An SSVEP-Based Brain–Computer Interface for Text Spelling With Adaptive Queries That Maximize Information Gain Rates

Abdullah Akce, James J. S. Norton, and Timothy Bretl, *Member, IEEE*

***Abstract*—This paper presents a brain-computer interface for text entry using steady-state visually evoked potentials (SSVEP). Like other SSVEP-based spellers, ours identifies the desired input character by posing questions (or queries) to users through a visual interface. Each query defines a mapping from possible characters to steady-state stimuli. The user responds by attending to one of these stimuli. Unlike other SSVEP-based spellers, ours chooses from a much larger pool of possible queries—on the order of ten thousand instead of ten. The larger query pool allows our speller to adapt more effectively to the inherent structure of what is being typed and to the input performance of the user, both of which make certain queries provide more information than others. In particular, our speller chooses queries from this pool that maximize the amount of information to be received per unit of time, a measure of mutual information that we call information gain rate. To validate our interface, we compared it with two other state-of-the-art SSVEP-based spellers, which were re-implemented to use the same input mechanism. Results showed that our interface, with the larger query pool, allowed users to spell multiple-word texts nearly twice as fast as they could with the compared spellers.**

***Index Terms*—Assistive technology, brain–computer interfaces, brain modeling, electroencephalography, user interfaces.**

I. Introduction

THE DEVELOPMENT of electroencephalogram (EEG)-based brain–computer interfaces (BCI) for text entry has exploded over the past decade [1]. These interfaces create a direct neural link between a human user and a computer, allowing the user to type without a keyboard or physical movements. One common input mechanism, which we consider in this paper, is the steady-state visually evoked potential (SSVEP). In an SSVEP-based speller, users are presented with a set of visual targets that are associated with possible characters. These targets blink on and off at slightly different but fixed frequencies. By attending to a particular target, the user elicits phase-locked EEG activity at the corresponding frequency. Measurement of this activity allows the computer to detect the target to which the user is attending, hence the user's desired character. While it is not within our scope to discuss the relative merits of BCI and non-BCI text entry, we note that SSVEP may remain applicable even when users have no control over gaze (e.g., as with "locked-in" syndrome). Attentional focus that is independent of visual focus, also elicits SSVEPs [2].

Most existing SSVEP-based spellers have fewer visual targets than possible characters. As a consequence, the user must attend to a sequence of targets in order to type a single character. For example, the SSVEP-based spellers of both Volosyak [3] and Cecotti [4] allow 28 possible characters—the standard alphabet, space, and delete—but have only five targets. The interface of Volosyak [3] arranges characters in a grid, associates four targets with cardinal directions (left, right, up, down) in which to move a cursor in this grid, and interprets the fifth target as selecting the character at the current location of the cursor. The interface of Cecotti [4] arranges all characters except delete in a static hierarchical menu with a decision between three groups at each level of the hierarchy (three groups of nine characters, then of three characters, and finally of one character), associates a target with each group, associates the fourth target with delete, and interprets the fifth target as moving up in the hierarchy.

When discussing these two interfaces, we find it helpful to regard each presentation of visual targets as a question or *query* posed to the human user. Each query defines a mapping from possible characters to visual targets, in the sense that there is a correct choice of target to which the user must attend in order to specify a given character most quickly. The queries of Volosyak [3] ask which direction the desired character is with respect to the cursor. The queries of Cecotti [4] ask which of three groups contains the desired character.

In this paper, we present a new SSVEP-based speller that is similar to the ones of Volosyak [3] and Cecotti [4] but that poses a different set of queries. These queries are of two types. A *range query* [Fig. 1(a)] asks which of five ordered groups of characters (e.g., "delete" through B, C through K, L through M, N through T, and U through "space") contains the desired character, by associating a target with each group. A *character query* [Fig. 1(b)] asks which of four ordered characters (e.g., C, F, G, and S)—if any—is the desired character, by associating a target with each one and by interpreting the fifth target as "none of them."

What is important about these two new types of queries is their variety. There are 2925 distinct range queries and 20475 distinct character queries, meaning that our interface has a total of 23400 queries from which to choose. You might say that our

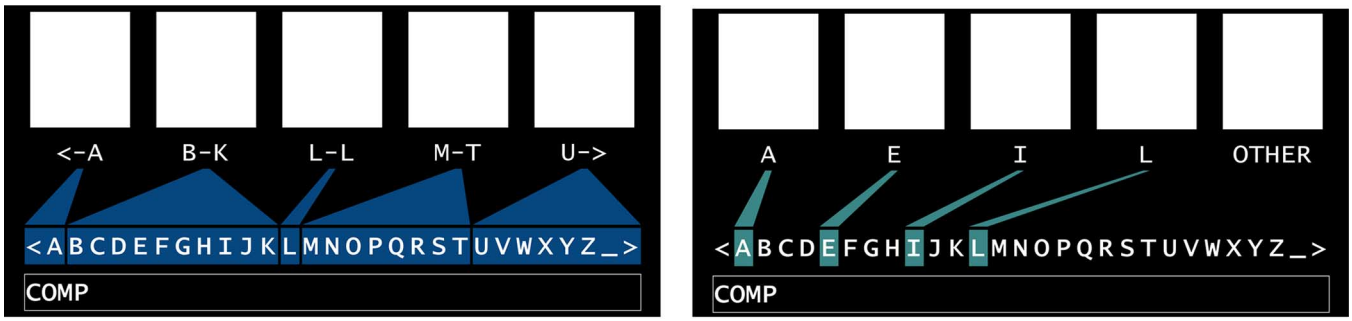
Manuscript received April 15, 2014; revised August 20, 2014; accepted September 21, 2014. Date of publication December 02, 2014; date of current version September 03, 2015. This work was supported by the National Science Foundation (NSF) under Grant 0955088 and Grant 0903622.

A. Akce was with Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. His is now with Google Inc., Mountain View, CA 94043 USA (e-mail: abdullah.akce@gmail.com)

J. Norton is with the Neuroscience Program, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: jnorton4@illinois.edu).

T. Bretl is with the Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: tbretl@illinois.edu).

Digital Object Identifier 10.1109/TNSRE.2014.2373338



(a) (b)

Fig. 1. Examples of the two types of queries posed by our speller: (a) range queries and (b) character queries. In each case, targets are arranged horizontally at the top of the screen (the white boxes) and text appears along the bottom. (a) Range query posed after spelling "COMP." (b) Character query posed after spelling "COMP."

interface has a *query pool* of size 23400. In contrast, the interface of Volosyak [3] has a much smaller query pool of size 28, equal to the number of characters in the grid. Similarly, the interface of Cecotti [4] has a query pool only of size 13, equal to the number of possible groupings of characters in the static hierarchical menu.

The reason that variety is important is that not all queries are equally informative. Language has an inherent structure that—depending on context—makes certain characters much more likely than others. In principle, larger query pools allow better adaptation to this structure. They give an interface the freedom to pick "the right question."

Indeed, our SSVEP-based speller chooses queries that explicitly maximize the amount of information to be received per unit of time about the desired character, a measure of mutual information that we call information gain rate (IGR). IGR is similar to other measures of mutual information like the commonly used information transfer rate (ITR) [5]–[7] and the less well known Nykopp bit rate [8]. The reason we use a new measure in this paper is that IGR—unlike ITR, for example—does not assume that each visual target is equally likely to be selected within a given time window. Instead, IGR takes into account the expected input performance of each user (characterized by selection accuracy and selection latency) as well as a probabilistic language model. We emphasize that IGR and the models that it takes into account are not in direct competition with the query pool. Rather, if the query pool gives us the freedom to pick the right query, IGR offers our interface the means to select that query. This choice of measure distinguishes our interface from others that also have larger query pools, like the motor-imagery-based speller of Blankertz *et al.* [9] and the P300-based speller of Ma *et al.* [10]—although, these other interfaces are harder to compare directly due to their use of input mechanisms other than SSVEP.

We acknowledge that many factors affect the performance of an SSVEP-based speller (e.g., the classification rate [11], [12], the stimulus design [13]–[15], and the layout of the interface [16], [17]). Larger query pools may also necessitate the use of serial visual search strategies by the participant, resulting in prolonged selection latencies. Nonetheless, empirical results with six qualified subjects showed that our SSVEP-based speller—with the larger query pool and with IGR as the performance measure to be maximized—allowed users to spell multiple-word texts nearly twice as fast as they would with the SSVEP-based spellers of Volosyak [3] and Cecotti [4].

In what follows, we first present our new speller (Section II). Next, we describe the experimental comparison to existing spellers (Section III) and give the results of this comparison (Section IV). Finally, we conclude by discussing the importance of these results with respect to the design of BCI text-entry systems (Section V).

II. Design of Our Speller

Our SSVEP-based speller allows a user to type a string of text that consists of the standard alphabet ("A" to "Z"), space ("_"), and delete ("<"). In order to type each individual character, the user must respond to a sequence of queries. Each query associates possible characters with one of five visual targets (i.e., five blinking SSVEP stimuli). Our speller poses two types of queries (Fig. 1).

1) A *range query* asks which of five ordered groups of characters contains the desired character, by associating a target with each group. In the example of Fig. 1(a), the five groups are "<" to "A", "B" to "K", "L" (i.e., a group with only one character), "M" to "T", and "U" to "_".
2) A *character query* asks which of four ordered characters—if any—is the desired character, by associating a target with each one and by interpreting the fifth target as "none of them." In the example of Fig. 1(b), the four characters are "A", "E", "I", and "L".

Each query reduces our speller's uncertainty about the user's desired character. Once the speller is confident enough, it makes a "guess" at the desired character, appends this guess to the string of text at the bottom of the screen, and proceeds with a new sequence of queries to obtain the next character.

In Section II-A, we define a formal model of our speller. In Section II-B, we use this model to derive algorithms that say how to choose each query (based on maximizing a measure of mutual information that we call IGR) and how to guess the desired character (based on maximizing likelihood with respect to a language model).

A. Models

In what follows, we denote the set of possible characters by $\mathcal{C} = \{\text{`}<\text{'}, \text{`A'}, \ldots, \text{`Z'}, \text{`\_'}\}$ and the set of possible targets, numbered from left to right, by $\mathcal{U} = \{1, \ldots, 5\}$.

1) Queries: We can describe any range or character query as a map $f:\mathcal{C} \to \mathcal{U}$. For example, the range query in Fig. 1(a) is given by

$$f(c) = \begin{cases} 1, & \text{if } c \in \{\text{`} < \text{'}, \text{`A'}\} \\ 2, & \text{if } c \in \{\text{`B'}, \ldots, \text{`K'}\} \\ 3, & \text{if } c \in \{\text{`L'}\} \\ 4, & \text{if } c \in \{\text{`M'}, \ldots, \text{`T'}\} \\ 5, & \text{otherwise.} \end{cases} \tag{1}$$

Suppose that the user in this example were trying to spell the word "COMPILE." Having already spelled "COMP," the current desired character would be `I'. The formal definition (1) of the range query in Fig. 1(a) makes clear that the correct choice of target would be $f(\text{`I'}) = 2$—i.e., that the user should attend to the second target from the left in order to specify their desired character most quickly. Similarly, the character query in Fig. 1(b) is given by

$$f(c) = \begin{cases} 1, & \text{if } c = \text{`A'} \\ 2, & \text{if } c = \text{`E'} \\ 3, & \text{if } c = \text{`I'} \\ 4, & \text{if } c = \text{`L'} \\ 5, & \text{otherwise.} \end{cases} \tag{2}$$

Continuing with our example, the correct choice of target in this case would be $f(\text{`I'}) = 3$. As can easily be derived, we have 2925 distinct range queries and 20475 distinct character queries from which to choose, for a total of $n = 23400$. We index the corresponding maps by $f_1, \ldots, f_n$.

2) Accuracy and Latency: As we have seen, there is a single correct choice of target—call it the *intended target*—to which the user should attend in response to a query. Because of uncertainty in the measurement and interpretation of SSVEP, the target that is actually selected—call it the *observed target*—may differ from the intended target. To capture this difference, we model the intended target as a discrete random variable X and the observed target as a discrete random variable Y, both taking values in $\mathcal{U}$. We also model the amount of time taken for the user to respond to a query as a continuous random variable T, taking values in the set of positive real numbers $\mathbb{R}^+$. Two statistical quantities then suffice to describe the input performance of a user:

- the conditional probability mass function $p_{Y|X}(y|x)$, which specifies the likelihood that the observed target is y given that the intended target is x;
- the conditional expectation $\mathbb{E}(T|X = x, Y = y)$, which specifies the average time taken for the user to respond to a query given that the intended target is x and the observed target is y.

We will refer to $p_{Y|X}(y|x)$ as the *accuracy model* and to $\mathbb{E}(T|X = x, Y = y)$ as the *latency model*. Although these two models may differ from one user to another, we assume that they remain the same over time—in other words, that input performance is the same when typing the first character in a string of text as when typing the last character. Accuracy and latency models can be computed from experimental data (e.g., during user training)—we will say how in Section III-C. For now, we assume both models are given.

3) Language: We model the desired character as a discrete random variable C, taking values in $\mathcal{C}$. The probability mass function $p_C(c)$ then completely describes the speller's uncertainty about the user's desired character. We will refer to $p_C(c)$ as the *language model*. There are standard ways to derive this language model from a database of English text—we will say how in Section III-D. For now, we assume that the language model is given.

Unlike the accuracy and latency models, the language model changes over time—indeed, the purpose of each query is to steer this change in a way that reduces uncertainty about the desired character. In particular, by application of Bayes' theorem (see Appendix A), it is possible to show that

$$p_{C|Y}(c|y) = \frac{p_{Y|X}(y|f_i(c))p_C(c)}{\sum_{s\in\mathcal{C}} p_{Y|X}(y|f_i(s))p_C(s)} \tag{3}$$

for all $c \in \mathcal{C}$. Equation (3) provides a recursive update rule: start with the current language model p_C, observe a target y in response to a query f_i, compute a new language model $p_{C|Y}$ using (3), and replace p_C with $p_{C|Y}$.

B. Algorithms

1) How to Choose Each Query: We can use the models defined in Section II-A to measure the amount of information to be received per unit of time about the desired character. We call this quantity IGR and define it as follows:

$$\text{IGR} = \frac{I(X;Y)}{\mathbb{E}(T)}. \tag{4}$$

The numerator in (4) is the mutual information between the intended target and the observed target. It is defined as

$$I(X;Y) = \sum_{x\in\mathcal{U}} \sum_{y\in\mathcal{U}} p_{X,Y}(x,y) \log\left(\frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)}\right) \tag{5}$$

and is a commonly used measure of information gain [18]. The denominator in (4) is the average time taken for the user to respond to a query. It is defined as

$$\mathbb{E}(T) = \sum_{x\in\mathcal{U}} \sum_{y\in\mathcal{U}} p_{X,Y}(x,y)\mathbb{E}(T|X = x, Y = y). \tag{6}$$

It is possible to show (see Appendix B) that

$$p_{X,Y}(x,y) = p_{Y|X}(y|x)p_X(x) \tag{7}$$

$$p_Y(y) = \sum_{x\in\mathcal{U}} p_{Y|X}(y|x)p_X(x) \tag{8}$$

and

$$p_X(x) = \sum_{\{c\in\mathcal{C}:f_i(c)=x\}} p_C(c) \tag{9}$$

so IGR (4) can be computed with knowledge of the accuracy model $p_{Y|X}(y|x)$, the latency model $\mathbb{E}(T|X = x, Y = y)$, the language model $p_C(c)$, and the query $f_i(c)$, all of which we defined in the previous section. Note that IGR is an explicit function of the query index i—we can make this dependence

clear by writing $\mathrm{IGR}(i)$. Our speller chooses the query with index that maximizes IGR

$$i_{\max} = \underset{i\in\{1,\ldots,n\}}{\arg\max}\ \mathrm{IGR}(i).$$

2) How and When to Guess the Desired Character: Our speller's best guess at the desired character given an observed target y in response to a query f_i is the character of maximum likelihood with respect to the updated language model

$$c_{\max} = \underset{c\in\mathcal{C}}{\arg\max} p_{C|Y}(c|y)$$

where $p_{C|Y}(c|y)$ is computed as in (3). The only remaining question is if our speller is confident enough about this guess to append $c_{\max}$ to the string of text, or if it should continue posing queries. To answer this question, our speller compares the maximum likelihood

$$p_{\max} = p_{C|Y}(c_{\max}|y)$$

to the likelihood of certain other possible characters. In particular, we define the set

$$\mathcal{C}_{\mathrm{inner}} = \{c \in \mathcal{C}\colon f(c) = y \text{ and } c \neq c_{\max}\}$$

of characters other than $c_{\max}$ that are associated with the same observed target y, and the set

$$\mathcal{C}_{\mathrm{outer}} = \{c \in \mathcal{C}\colon f(c) \neq y\}$$

of characters not associated with this target. Next, we compute

$$p_{\mathrm{inner}} = \begin{cases} 0, & \text{if } \mathcal{C}_{\mathrm{inner}} = \emptyset \\ \max\{p_{C|Y}(c|y)\colon c \in \mathcal{C}_{\mathrm{inner}}\}, & \text{otherwise} \end{cases}$$

and

$$p_{\mathrm{outer}} = \begin{cases} 0, & \text{if } \mathcal{C}_{\mathrm{outer}} = \emptyset \\ \max\{p_{C|Y}(c|y)\colon c \in \mathcal{C}_{\mathrm{outer}}\}, & \text{otherwise} \end{cases}$$

i.e., the maximum likelihood over characters in $\mathcal{C}_{\mathrm{inner}}$ and $\mathcal{C}_{\mathrm{outer}}$, respectively (or zero if either set is empty). Our speller stops posing queries when both

$$\frac{p_{\max}}{p_{\mathrm{inner}}} > \alpha \qquad \text{and} \qquad \frac{p_{\max}}{p_{\mathrm{outer}}} > \beta$$

where the thresholds α and β are parameters.

III. Methods

A. Participants

We performed experiments with 11 able-bodied participants between the ages of 20 and 30 who had normal or corrected-to-normal vision. All experiments were approved by the Institutional Review Board of the University of Illinois.

B. Signal Recording and Classification

The steady-state stimuli were five targets presented on an LCD monitor, ordered from left to right, at 7.50, 10.0, 6.67, 12.0, and 8.57 Hz. These stimuli appear as white squares across the top of Fig. 1. EEG signals were extracted from seven electrode sites across the occipital region of the scalp (PO7, PO3, PO4, PO8, O1, OZ, O2) at impedances not exceeding 10 kΩ. All electrodes were referenced to electrode location PZA [19]. EEG signals were acquired using a 128-channel bioamplifier at 256 Hz, bandpass-filtered from 1 to 30 Hz, and analyzed using a 1.5 s sliding window with an overlap of 1.375 s. A classifier, based on the traditional power spectral density analysis (PSDA) method [20], was used to determine user selections. In our implementation of this classifier, multi-electrode EEG data were filtered into four different spatial representations using bipolar and Laplacian combinations (see [21] for a description). The specific combinations, taken from Prueckl [22], were as follows:

$$\begin{aligned} CH_1 &= 4 * OZ - (O1 + O2 + PO7 + PO8) \\ CH_2 &= 2 * OZ - (O1 + O2) \\ CH_3 &= 4 * OZ - (O1 + O2 + PO3 + PO4) \\ CH_4 &= 2 * OZ - (PO7 + PO8). \end{aligned}$$

The fast Fourier transform (FFT) with a rectangular window and zero-padded to 1024 points, was computed for each combination using MATLAB's "fft" function. The result was then multiplied by its complex conjugate to obtain power spectra. A signal-to-noise ratio (SNR) was obtained for each combination and each frequency by dividing the power of the signal (average power at the frequency of interest ± 0.2 Hz) by the average power of the noise (average power in the frequency band of 6.25–12.5 Hz excluding the frequency of interest ± 0.2 Hz). The highest and lowest SNR values for each frequency were discarded. The two remaining SNR values were averaged to obtain a single value for each frequency. If any of these five averaged values exceeded a pre-determined threshold, the corresponding target was selected as the observed target. If more than one frequency exceeded the threshold during the same window of time, the lowest frequency was selected as the observed target.

C. Training Phase

The accuracy model and the latency model (Section II-A2) for each user were derived from data collected during a training phase. In this training phase, 100 queries were presented to each user. An arrow specified the intended target for each query. Users were asked to respond by attending to that target. In total, each of the five targets was specified as the intended target 20 times, in random order. For each query, the intended target, the observed target, and the user response time were recorded. The conditional probability $p_{Y|X}(y|x)$ was computed as the ratio of the number of times y was the observed target given that x was the intended target over the number of times x was the intended target. If y was never observed, in other words if the empirical value of $p_{Y|X}(y|x) = 0$, a small number (0.01) was added. These values were then normalized to obtain a probability measure. The conditional expectation $\mathbb{E}(T|X = x, Y = y)$ was computed as the average user response time over all queries in which the intended target was x and the observed target was y. If y was never observed, in other words if the empirical value of $\mathbb{E}(T|X = x, Y = y) = 0$, then this empirical value was replaced with the average user response time over all queries (over all intended and observed targets).

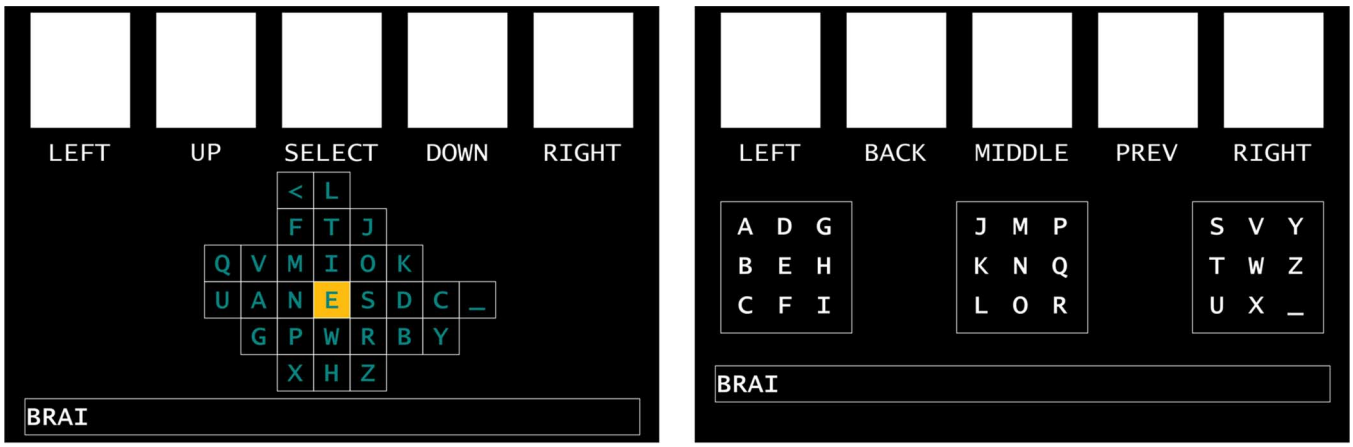


Fig. 2. Sample screenshots of the spellers re-implemented for comparison to our speller: speller of Volosyak [shown in (a)] and speller of Cecotti [shown in (b)].

D. Spelling Phase

Following training, participants completed a three-part spelling phase. The purpose of this spelling phase was to evaluate the performance of participants using our speller with their performance using two existing SSVEP spellers, the one of Volosyak [3] and the one of Cecotti [4], that were highlighted in a review of BCI text-entry [1]. All three spellers presented the same number of visual targets (five) and allowed users to type a string of text consisting of the same standard alphabet ("A" to "Z"), space ("_"), and delete ("<"). These spellers were implemented as follows.

- Our interface (Fig. 1) was implemented exactly as described in Section II. The accuracy and latency models were derived from data collected during training as described in Section III-C. The language model was constructed with prediction by partial matching (PPM) [23] applied to the English corpus provided with the Dasher text-entry interface [24]. The conditional probability of the delete ("<") character, which was not included in the PPM model, was fixed at 0.05.
- The interface of Volosyak [3] arranges characters in a grid according to their frequency in English text, associates four targets with cardinal directions (left, right, up, down) in which to move a cursor in this grid, and interprets the fifth target as selecting the character at the current location of the cursor [Fig. 2(a)]. As an example, a user might try to select "B" with the following sequence of intended targets: *right*, *right*, *down*, and then *select*. There were differences between our implementation and the original implementation of [3], both in the location of targets on the screen and in the arrangement of characters in the grid. These differences are potential sources of error and will be discussed further in Section V.
- The interface of Cecotti [4] arranges all characters except delete ("<") in a static hierarchical menu with a decision between three groups at each level of the hierarchy (three groups of nine characters, then of three characters, and finally of one character), associates a target with each group, associates the fourth target with delete ("<"), and interprets the fifth target as moving up in the hierarchy [Fig. 2(b)]. For example, a user might try to select "B" with the following sequence of intended targets: *left*, *left*, and *middle*. Our implementation and the original implementation of [4] differ only with respect to the location of targets on the screen.

TABLE I
TARGET TEXTS AND THEIR NLL

Text	Text	NLL (bits/char)
Txt1	BCI	7.19
Txt2	BRAIN	3.04
Txt3	SIREN	5.31
Txt4	BRAIN_COMPUTER_INTERFACE	1.98
Txt5	PLEASE_GET_ME_A_BLANKET	2.64

During the evaluation of each speller, subjects were asked to specify the texts in Table I. The first three (Txt1, Txt2, Txt3) were single-words texts and the last two (Txt4 and Txt5) were multiple-word texts. Table I also lists the likelihood of each text with respect to our language model as measured by the average number of bits necessary per character, a quantity that is called the negative log-likelihood per character (NLL). The multiple-word texts (Txt4, Txt5) had NLL comparable to the average NLL of English texts, which is about 2 bits per character [25], [26]. Participants completed all five texts, in order, for a single speller before moving on to the next speller. There was a short (one minute) break between each speller. To reduce possible bias, the order in which the spellers were evaluated was randomized. Experimentation was halted at the request of the user or if user performance was lower than one character per minute (cpm).

E. Simulation Study

As we acknowledged in Section III-D, there were small differences between our implementation and the original implementation of the spellers of Volosyak [3] and Cecotti [4]. In our implementation of the speller of Volosyak, the fifth row of the character grid was shifted one cell to the right of the original implementation. To quantify the effect of this layout change we

performed Monte-Carlo simulations using two versions of the speller of Volosyak: the one utilized in the present study, and the one in [3] with the original layout. A total of 1000 simulations were conducted with each text (listed in Table I) and participant (S1-S6, S8, S9). In each query, recall that there is a single correct choice of intended target, the simulation assumes that the human subject always makes the correct choice of intended target. The observed targets and the target selection latencies were randomly sampled according to the accuracy and latency models of each participant. In particular, given an intended target x the observed target y was sampled according to the probability distribution $p_{Y|X}(y|x)$. The selection latency was sampled from a normal distribution with mean $\mathbb{E}(T|X = x, Y = y)$ and with standard deviation computed from the participant's training trials. Results were obtained for our speller, the two versions of the speller of Volosyak, and the speller of Cecotti.

IV. Experimental Results

Of the 11 subjects who participated in our study, six (S1, S2, S3, S4, S5, S6) were able to complete the entire experiment. Three of the participants (S7, S8, S9) completed the training phase, but were unable to complete all three parts of the spelling phase. The remaining two participants were unable to complete the training phase with at least 50% accuracy on all of the targets. Their data have been excluded from further analysis. All subjects were naïve to EEG-based BCIs with the exception of subjects S1 and S2, who had extensive experience. Specifically, subjects S1 and S2 had previously participated in greater than 20 h of experiments with SSVEP-based BCIs. The following performance measures were used.

- *Input error* (ϵ), the fraction of the number of incorrect queries—in which the observed target did not match the intended target—to the number of all queries.
- *Input latency* $(\mathbb{E}(T))$, the mean latency—the time it takes to obtain a user response after the onset of a stimulus—across all queries.
- *Input/character ratio* (C), the average number of user responses required to spell a single character of the target text, i.e., the ratio of the total number of queries to the number of characters in the target text.
- *Spelling rate* (R), the average number of characters spelled per minute (cpm), without counting any instances of delete, i.e., the ratio of the number of characters in the target text to the total time elapsed in spelling.

We note that it was indeed possible to compute the input error during the spelling phase, since—under the assumption that errors are derived from incorrect classification of SSVEP response and not from incorrect user behavior—the intended target (either a singleton or a finite set) was always known.

A. Training Phase

Table II shows the training data obtained from each participant. Across the six subjects who completed the entire experiment, the input error was 2%, and the input latency was about 3 s. For subjects S8 and S9, who only completed the training phase, error was higher, 13% for S8 and 20% for S9. Average input latency for subjects S7 and S8 was comparable to those who completed the entire study. Input latency for S9, however, was considerably higher at 6.97 s.

TABLE II
Training Phase. (a) Average Input Accuracy (%) of Each Subject for Each Target, (b) Average Input Latency (Seconds) of Each Subject for Each Target

Subject	Target 1	Target 2	Target 3	Target 4	Target 5	Avg
S1	100	100	100	100	100	100
S2	100	95	100	100	100	99
S3	100	100	80	100	100	96
S4	100	100	100	100	95	99
S5	100	100	100	100	95	99
S6	100	95	100	95	100	98
Avg	100	98	96	99	98	98.5
S7	100	100	95	100	100	99
S8	90	95	100	55	95	87
S9	75	75	80	95	75	80
Avg	88.3	90	91.7	83.3	90	88.7

(a)

Subject	Target 1	Target 2	Target 3	Target 4	Target 5	Avg
S1	2.66	2.63	3.07	3.54	2.64	2.91
S2	6.07	3.45	3.77	3.84	2.85	3.99
S3	2.76	2.79	3.94	2.27	2.25	2.80
S4	2.65	1.77	1.88	2.85	2.10	2.25
S5	1.44	1.38	1.98	1.74	1.53	1.61
S6	4.80	4.04	3.90	3.50	3.61	3.97
Avg	3.40	2.68	3.09	2.96	2.50	2.92
S7	1.97	2.57	1.84	3.73	2.40	2.50
S8	3.37	3.65	2.65	8.17	2.40	4.05
S9	7.65	9.93	5.23	3.82	8.22	6.97
Avg	4.33	5.38	3.24	5.24	4.34	4.51

(b)

B. Spelling Phase

Table III shows the spelling rates obtained for each speller and text during the three-part spelling phase. We applied the Friedman Test—a common nonparametric statistical test for repeated measures experiments—to determine any significant differences in spelling rate due to the speller interface across all of the texts (Txt1, Txt2, Txt3, Txt4, and Txt5). The Friedman Test revealed a significant main effect of speller interface ($\chi^2 = 10.17, p < 0.01$) on spelling rate. Post-hoc tests with Bonferroni correction revealed that our speller was significantly faster than the speller of Volosyak ($p < 0.05$) and the speller of Cecotti ($p < 0.05$), but our implementations of the spellers of Volosyak and Cecotti did not differ significantly in performance from one another ($p > 0.5$). For single-word texts (Txt1, Txt2, and Txt3) average spelling rates obtained with all three spellers were similar, averaging 7.33 cpm for our speller, 6.32 cpm for the speller of Volosyak, and 6.25 cpm for the speller of Cecotti. For multiple-word texts (Txt4 and Txt5), the average spelling rate obtained with our speller (11.93 cpm) was nearly twice as fast as those obtained with the spellers of Volosyak (5.69 cpm) and Cecotti (6.22 cpm). Notably, using our speller subject S2

TABLE III
SPELLING PHASE RESULTS - SPELLING RATE. (A) AVERAGE SPELLING RATES, BY SUBJECT AND TEXT FOR OUR SPELLER, (B) AVERAGE SPELLING RATES, BY SUBJECT AND TEXT FOR SPELLER OF VOLOSYAK [3], (C) AVERAGE SPELLING RATES, BY SUBJECT AND TEXT FOR SPELLER OF CECOTTI [4]

Subject	Txt1	Txt2	Txt3	Single Words	Txt4	Txt5	Multiple Words
S1	6.75	9.93	4.31	7.00	10.96	11.09	11.03
S2	7.89	10.00	11.11	9.67	15.05	17.12	16.09
S3	2.31	10.33	8.90	7.18	12.23	13.44	12.84
S4	3.93	10.15	4.76	6.28	10.41	8.97	9.69
S5	4.82	14.74	11.16	10.24	15.67	15.53	15.60
S6	1.95	2.26	6.63	3.61	6.95	5.72	6.34
Avg	4.61	9.57	7.81	7.33	11.88	11.98	11.93

(a)

Subject	Txt1	Txt2	Txt3	Single Words	Txt4	Txt5	Multiple Words
S1	4.52	5.35	5.16	5.01	4.27	4.55	4.41
S2	5.61	6.10	10.67	7.46	5.96	5.61	5.79
S3	4.67	7.75	9.62	7.35	7.43	8.15	7.79
S4	4.45	4.78	8.72	5.98	5.92	6.52	6.22
S5	7.48	7.24	12.18	8.97	7.55	8.18	7.87
S6	2.09	2.77	4.65	3.17	2.75	1.38	2.07
Avg	4.80	5.67	8.50	6.32	5.65	5.73	5.69

(b)

Subject	Txt1	Txt2	Txt3	Single Words	Txt4	Txt5	Multiple Words
S1	3.52	4.56	3.59	3.89	3.95	4.75	4.35
S2	5.59	8.95	6.14	6.89	7.35	6.80	7.08
S3	8.25	9.15	7.53	8.31	9.03	4.22	6.63
S4	5.65	8.39	8.82	7.62	6.16	7.35	6.76
S5	3.44	10.51	9.34	7.76	8.12	9.87	9.00
S6	2.67	3.42	2.90	3.00	3.17	3.87	3.52
Avg	4.85	7.50	6.39	6.25	6.30	6.14	6.22

(c)

achieved more than 17 cpm by spelling Txt5 with zero input error and with 2.5 s of mean input latency.

Table IV shows the average input error, input latency, and input per character ratio for each of the three spellers and five texts. Input errors [Table IV(a)] increased compared to training, but were similar across both single-word and multiple-word texts at about 5%. Input latencies [Table IV(b)] also increased compared to training from 2.92 s to 3.38 s for the speller of Cecotti, 3.60 s for our speller, and 3.91 s for the speller of Volosyak. During the spelling of multiple-word texts our speller required less than half (1.60) the number of the inputs as the spellers of Volosyak (3.74) or Cecotti (3.61) to specify the same characters [Table IV(c)].

C. Simulation Study

Table V shows the results of Monte-Carlo simulations. For single-word texts, the original layout for the speller of Volosyak [3] was, on average, 0.92 cpm faster for S1-S6 than the layout we used in our implementation of this speller. For multiple-word texts, the original layout of the speller of Volosyak [3] was 0.09 cpm faster for S1-S6 than the layout we used in our implementation of this speller.

TABLE IV
SPELLING PHASE RESULTS—INPUT ERROR, INPUT LATENCY, AND INPUT PER CHARACTER RATIO. (A) % INPUT ERROR (ϵ) FOR EACH TARGET, AVERAGED ACROSS SUBJECTS, (B) INPUT LATENCY ($\mathbb{E}(T)$) IN SECONDS FOR EACH TARGET, AVERAGED ACROSS SUBJECTS, (C) INPUT/CHARACTER RATIO (C) FOR EACH TARGET, AVERAGED ACROSS SUBJECTS

Speller	Txt1	Txt2	Txt3	Single Words	Txt4	Txt5	Multiple Words
Ours	0.09	0.04	0.02	0.05	0.02	0.04	0.03
Volosyak	0	0.03	0.11	0.05	0.04	0.05	0.05
Cecotti	0.03	0.02	0.05	0.03	0.04	0.09	0.07

(a)

Speller	Txt1	Txt2	Txt3	Single Words	Txt4	Txt5	Multiple Words
Ours	3.66	3.40	3.63	3.60	3.44	3.45	3.45
Volosyak	4.35	3.46	3.93	3.91	3.55	3.31	3.43
Cecotti	3.98	2.95	3.22	3.38	3.25	2.95	3.10

(b)

Speller	Txt1	Txt2	Txt3	Single Words	Txt4	Txt5	Multiple Words
Ours	4.44	2.40	2.40	3.08	1.58	1.62	1.60
Volosyak	3.33	3.40	2.07	2.93	3.35	4.13	3.74
Cecotti	3.56	3.13	3.43	3.37	3.38	3.84	3.61

(c)

TABLE V
SIMULATION STUDY. (A) AVERAGE SIMULATED SPELLING RATES (R) OF SINGLE-WORD TEXTS (TXT1, TXT2, TXT3) FOR AN AVERAGE OF SUBJECTS S1-S6, SUBJECT S8, AND SUBJECT S9. SIMULATIONS THAT FAILED TO PRODUCE THE TARGET TEXT ARE DENOTED "-". (B) AVERAGE SIMULATED SPELLING RATES (R) OF MULTIPLE-WORD TEXTS (TXT4, TXT5) FOR AN AVERAGE OF SUBJECTS S1-S6, SUBJECT S8, AND SUBJECT S9. SIMULATIONS THAT FAILED TO PRODUCE THE TARGET TEXT ARE DENOTED "-"

Speller	Average (S1-S6)	S8	S9
Ours	9.45	6.15	3.99
Volosyak	7.26	3.32	-
Volosyak (original)	8.18	3.60	-
Cecotti	6.17	4.40	1.22

(a)

Speller	Average (S1-S6)	S8	S9
Ours	15.1	10.38	1.78
Volosyak	6.29	3.52	-
Volosyak (original)	6.38	3.58	-
Cecotti	6.03	4.49	1.21

(b)

V. DISCUSSION

When asked to specify multiple-word texts, participant performance with our speller (11.93 cpm) was nearly double that with the compared spellers (5.69 cpm for the speller of Volosyak [3] and 6.22 cpm for the speller of Cecotti [4]). This increase in performance was not due to differences in input error [Table IV(a)], in input latency [Table IV(b)], or in the size and shape of visual targets, which were identical. Instead, the performance increase can be attributed to the reduction in the number of queries required to determine the desired input character. In particular, when evaluated on multiple-word texts,

our speller required less than half the number of queries as the spellers of Cecotti [4] or Volosyak [27] to specify a character, on average [Table IV(c)]. Our use of a larger query pool and of IGR as a measure of performance to be maximized when choosing queries from this pool was what led to this reduction in the number of required queries.

Despite the fact that users specified texts faster with our speller, results show that the spelling rate for single-word texts was comparable for all three interfaces: 7.33 cpm with our speller, 6.32 cpm with the speller of Volosyak [3], and 6.25 with the speller of Cecotti [4]. We attribute the lower spelling rate of our interface under these conditions to the higher NLL values of the single-word texts (Table I). In particular, we observe a clear inverse relationship between NLL and spelling rate in Table III(a): Txt1 had an NLL of 7.19 and was spelled at a rate of 4.61 cpm, Txt3 had an NLL of 5.31 and was spelled at a rate of 7.81 cpm, and Txt2 had an NLL of 3.04 and was spelled at a rate of 9.57 cpm. Both multiple-word texts had lower NLLs—closer to the average of English text, which is 2 bits per character [25], [26]—and consequently higher average spelling rates. The reason for this trend is that, by using a language model, our speller tries to take advantage of the fact that text with high NLL (e.g., "BCI" as in Txt1) is rare in everyday conversation.

Results from the simulation studies suggest that the increase in performance with our speller as compared to the speller of Volosyak [3] was not due to the small difference in character layout. The fifth row of the character grid in our implementation was shifted one character to the right of the original implementation described in [3]. We simulated performance with our implementation and compared it with the performance of the original implementation. These simulations showed no difference between our implementation (6.38 cpm) and the original (6.29 cpm) for multiple-word texts. For single-word texts, the original implementation was slightly faster (8.18 cpm) than our implementation (7.26 cpm). We note that the original versions of the spellers of Volosyak [3] and Cecotti [4] also differed from our implementations in the locations of the targets. The effect of this change has not been investigated further, may have increased overall input latency, and represents a potential source of error.

One interesting trend that emerged from our study is that the average input latency of users with our speller (3.60 s) was higher than the average input latency for the speller of Cecotti [4] (3.38 s). The input latency of all three spellers was slower than the average input latency during training (2.92 s). Since the layout of characters in our speller changes for each query, the user needs to visually search the layout in order to locate their desired character, slowing target selection in our speller as opposed to the speller of Cecotti [4]. Our speller was designed to minimize this issue by displaying characters in a single, alphabetically ordered, row. There may be conditions, however, when smaller query pools are actually preferable. Another possible drawback of our speller is that it requires training. This requirement is a limitation of the design, but it may be possible to either minimize this training step or to use an online training paradigm. Some more advanced classifier designs also require training data [28]—it may be possible to train both the speller and the classifier simultaneously. With respect to our use of multiple query types, further work would be needed to characterize the impact of each type of query on overall performance.

VI. Conclusion

In this paper, we presented a steady-state visually evoked potential based brain-computer interface that allowed users to input text by responding to a sequence of queries. These queries were chosen from a large query pool to maximize IGR, the expected amount of information to be received per unit of time about the desired character. The computation of IGR was based on three models, a language model (that predicted likely characters based on context) and two models of user performance (input accuracy and input latency). Experimental results demonstrated that six subjects were able to use our interface to input multiple-word text at an average of 11.93 cpm, with one subject achieving an average spelling rate of 16.09 cpm.

There are several ways in which the interface described here could be improved. Input response times could be reduced through the use of different classifiers (such as those by Lin [29] or Johnson [28]), the shape and size of the stimuli could be changed, the number of input classes could be increased, word completion [30] could be implemented, and different frequencies could be assigned to the targets. As an example of how these changes might improve the interface, consider the assignment of frequencies to targets. It is clear from our training data (Table II) that this association matters. For example, during training subject S2 selected Target 5 (8.57 Hz) more than twice as fast (2.85 s) as Target 1 (7.5 Hz, 6.07 s). In other words, for subject S2, Target 5 was easier to select than Target 1. If we switch the assignment of frequencies to targets, we would expect Target 1 (8.57 Hz) to be easier to select than Target 5 (7.5 Hz). In our interface, the set of characters with which Target 1 and Target 5 are associated are different. Thus, when we change the assignment of frequencies to targets, we expect a specific set of characters to be easier to select. IGR could be used to determine the best assignment of frequencies to targets. This could improve the maximum spelling rate of participants and represents a topic of future work.

Appendix A
Derivation of Language Model Update Rule

We will proceed to derive (3), which says how to update the language model given an observed target y in response to a query f_i. Bayes' theorem tells us that

$$p_{C|Y}(c|y) = \frac{p_{Y|C}(y|c)p_C(c)}{\sum_{s\in\mathcal{C}} p_{Y|C}(y|s)p_C(s)} \tag{10}$$

for all $c \in \mathcal{C}$. Note that

$$p_{Y|C}(y|c) = \sum_{x\in\mathcal{U}} p_{Y|X,C}(y|x,c)p_{X|C}(x|c) \tag{11}$$

$$= \sum_{x\in\mathcal{U}} p_{Y|X}(y|x)p_{X|C}(x|c) \tag{12}$$

$$= p_{Y|X}(y|f_i(c)) \tag{13}$$

where (11) follows from the law of total probability, (12) follows by assumption that the observed target and the desired character are conditionally independent given knowledge of the desired target, and (13) follows by assumption of zero user error in response to a query, since in this case

$$p_{X|C}(x|c) = \begin{cases} 1, & \text{if } f_i(c) = x \\ 0, & \text{otherwise.} \end{cases}$$

After substitution of (13) into (10), we arrive at (3).

Appendix B
Derivation of Information Gain Rate

We will proceed to derive (7)–(9), which are used to compute the IGR. Equation (7) follows from the definition of conditional probability. Equation (8) follows from the law of total probability. Equation (9) follows from the law of total probability

$$p_X(x) = \sum_{c \in \mathcal{C}} p_{X|C}(x|c) p_C(c)$$

and by assumption of zero user error in response to a query, since in this case

$$p_{X|C}(x|c) = \begin{cases} 1, & \text{if } f_i(c) = x \\ 0, & \text{otherwise.} \end{cases}$$

References

[1] H. Cecotti, "Spelling with non-invasive brain-bomputer interfaces—Current and future trends," *J. Physiol.-Paris*, vol. 105, no. 1–3, pp. 106–114, Jul. 2011.

[2] B. Z. Allison, D. J. McFarland, G. Schalk, S. D. Zheng, M. M. Jackson, and J. R. Wolpaw, "Towards an independent brain-computer interface using steady state visual evoked potentials," *Clin. Neurophysiol.*, vol. 119, no. 2, pp. 399–408, Feb. 2008.

[3] I. Volosyak, "SSVEP-based Bremen-BCI interface—Boosting information transfer rates," *J. Neural Eng.*, vol. 8, no. 3, p. 036020, Jun. 2011.

[4] H. Cecotti, "A self-paced and calibration-less SSVEP-based braincomputer interface speller," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 18, no. 2, pp. 127–133, Apr. 2010.

[5] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, "Brain-computer interfaces for communication and control," *Clin. Neurophysiol.*, vol. 113, no. 6, pp. 767–791, Mar. 2002.

[6] , B. Allison, S. Dunne, R. Leeb, J. D. R. Millán, and A. Nijholt, Eds., *Towards Practical Brain-Computer Interfaces: Bridging the Gap From Research to Real-World Applications*. New York: Springer, 2012.

[7] P. Yuan *et al.*, "A study of the existing problems of estimating the information transfer rate in online brain-computer interfaces," *J. Neural Eng.*, vol. 10, no. 2, p. 026014, Apr. 2013.

[8] T. Nykopp, "Statistical modelling issues for the adaptive brain interface," M.S. thesis, Helsinki Univ. Technol., Helsinki, Finland, 2001.

[9] B. Blankertz, M. Krauledat, G. Dornhege, J. Williamson, R. Murray-Smith, and K.-R. Müller, "A note on brain actuated spelling with the Berlin brain-computer interface," in *Universal Access in Human-Computer Interaction. Ambient Interaction*, C. Stephanidis, Ed. Berlin, Germany: Springer-Verlag, 2007, vol. 4555, pp. 759–768.

[10] R. Ma, N. Aghasadeghi, J. Jarzebowski, T. Bretl, and T. Coleman, "A stochastic control approach to optimally designing hierarchical flash sets in P300 communication prostheses," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 20, no. 1, pp. 102–112, Jan. 2012.

[11] J. Mak *et al.*, "Optimizing the P300-based brain-computer interface: Current status, limitations and future directions," *J. Neural Eng.*, vol. 8, no. 2, p. 025003, Mar. 2011.

[12] R. Fazel-Rezai *et al.*, "P300 brain computer interface: Current challenges and emerging trends," *Front. Neuroeng.*, vol. 5, no. 14, Jul. 2012.

[13] M. Cheng, X. Gao, S. Gao, and D. Xu, "Multiple color stimulus induced steady state visual evoked potentials," in *Proc. IEEE EMBS Annu. Conf.*, 2001, vol. 2, pp. 1012–1014.

[14] D. Zhu, J. Bieger, G. G. Molina, and R. M. Aarts, "A survey of stimulation methods used in SSVEP-based BCIs," *Comput. Intell. Neurosci.*, vol. 2010, Jan. 2010.

[15] C. Jia, X. Gao, B. Hong, and S. Gao, "Frequency and phase mixed coding in SSVEP-based brain-computer interface," *IEEE Trans. Biomed. Eng.*, vol. 58, no. 1, pp. 200–206, Jan. 2011.

[16] J. Williamson, R. Murray-Smith, B. Blankertz, M. Krauledat, and K.-R. Müller, "Designing for uncertain, asymmetric control: Interaction design for brain-computer interfaces," *Int. J. Human-Computer Studies*, vol. 67, no. 10, pp. 827–841, 2009.

[17] H.-J. Hwang, J.-H. Lim, Y.-J. Jung, H. Choi, S. W. Lee, and C.-H. Im, "Development of an SSVEP-based BCI spelling system adopting a QWERTY-style LED keyboard," *J. Neurosci. Methods*, vol. 208, no. 1, pp. 59–65, 2012.

[18] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New York: Wiley-Interscience, 2006.

[19] V. Jurcak, D. Tsuzuki, and I. Dan, "10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems," *NeuroImage*, vol. 34, no. 4, pp. 1600–1611, Feb. 2007.

[20] L. J. Trejo, R. Rosipal, and B. Matthews, "Brain-computer interfaces for 1-D and 2-D cursor control: Designs using volitional control of the EEG spectrum or steady-state visual evoked potentials," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 14, no. 2, pp. 225–229, Jun. 2006.

[21] O. Friman, I. Volosyak, and A. Gräser, "Multiple channel detection of steady-state visual evoked potentials for brain-computer interfaces," *IEEE Trans. Biomed. Eng.*, vol. 54, no. 4, pp. 742–750, Apr. 2007.

[22] R. Prueckl and C. Guger, "A brain-computer interface based on steady state visual evoked potentials for controlling a robot," in *Bio-Inspired Systems: Computational and Ambient Intelligence*, J. Cabestany, F. Sandoval, A. Prieto, and J. M. Corchado, Eds. New York: Springer, 2009, vol. 5517, LNCS, pp. 690–697.

[23] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Trans. Commun.*, vol. 32, no. 4, pp. 396–402, Apr. 1984.

[24] D. J. Ward, A. F. Blackwell, and D. J. C. MacKay, "Dasher: A gesture-driven data entry interface for mobile computing.," *Human-Computer Interact.*, vol. 17, no. 2/3, pp. 199–228, Jun. 2002.

[25] R. Begleiter, R. El-Yaniv, and G. Yona, "On prediction using variable order Markov models," *J. Artif. Intell. Res.*, vol. 22, pp. 385–421, Dec. 2004.

[26] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press, 2002.

[27] I. Volosyak, A. Moor, and A. Gräser, "A dictionary-driven SSVEP speller with a modified graphical user interface," *Adv. Computat. Intell.*, vol. 6691, pp. 353–361, 2011.

[28] E. C. Johnson, J. J. Norton, D. Jun, T. Bretl, and D. L. Jones, "Sequential selection of window length for improved SSVEP-based BCI classification," in *Proc. IEEE EMBS Annu. Conf.*, 2013, pp. 7060–7063.

[29] Z. Lin, C. Zhang, W. Wu, and X. Gao, "Frequency recognition based on canonical correlation analysis for SSVEP-based BCIs," *IEEE Trans. Biomed. Eng.*, vol. 53, no. 12, pp. 2610–2614, Dec. 2006.

[30] A. Vilic, T. W. Kjaer, C. E. Thomsen, S. Puthusserypady, and H. B. Sorensen, "DTU BCI speller: An SSVEP-based spelling system with dictionary support," in *Proc. IEEE EMBS Annu. Conf.*, 2013, pp. 2212–2215.

Abdullah Akce received the B.S. degree in computer engineering from Bogazici University, Istanbul, Turkey, in 2007, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2013.

His research interests lie at the intersection of robotics, machine learning, and human–computer interaction. He is currently with Google Inc.

James J. S. Norton is from Ormond Beach, FL, USA. He received the B.S. degree in psychology from the University of Florida, Gainesville, FL, USA. He joined the Bretl Research Group at the University of Illinois, Urbana, IL, USA, in 2010, where he is currently working toward the Ph.D. degree in neuroscience with a concentration in neuroengineering.

His primary research interests include the development of brain–machine interfaces based on electroencephalography and the noninvasive study of human visual processing.

Timothy Bretl (S'02–M'05) received the B.S. degree in engineering and the B.A. degree in mathematics from Swarthmore College, Swarthmore, PA, USA, in 1999, and the M.S. and Ph.D. degrees both in aeronautics and astronautics from Stanford University, Stanford, CA, USA, in 2000 and 2005, respectively. Subsequently, he was a Postdoctoral Fellow in the Department of Computer Science, also at Stanford University.

Since 2006, he has been with the University of Illinois at Urbana-Champaign, where he is an Associate Professor of Aerospace Engineering with an affiliate appointment in the Coordinated Science Laboratory.

Dr. Bretl was a recipient of the National Science Foundation Faculty Early Career Development Award, in 2010.