to the P300 Speller have focused mainly on improvement of classification algorithms designed to detect P300 signals [4], [5], [2].

3.2 Hex-O-Spell

Hex-O-Spell keyboard layout is comprised of 6 hexagonal fields surrounding a circle, each containing 5 characters (figure 3). The center of the circle contains an arrow used to make selections. A language model determines how the letters are arranged within each hexagon [6].

Hex-O-Spell is mentally controlled by imagined right hand and foot movements [7]. 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 [7]. Selection of a hexagon is caused by sustained right foot motor imagery, and causes all other hexagons to be cleared [7]. The 5 characters in the chosen hexagon are then redistributed among the empty hexagons, and the arrow is reset to its minimal length [7]. Parameters such as the arrow turning and growing speeds can be tailored to the user.

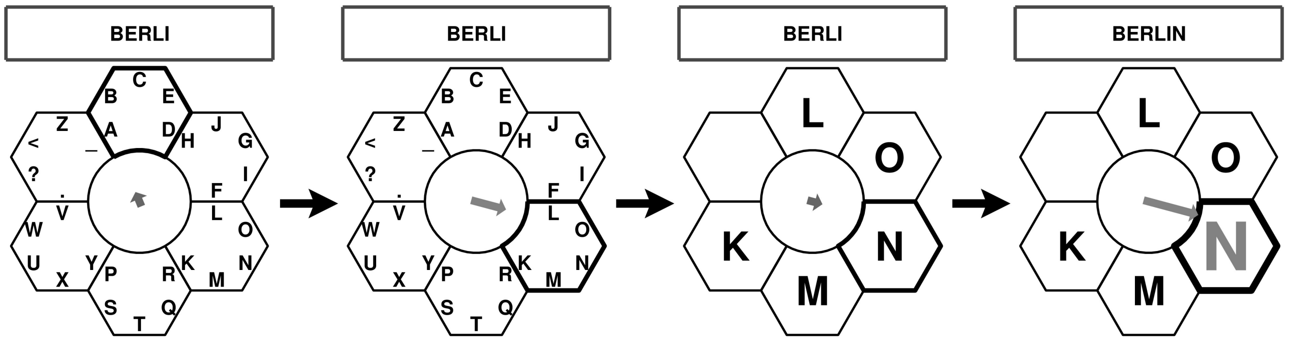


Figure 3: Hex-O-Spell

Hex-O-Spell [26], [27] is controlled by the Berlin Brain Computer Interface (BBCI). The BBCI is an EEG-based system which decodes motor imagery without subject training [28]. Much of the focus concerning the improvement of Hex-O-Spell, has been on the improvement of the front end. The relationship between control, action, and belief were considered in its design, as was the "cost" of an action [7]. The language model used by Hex, its use of machine learning techniques and its ability to operate at high decision speeds, affords high quality feedback to the user. [29] reports 98% accuracy at an average speed of 1 decision every 2.1 s.

Hex-O-Spell was tested with two subjects who had no training on the BBCI, and who had very little experience with the keyboard [7]. 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 [6].

3.3 Dasher

Dasher is a zooming interface (figure 4) which uses a language model to predict the probabilities of symbols given a context, and determine the size of the symbols based on those probabilities. So the letters with the largest probabilities given the output text will be more easily seleccted. When the predictions are correct, sequences of letters can be chosen quickly with a continous gesture.

Dasher was designed as an alternative to the standard keyboard, and designed to be used with any input methods. It uses a sextuplet-gram language model to accurately predict letters contextually [8]. The language model can be trained on any text, making it biased towards the set of words and phrases the user is most likely to use [9]. With error-free, binary input, Dasher can emit text at a rate of 1 character every 1 bits [9]. Spelling rates of up to 14.4 chars/min have been reported with BCI-controlled Dasher [10], with an average of ~ 5.6 chars/min.

4 Our Approach

Natural language has a non-uniform information density—the amount of new information communicated by each additional letter is highly dependent on previous letters and context. We use a language model to predict away as much of the redundancy as we can, so that each choice the user makes provides the maximal amount of additional information.

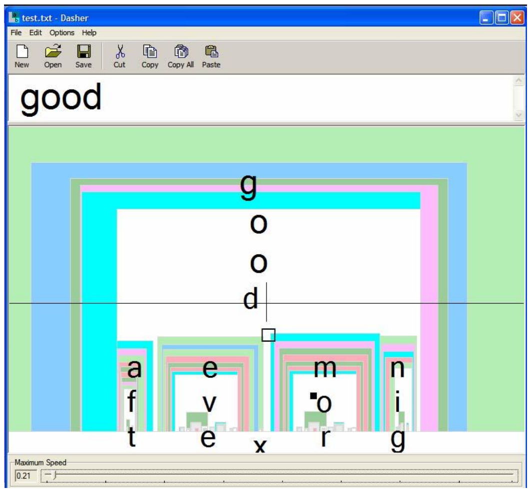


Figure 4: Dasher

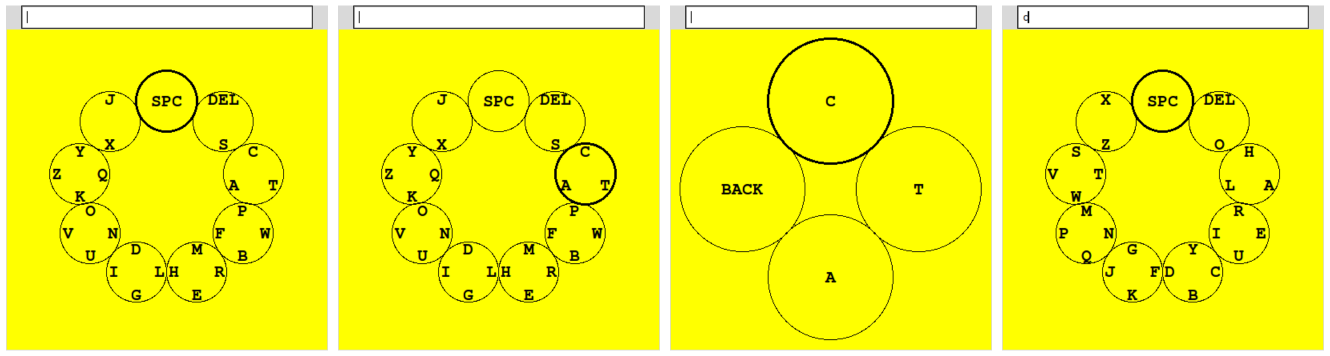
4.1 Circ-O-Spell

This keyboard is an extension of the Hex-O-Spell keyboard [31]. The hexagonal fields are replaced by circular fields, containing sets of symbols (figure 5(a)). 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 (figure 5(c)). 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 (figure 5(d)).

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 [32]. n-grams do have shortcomings in that they can only capture dependencies within their own length [33]. They cannot explicitly represent any long range dependencies and are thus unable to distinguish them from noise [34].

Higher order n-grams can provide more information, as they reflect the frequencies of larger units [35] but tend to introduce more sparsity [36]. This sparsity can be resolved, however, with large corpora, smoothing techniques, and the incorporation of contextual information [37], [38], [39]. [40] 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 [41]. The order of a Markov process defines the number of previous states on which the current state is



(a) (b) (c) (d)

Figure 5: Circ-O-Spell

Dependent. 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 [42]. First-order Markov processes are characterized by their prior probabilities, transition models, and emission models. Borrowing notation from [43], where X_t denotes a state variable at time t, prior probabilities $P(X_.)$ 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 [44], [45]. 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 notation in Table 2.

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

$$P(X_1 = x_j | X_0 = x_i) = \frac{C_1(x_i x_j)}{\sum_{i=1}^{k}\sum_{j=1}^{k} C_1(x_i x_j)} \tag{2}$$

$$P(X_t = x_j | X_{t-1} = x_i) = \frac{C_2(x_i x_j)}{\sum_{i=1}^{k}\sum j = 1^k C_2(x_i x_j)} \tag{3}$$

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 [46].

Table 2: Explanation of notation.

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

4.2 BinSpell

4.2.1 Design and Operation

BinSpell (Figure 6(a)) 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 (Figure 6(b)). 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 (Figure 6(c)).

4.2.2 Implementation Details

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 [47]. 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 [48].

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 Figure 7.

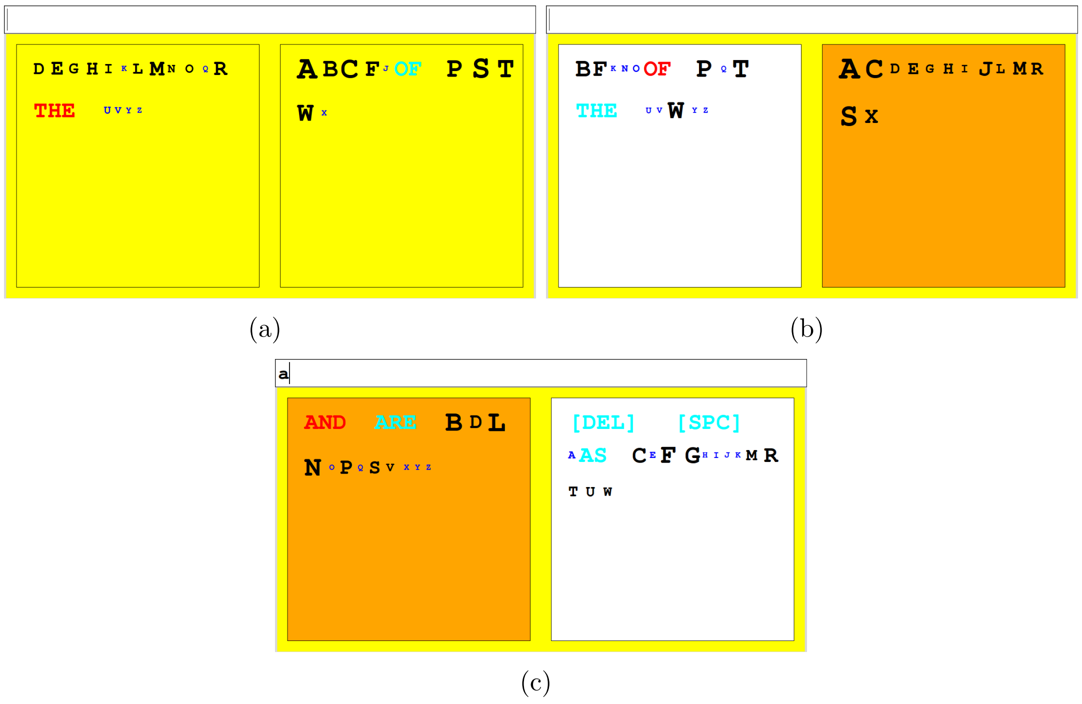


Figure 6: BinSpell

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 (Figure 6(b)). 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 (notation is in Table 2).

$$P(W_0 = w_i) = \frac{C_1^W(w_i)}{\displaystyle\sum_{j=1}^{n} C_1^W(w_j)} \tag{4}$$

$$P(W_t = w_j | W_{t-1} = w_i) = \frac{C_2^W(w_i w_j)}{\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n} C_2^W(w_i w_j)} \tag{5}$$

$$P(X_t = x_k | X_{t-1} = x_j, X_{t-2} = x_i) = \frac{C_3^X(x_i x_j x_k)}{\displaystyle\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} C_3^X(x_i x_j x_k)} \tag{6}$$

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.

5 Methods

5.1 Varying Weight, Threshold, and Error-rate

We began by attempting to determine the most effective combination of weights and thresholding levels. For this purpose, we modified BinSpell to include a routine which automates the typing out of text strings. Strings are parsed into individual word-tokens, using space as delimiter and words are parsed into character-tokens. Then character-tokens are passed sequentially, accompanied by the word-token to which they belong Both boxes of symbols are checked for the presence of the word first, followed by its accompanying character, and a bit corresponding to each box is sent to BinSpell's state update routine. Misclassification are simulated by the flipping of a bit at error-rates 0.1, and 0.2. A count is kept of both the number of bits and the number of characters correctly output. Once a phrase has been typed out completely without error, the number of bits sent is used to calculate a typing speed. The calculated typing speeds assume 3s per bit.

This process is repeated for 5 phrases, 100 times each. The phrases are chosen from among the top 10 Google search queries, for the U.S. on Nov. 13th, 200949. These were:

1. tony alamo

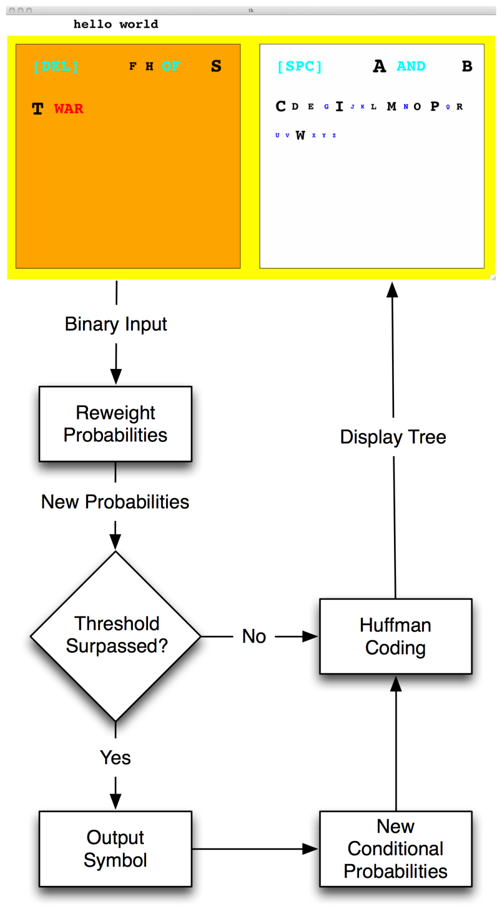
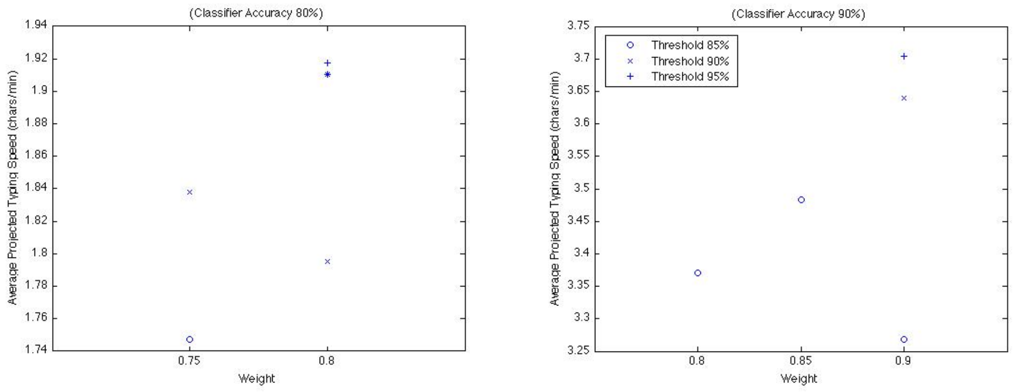


Figure 7: BinSpell FlowChart

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

Classifier accuracies of both 80% and 90% were simulated. The total number of characters typed for each accuracy simulation was 193.



(a) (b)

Figure 8: Effects of varying weight and threshold on the typing performance.

6 Results

The effects on typing speed of varying the weights and thresholds for error rates of 0.2 and 0.1, are shown in Figure 8. On the x-axis are the values by which the probabilities of the symbols in a chosen set are weighted. The best performance simulating both error-rates was obtained when the weights were set to the classifier accuracy. For both simulations, weighting by values less than the classifier accuracy resulted in reduced performance. Also in both cases, thresholding at a value equal to the weights, significantly reduced performance. This reduction is pronounced when the accuracy is 80%. The lowest typing speeds resulted when the weights were set to lower than the classifier accuracy.

Figure 9 shows the number of characters versus time for each of the error rates tested. Predictably, the lowest time for all error-rates corresponds to the shortest sentence and the times vary fairly significantly with error-rates. There is a noticeable pattern in variation among the error-rates.

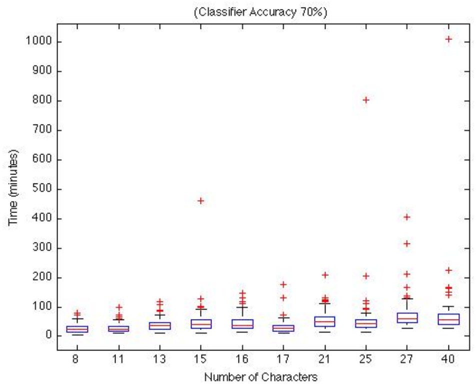
7 Discussion

7.1 Absolute Performance

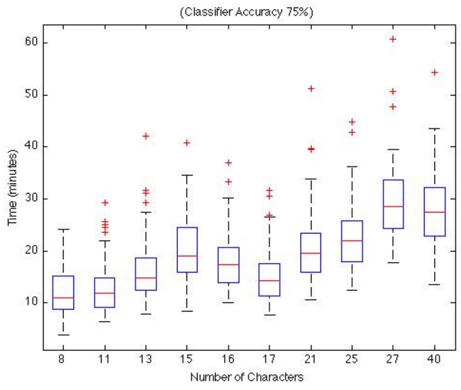
If the weighting is thought of as a "confidence", then the fact that setting the weights equal to classifier accuracy seems reasonable. Then weighting by values lower than the classifier accuracy can be thought of as having less confidence in a selection, despite a less noisy environment. This means the probabilities grow more "cautiously" than is necessary. This results in an increase in the number of decisions necessary to access the symbol of choice, incurring a penalty on typing speed. An increase in the number of steps to a given symbol may also mean more opportunity for error to occur, also reducing the typing speed.

Thresholding is intended to minimize the number of incorrect symbols typed due to classifier error. It is possible that setting the thresholding value equal to the classifier error reduces the amount of redundancy needed to correct the error, resulting in lower performance.

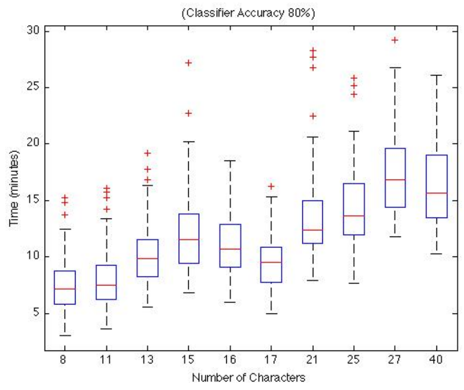
Redundancy is a tool used in information theory for the detection and correction of noise. The redundancy introduced by the re-ordering of the symbols is intended minimize the effect of



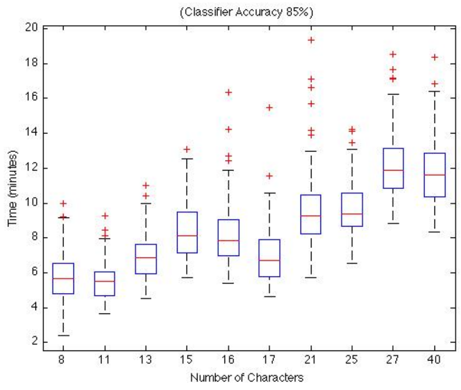
(a)



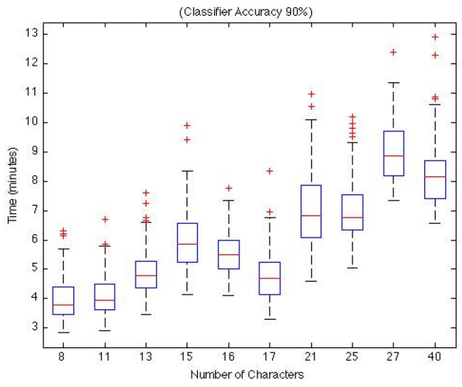
(b)



(c)



(d)



(e)

Figure 9: Time to type each phrase for varying error rates.

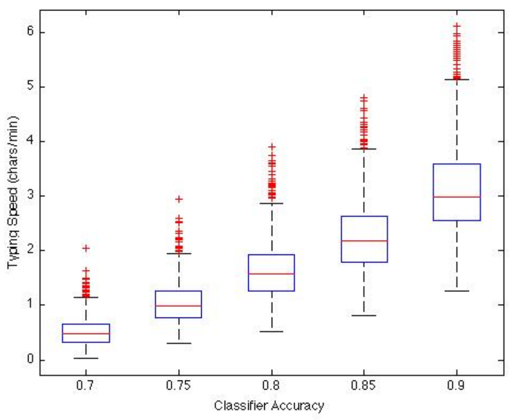


Figure 10: Comparison of typing speeds for varying error rates.

classifier error on the ability to communicate using BCI, but at a sacrifice in speed. A language model is employed to help counter this trade off, thus enabling communication.

The pattern in the variation of the times to type each phrase, can be most likely explained by the presence of a dictionary. It is likely that each reduction in time corresponds to a phrase which contains a significant number of words found in BinSpell's dictionary. Thus there were more opportunities to suggest the correct word completion, and the number of decisions made to complete the phrase was reduced. All of the words in the 40-character phrase were in the dictionary. The pattern may also reflect a need for higher order n-grams, in order to better discriminate between words.

7.2 Relative Performance

7.2.1 BinSpell vs. P300

One of the aims of research into BCI typing designs is to reduce the cognitive load on the user. [50] 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 6×6 matrix of the P300 speller presents a difficulty to users [51]. 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 [52].

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.

7.2.2 Circ-O-Spell vs. Hex-O-Spell

Circ-O-Spell's primary advantage over Hex-O-Spell is its flexible layout. The number of symbols per set is not fixed. Thus, in the best case scenario, it may only take one bit to select a letter, if a single letter is in a set. The best case scenario for Hex-O-Spell is 2: one for selection of the set, followed by selection of the character. In addition, the dynamic layout of Circ-O- Spell, given a robust language model, would facilitate access to the most frequently used characters, given the typed context. Thus, on average, fewer bits would be required for communication.

7.2.3 BinSpell vs. Hex-O-Spell and Dasher

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 [53], [54]. 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.

Hex-O-Spell requires a user be able to reliably maintain a state in order to make a selection [7]. But factors such as fatigue and attention level, can cause significant BCI performance drops [10], [11], [12]. Because Hex-O-Spell is designed for binary input, the selection element can only rotate in one direction. Thus overshooting targets can introduce significant penalties in time, due to the lengthy rotation is necessary to return to the desired location [7]. BCI control of BinSpell only requires that the user be able to switch states.

It is difficult to control a cursor with a single bit. Effective cursor control via BCI often requires several hours to weeks of training [13], [14], [10]. Dasher was designed for continuous relatively noise-free input. Given 0.33 bits/s, and a high error-rate, BinSpell was designed to enable meaningful communication with extremely limited bandwidth. Given same bandwidth limitation and noisy environment, Dasher's cursor would become unmanageable.

7.3 Additional Advantages

7.3.1 P300

The P300 paradigm places limits on the user in several ways. The P300's reliance on endogenous signals makes it a strictly synchronous system. The user is forced to proceed at the pace of the stimuli, and does not have the option of varying the pace at which they proceed. It relies on a single signal type, which may not be the best suited for the user [15]. Also, P300's fixed grid approach means it is not easily modified to include alternate selection options.

In contrast, BinSpell can be extended to work with an asynchronous BCI and any type of BCI approach. Its flexible, open design allows for the addition of words, phrases, and pictures.

The rapid rate of row/column intensification in the 6×6 matrix of the P300 speller presents a difficulty to users [55]. 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 [56]. BinSpell is dynamic keyboard with a moderate presentation rate.

7.3.2 Hex-O-Spell

Parameters relating to the selection element employed by Hex-O-Spell, have to be tuned to the user for optimal performance. Thus a new user is forced to rely on an outside expert or a trained user in order use Hex-O-Spell optimally. In contrast, BinSpell requires no extra tuning, which makes it better suited for the general population.

Lastly, due to it's fixed number of sets, and fixed number of options per set, Hex-O-Spell provides little flexibility for the addition of options such as words and phrases, while the open plan of BinSpell was designed with this adaptability in mind.

7.3.3 Dasher

Although Dasher offers the highest typing rates, there is a steep learning curve associated with it's control, even by conventional input methods [10]. Additionally, BCI control of continuous targets often require significant training time, and often impose significant cognitive load on the user [10]. Training for BinSpell, in comparison, consists of a short explanation on it's manipulation. And BinSpell's lack of dependence on a continuous selection element, reduces the BCI training time as well.

8 Conclusion

We have presented a virtual keyboard designed for effective control in a severely noisy and bandwidth-limited environment. We borrow from machine learning and information theory, in order to minimize the adverse effects of the low signal to noise ratio inherent in EEG signals, and the limitations of an inaccurate classifier, thus allowing communication. We argue that our design is better suited for noisy, bandwidth-limited communication than existing virtual keyboards. Our results suggest that it is possible to enable communication rates over extremely low bandwidth channels, comparable to those of existing BCI-controlled spellers.

9 Improving BinSpell and Future Work

An ideal BCI keyboard with given our limitation of 0.33 bits/choice would provide 0.33 bit of additional information each choice, with respect to the language model. A higher order, adaptive language model would allow BinSpell to use bits more efficiently, while adding to a speed up in typing rates. We intend to further investigate the question of how best to weigh letter-level and word-level n- grams, with respect to one another.

Keyboards with static layouts do not incur a penalty (due to scanning time) on the typing speed [16]. Static layouts allow users to develop strategies which facilitate their learning the keyboard, thus increasing the overall communication speed [16]. We have devised a way to keep our policy of reordering symbols, while allowing for a static interface. Using this approach, both sets of symbols would be drawn in exactly the same locations in both boxes. The shuffling would consist of making them visible or not, depending on what side of the Huffman tree they were found. This minimally dynamic approach would reduce the amount of searching done by the user, and allow them to devise strategies associated with static interfaces.

We also intend to test the performance of BinSpell with a BCI.