A POMDP Approach to Optimizing P300 Speller BCI Paradigm

Jaeyoung Park and Kee-Eung Kim *Member, IEEE*

Abstract

To achieve high performance in brain-computer interfaces (BCIs) using P300, most of the work has been focused on feature extraction and classification algorithms. Although significant progresses have been made in such signal processing methods in the lower layer, the issues in the higher layer, specifically determining the stimulus schedule in order to identify the target reliably and efficiently, remain relatively unexplored. In this article, we propose a systematic approach to compute an optimal stimulus schedule in P300 BCIs. Our approach adopts the partially observable Markov decision process (POMDP), which is a model for planning in partially observable stochastic environments. We show that the thus obtained stimulus schedule achieves a significant performance improvement in terms of the success rate, bit rate, and practical bit rate through human subject experiments.

Index Terms

Brain-computer interface (BCI), Electroencephalography (EEG), Partially observable Markov decision process (POMDP), P300, P300 speller

I. Introduction

Brain-computer interfaces (BCIs) interpret the electrical activities of the neurons in brain and convey corresponding messages or commands to the external systems [1]. Invasive methods in BCIs directly record electrical activities by, for example, implanting electrodes into the brain, whereas non-invasive methods in BCIs record indirectly from the outside of the brain. Non-invasive methods such as using the electroencephalography (EEG) recorded from the scalp are widely advocated for the practicality due to the user safety, ease of use, and low set-up cost [2].

The authors are with the Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon 305-701, Korea (e-mail: jypark@ai.kaist.ac.kr; kekim@cs.kaist.ac.kr).

Manuscript received January ?, ?; revised January ?, ?.

One of the most reliable signal features in EEG for constructing BCIs is the P300 component in the event-related potential (ERP) [3]. The ERP refers to the measurement of brain response resulting from a thought or perception, and the P300 is a positive peak component in the ERP at about 300ms after an infrequent and significant stimulus is given [4], [1]. Because the P300 can be utilized to identify the user's intent, one can construct the BCI based on the P300. The P300 speller system [4] for the purpose of emulating a keyboard is one of the most well known applications of P300 BCIs. In a typical setting of the P300 speller system, the user faces the 6×6 matrix on a video display terminal with each entry containing one letter (see Fig. 1). Among 36 letters in the matrix, the user gazes at the letter that the user desires to input while all the letters in a row or a column are flashed (stimulated) in a random order. If the gazed letter is flashed, the P300 is elicited with some chance of failure at about 300ms after the flash. Thus, using a method for detecting the P300, we can identify the letter that the user is gazing at, i.e. intending to input. However, the elicitation and detection of the P300 is inherently imperfect, and therefore each row and column is flashed multiple times and the detection results are combined in order to improve the detection accuracy. The P300 speller system uses a visual flash as a stimulus but in general other types of stimuli (e.g. sound) can be used for the P300 BCI.

The main motivation for the work presented in this paper was the inefficiency in the flash schedule of random sequences currently adopted by most of the work on P300 BCIs. For example, the P300 speller system in the literature generates the same number of flashes for every row and column in a random order. However, given additional information such as the temporal regularities in the user's intent or the characteristics of the P300 detection results, we may want to incorporate this information for determining the optimal flash schedule that identifies the user's intent accurately while using the smallest number of flashes. For example, in the P300 speller system, if it is very unlikely that the desired letter is in the first row, flashing the first row may be of little value. In contrast, if it is highly likely that the desired letter is either in the second or the third row but highly uncertain between them, it is desirable to flash one of these rows in order to resolve the uncertainty. In addition, it is hard to determine when to stop generating stimuli and make decision on the user's intent. Some studies on the P300 speller system have proposed using thresholds on the sum of detection confidence values, but the question remains how this technique could be integrated into the computation of an optimal flash schedule.

This article presents a principled and integrated approach to the aforementioned issues in P300 BCIs. Specifically, we use the partially observable Markov decision process (POMDP) [5] to model the problem of achieving the maximum accuracy in identifying the user's intent while minimizing the number of flashes. The POMDP is a rigorous framework for modeling sequential decision making problems under uncertainty in the dynamics and sensing. It is a natural model for P300 BCIs since the temporal regularities in the user's intent and the noisy characteristics in the P300 detection result can be represented in an integrated way. Being a decision-theoretic model, the above aspects are modeled using probabilities, and the rewards (or costs) are specified for each flash and final decision. Using an algorithm for solving POMDPs, we can obtain an optimal control strategy that determines where to flash given the past history of flashes and the corresponding P300 detection results. In addition, the control strategy will determine whether to continue flashing or stop with a decision on the user's intent, solely based on the maximization of overall accumulated rewards. We show that the P300 speller system optimized using the POMDP achieves a significant improvement in the communication bandwidth. We also provide some speed up techniques that handle the computational intractability issues in solving the POMDP. Throughout this paper, we use the P300 speller system to demonstrate our approach, but we emphasize that it can be generalized to other P300 BCI paradigms. The preliminary work related to this article was presented in [6], [7].

II. Preliminaries

Fig. 1 shows the overall architecture commonly adopted by P300 speller systems. This section describes our implementation of each component in the system.

A. Flash Epoch Design

Our flash epoch design follows the typical setting in P300 speller systems: the user gazes at the letter that the user desires to input (i.e., target letter) and the 6 letters in a row or column of the 6×6 matrix are flashed together at the beginning of each flash epoch. A flash epoch takes a time interval of 250ms. Each flash turns on the letters in the row or column for 125ms and then turns off for the remaining 125ms. A trial is defined to be a sequence of flash epochs for identifying one target letter. Each trial is composed of a row trial for identifying the row containing the target letter (i.e., target row), followed by a column trial for identifying the target column. Row trials are composed of row flashes, and column trials are composed of column flashes. The target letter is determined by the identified target row and target column. A pause interval of 2.5s is given between consecutive trials (see Fig. 2).

B. Signal Acquisition, Preprocessing, and P300 Detection

We used Biopac MP150 data acquisition system with 16 channels at 1kHz sampling rate for acquiring EEG signals. The locations of electrodes, according to the international 10-20 system, were as follows:

AFz, Fz, F3, F4, Cz, C3, C4, T7, T8, Pz, P3, P4, P7, P8, O1, O2. We took the window of the EEG signal data between 200ms and 450ms after each flash (Fig. 2) since P300 is expected to appear *approximately* at 300ms after the flash.

This window of data was then fed into the preprocessor to extract the relevant features from the raw signal data, and subsequently passed to the classifier to detect the existence of the P300. In order to construct the preprocessor and the P300 classifier, we prepared the training data where each instance was the window of EEG signal data for either a target or non-target letter. These training data were collected on each human subject using the flash epoch design described in the previous subsection.

In the preprocessor, the window of raw signal data were band-pass filtered (0.5-30Hz) and down-sampled to 100Hz. The spatial projection algorithm [8] was then used to extract features from the signals. This algorithm generates the set of linear filters that maximally discriminates between the target and non-target instances. Formally, given an instance E_i (an $m \times n$ matrix where m=16 is the number of channels and n=25 is the number of signal samples) for the i^{th} flash, the spatial projection algorithm generates a maximum of m linear filters. Each filter f_j is an m-dimensional vector, and generates one n-dimensional feature vector by projection $f_j^T E_i$ (an n-dimensional row vector). The number of filters was determined by cross-validation. In our case, we limited the maximum number of filters to 5, since this number was sufficiently large for most of the human subjects participating in this study and guaranteed timely processing of data.

The classifier that detects the existence of the P300 in the preprocessed EEG signal instance was obtained by the LIBLINEAR package [9]. In order to represent the likelihood of the P300 existence, we used L2-regularized logistic regression which outputs a real number between 0 and 1. The parameters for executing LIBLINEAR were determined by 5-fold cross-validation on the training data.

C. Conventional Flash Schedule (π_{RAND})

The P300 speller system conventionally uses a random flashing scheme (π_{RAND}) where the flash sequences are generated randomly and the desired letter is determined after a prescribed number of flashes or the confidence reaches above a prescribed threshold. The row and column trials in our π_{RAND} implementation use a constant number of flashes. At the end of trials, we take the sum of the output values of the P300 classifier for each row and column, and determine the target row and column with the maximum value.

D. Partially observable Markov decision processes

The partially observable Markov decision process (POMDP) [5] is a mathematical model for sequential decision making problems under uncertainty in the dynamics and observation. It is defined by an 8-tuple $\langle S, A, Z, b_0, T, O, R, \gamma \rangle$ where: S is the set of environment states; A is the set of actions available to the agent; Z is the set of all possible observations; b_0 is the initial belief state where $b_0(s)$ denotes the probability that the environment starts in state s; T is the transition probability where $T^a_{s,s'}$ denotes the probability that the environment changes from state s to s' when executing action a; O is the observation probability where $O^z_{s,a}$ denotes the probability that the agent makes observation z when executing action a and arriving at state s; R is the reward function where R^a_s denotes the reward received by the agent when executing action a in state s; γ is the discount factor such that $0 \leq \gamma < 1$.

Since the agent cannot directly know the environment state due to the uncertainty in observations, it maintains the probability distribution over the states based on the history of actions and observations. The probability distribution over the states is defined as a belief state where $b_t(s)$ denotes the probability that the state is s at time step t. The belief state b_t can be regarded as the posterior distribution of the states given the initial belief b_0 and the history of actions and observations:

$$b_t(s) = P(S_t = s | b_0, a_0, z_1, a_1, z_2, \cdots, a_{t-1}, z_t)$$

After executing action a_t and making observation z_{t+1} in belief state b_t, the belief state $b_{t+1} = \tau(b_t, a_t, z_{t+1})$ at the next time step is updated by the Bayes rule,

$$b_{t+1}(s') = \frac{O^{z_{t+1}}_{s',a_t} \sum_{s \in S} T^{a_t}_{s,s'} b_t(s)}{P(z_{t+1} | b_t, a_t)} \tag{1}$$

where $P(z_{t+1} | b_t, a_t)$ can be regarded as the normalizing constant.

A policy determines the actions to be executed by the agent on a given belief state. Formally, a policy π of a POMDP can be defined as a mapping from belief states to actions, i.e., $\pi : \Delta S \rightarrow A$. Every policy has an associated value function, which is (in the case of infinite horizon problems) the expected cumulative discounted reward by following the policy starting from a given belief state. When solving a POMDP, we search for an optimal policy that maximizes the value for each belief state. The maximum value for the belief state can be defined recursively

$$V^*(b) = \max_a \left[R(b,a) + \gamma \sum_z P(z|b,a) V^*(\tau(b,a,z)) \right],$$

where $R(b,a) = \sum_s R^a_s b(s)$. The optimal value function V^* can be computed by a series of dynamic programming backups

$$\begin{aligned} V_n(b) &= HV_{n-1}(b) \\ &= \max\nolimits_a \left[R(b,a) + \gamma \textstyle\sum_{z\in Z} P(z|b,a)V_{n-1}(\tau(b,a,z))\right], \end{aligned}$$

for every belief state $b \in \Delta S$. We can also derive that value function V_n is piecewise linear and convex, hence it can be represented as a set of vectors $\Gamma_n = \{\alpha_1, \cdots, \alpha_m\}$ and the value at a particular belief state b is calculated as

$$V_n(b) = \max\nolimits_{a\in\Gamma_n} \textstyle\sum_{s\in S} \alpha(s)b(s).$$

Once we compute the optimal value function V^*, the optimal policy is obtained by

$$\pi^*(b) = \arg\max\nolimits_a \left[R(b,a) + \gamma \textstyle\sum_z P(z|b,a)V^*(\tau(b,a,z))\right]. \tag{2}$$

It is intractable to exactly compute the optimal value function and optimal policy, mainly because there are infinitely many belief states. Some of the POMDP algorithms such as the witness algorithm [5] exploit the piecewise linear and convex property of value functions, but they are still limited to problems of small sizes. Instead, approximate algorithms such as point-based value iteration (PBVI) [10] or heuristic search value iteration (HSVI) [11], [12] are used in practice. These approximate algorithms are scalable, yet the solutions found are almost optimal in various benchmark POMDP problems. Since a complete review of POMDP algorithms is outside the scope of this article, we refer the readers to the references mentioned above.

III. Flash Strategy Using POMDPs

A. POMDP-based Flash Strategy (π_{POMDP})

The problem of finding an optimal flash schedule for the P300 speller system can be naturally modeled as a POMDP. The optimization objective of the flash schedule is reflected in the reward function that favors a high accuracy in identifying the target letter while using the minimum number of flashes. Note also that the flash schedule corresponding to the optimal policy of the POMDP is inherently a *strategy* rather than a static schedule: it *adaptively* determines which row or column to flash in each epoch based on the history of P300 detection results, which is summarized by the belief state representing the posterior probability of each row and column being the target.

We compute two optimal flash strategies by constructing and solving two POMDP models, one for the row trials and the other for the column trials. Once we obtain them, we first execute the row flash strategy until the target row is determined, and then the column flash strategy until the target column is determined. We only describe below our POMDP modeling of row trials since the model for column trials is almost identical.

Let N be the number of rows in the matrix. Note that N=6 in this article since we consider the P300 speller with a 6×6 matrix. The states in the POMDP correspond to the candidate target rows, hence a total of N states. For each row in the matrix, we can either flash it in the hope of detecting P300 (N flash actions) or claim that it's the target row (N select actions), hence a total of $2N$ actions. The output value from the P300 classifier serves as the observation, where the continuous output value between 0 and 1 was discretized into intervals of size 0.1 (e.g. z_1 for the output value in $[0.0, 0.1)$, z_2 for the output value in $[0.1, 0.2)$, etc.), hence a total of 10 observations.

Since we prefer finding the target row as soon as possible, we assigned the reward of -1 for the flash actions. In addition, since we want to find the target row as accurately as possible, we assigned the reward of +10 for the select actions that correctly identify the target row (e.g., select row #1 when the target is row #1), and -100 for those that incorrectly identify the target row (e.g., select row #2 when the target is row #1).

The transition probabilities for the flash actions were defined to be the identity matrix based on the assumption that the target row does not change within a trial. The transition probabilities for the select actions were defined to be the uniform distribution based on the assumption that the target row will be reset to a new target row with equal probability after each trial.

The observation probabilities for the flash actions were obtained using the classifier output values for each subject. Specifically, we obtained the histogram of classifier output values when the target row is flashed from the training data for the P300 classifier, and used this empirical distribution for the observation probabilities for the flash actions on the target row. We followed the same method for obtaining the observation probabilities for the non-target rows. Note that if the target row is flashed, it is likely to obtain one of the observations from z_6 to z_{10} since we expect a high output value from the classifier. However, since the classifier is not perfect, there is also a non-zero probability of obtaining one of the observations from z_1 to z_5. The observation probabilities model these errors in the P300 detection results. Since the training data collected during the pilot trial is not perfectly reliable because of various reasons (e.g., subject fatigue or attentiveness), we smoothed the empirical distribution by mixing it with the uniform distribution:

$$O^z_{s,a} = (1 - q_O)\hat{O}^z_{s,a} + q_O \tfrac{1}{|Z|}$$

where $\hat{O}_{s,a}$ is the empirical distribution from the training data and q_O is the mixing weight. In the experiments, we simply set q_O=0.3 for all subjects. For select actions, we used the uniform distribution for the observation probabilities since the P300 classifier output value is obtained without flash and thus contains no information.

The initial distribution b_0 on the states was set to the uniform distribution since we assume no prior knowledge about the likelihood of each row containing the target letter.

We refer to the flash strategy computed from the above POMDP as π_{POMDP}.

B. POMDP using Bigram Model ($\pi_{\textit{POMDP-BIGRAM}}$)

We can incorporate prior knowledge into the POMDP model and further improve the performance of the flash strategy. This is especially true for the P300 speller system since there is a strong regularity in the sequence of alphabets in a natural language. There is a wealth of language models for capturing this regularity [13], and it has been suggested that the P300 speller system can benefit from them [14]. In this paper, we show how the bigram language model can be used in the POMDP.

The bigram model is defined by the conditional probability distribution $P(w_t|w_{t-1})$ over the letters w_t in the speller matrix given the previous letter w_{t-1}. We estimated the bigram model using 9 novels available online as a part of the project Gutenberg. Suppose that $l_{i,j}$ is the letter in the row i and the column j. Using the bigram model, the probability of target letter being $l_{i,j}$ is computed as

$$P(w_t = l_{i,j}) = \sum_{i',j'} P(w_t = l_{i,j}|w_{t-1} = l_{i',j'})P(w_{t-1} = l_{i',j'}). \tag{3}$$

The above equation requires the probability $P(w_{t-1} = l_{i',j'})$ of the previous target letter being $l_{i',j'}$. In order to obtain this probability, let b^r_{t-1} be the final belief state in the previous row trial, and b^c_{t-1} be the final belief state in the previous column trial. We then have

$$P(w_{t-1} = l_{i',j'}) = b^r_{t-1}(i')b^c_{t-1}(j'). \tag{4}$$

Using Eqn. 3 and Eqn. 4, the initial belief state for the current row trial is defined as

$$b^r_{t,0}(i) = \textstyle\sum_j P(w_t = l_{i,j}).$$

In order to determine the initial belief state for the current column trial, let b^r_t be the final belief state of the current row trial (finished just before the onset of the current column trial). We then have

$$b^c_{t,0}(j) = \textstyle\sum_i b^r_t(i)\eta^i_t P(w_t = l_{i,j}),$$

where $\eta^i_t = 1/\sum_k P(w_t = l_{i,k})$.

Since there are chances where the previous target letter was incorrectly determined or the pattern in the alphabet sequence differs significantly from the estimated bigram model, we mixed the bigram-predicted probability with the uniform distribution:

$$b^r_{t,0} = (1-q_T)b^r_{t,0} + q_T\tfrac{1}{N}, \text{ and } b^c_{t,0} = (1-q_T)b^c_{t,0} + q_T\tfrac{1}{N},$$

where q_T is the mixing weight. We set q_T=0.6 in the experiments.

In summary, the above is basically the same as π_{POMDP} described in the previous subsection, except that the initial belief state is computed from the bigram model instead of using the uniform distribution. We refer to this strategy as $\pi_{\text{POMDP-BIGRAM}}$.

C. Constraints on Flash Strategy

When we compute the optimal flash strategy from the POMDP, we have to take into account two constraints that originate from the inherent characteristics of the P300.

The first constraint is the delay in the P300 detection. In our P300 speller system, the time to obtain the P300 detection result takes more than 500ms (but less than 750ms) because an epoch ends at 450ms after the flash and a small amount of additional delay is incurred by processing the signal data. Thus, in the context of the POMDP, the relevant observation is available only after executing the next two actions because each action is executed in 250ms intervals (Fig. 2). We used the extended version of the POMDP model with delayed observations [15].

The second constraint is the phenomenon known as the *repetition blindness*, where the P300 may not be elicited if the target letter is repeatedly flashed within 500ms [16], [17]. For example, if the first row is flashed within 500ms from the last flash on the first row, the P300 may not be existent in EEG even though the first row contains the target letter. This constraint is handled by simply avoiding the flash on the same row or column within 500ms.

These two constraints lead to the modified definition of the optimal value function:

$$\begin{aligned} V^*(b, a_{t-2}, a_{t-1}) \\ &= \max_{a_t \in A - \{a_{t-2}, a_{t-1}\}} \sum_{s_{t-2}, s_{t-1}, s_t} b(s_{t-2}) \left[\prod_{i=t-2}^{t-1} T^{a_i}_{s_i, s_{i+1}} \right] R^{a_t}_{s_t} \\ &\quad + \gamma \sum_z P(z|b, a_{t-2}) V^*(\tau(b, a_{t-2}, z), a_{t-1}, a_t). \end{aligned}$$

The first term in the right-hand side represents the expected immediate reward, and the second term represents the expected maximum return after executing action a_t. Note that the value function now depends on the sequence of actions executed during the delay, as well as the current belief state b.

We modified the PBVI algorithm to optimize based on the above optimal value function while exploiting the symmetry [18], [19] in the model to achieve a significant reduction in the solution time. The description of the algorithm is provided in the Appendix.

IV. Experiments

A. Human Subjects and Study Protocol

10 able-bodied students (7 male and 3 female students) at Korea Advanced Institute of Science and Technology (KAIST) participated in this study. This study was approved by the KAIST Institutional Review Board (KH2010-24).

Our experiment consisted of 4 consecutive sessions for each subject: TRAIN, RAND, POMDP, POMDP-BIGRAM. In the TRAIN session, we collected training data for the preprocessor and P300 classifier using 20 trials. Each trial randomly assigned a target letter among 36 letters, and each trial flashed 10 times for each row and column in a random order. Thus, there were a total of 10*6*2 = 120 row and column flashes in each trial. The observation probabilities of the POMDP model for the subject were also obtained from the histogram of the training data and P300 classifier output values. In the RAND, POMDP, and POMDP-BIGRAM sessions, “MACHINE LEARNING” (16 letters including the space) was spelled by the subject using π_{RAND}, π_{POMDP}, and $\pi_{\text{POMDP-BIGRAM}}$, respectively. Subjects A, C, E, G, and I performed the sessions in the order of TRAIN, POMDP, POMDP-BIGRAM, and RAND. Subjects B, D, F, H, and J performed the sessions in the order of TRAIN, RAND, POMDP, and POMDP-BIGRAM.

B. Measurements

We measured the performance of each flash control in terms of the accuracy, bit rate, and practical bit rate.

The accuracy is defined as the ratio of the number of correctly spelled letters to the total number of input characters.

The bit rate [20], [21] measures the quantity of the transferred information per unit time. It is defined as $B \cdot D$, where B is the number of bits per decision, and D is the number of decisions per unit time. Given the total number of characters N on the screen and the accuracy P, the number of bits per decision is defined by,

$$B = \log_2 N + P \log_2 P + (1 - P) \log_2 \frac{1 - P}{N - 1}.$$

The number of decisions per unit time is computed by $D = S/T$, where S is the number of letters attempted to input, and T is the total time taken by the user.

The practical bit rate [22] adjusts the quantity by the additional time required to correct the error in the target letter selection. For an incorrectly selected letter, the user has to delete it and re-input the letter. Hence, two additional trials should be performed by the user to attempt a correction. Since this error correction attempt may fail again, we have to compute the expected total number of trials to succeed in error correction. Given the error rate $E = (1 - P)$ and the number of letters S to input, the expected number of trials required to correctly input all the letters is defined by

$$S + 2(SE) + 2((2SE)E) + 2((2(2SE)E)E) + \cdots$$
$$= S \sum_{i=0}^{\infty} (2E)^i = S/(1 - 2E)$$

when $E < 0.5$. Thus, the practical bit rate uses $D = S/T'$, where T' is the time required to input $S/(1 - 2E)$ letters, which is $T' = (T/S) \cdot (S/(1 - 2E)) = T/(1 - 2E)$.

C. Results

Table I shows the performances of π_{RAND}, π_{POMDP}, $\pi_{\text{POMDP-BIGRAM}}$. For π_{RAND}, we used 10 flashes for each row and column, resulting in a total of 120 flashes per target letter selection. Since π_{RAND} does not have a mechanism to determine when to stop flashing, we report its performance when the practical bit rate hits its maximum value. Hence, 120 is the maximum number of flashes per target letter selection for π_{RAND}.

We conducted a one-way within-subject ANOVA test on the practical bit rate; Mauchly's test indicated that the sphericity assumption was violated (p-value $\approx 0.000 < 0.05$), thus the degree of freedom was corrected according to the Greenhouse-Geisser estimates of the sphericity ($\epsilon = 0.537$). There was a significant difference among the three flash controls ($F(1.073, 9.659) = 15.601$, p-value $= 0.003 < 0.05$). A post-hoc test using Bonferroni indicated that π_{POMDP} was significantly better than π_{RAND} (p-value $= 0.027 < 0.05$), and $\pi_{\text{POMDP-BIGRAM}}$ was again significantly better than π_{POMDP} (p-value $= 0.001 < 0.05$) as well as π_{RAND} (p-value $= 0.007 < 0.05$). In terms of the average practical bit rate, π_{POMDP} achieved a 55% performance improvement over π_{RAND}, and $\pi_{\text{POMDP-BIGRAM}}$ achieved a 86% performance improvement over π_{RAND} and 20% over π_{POMDP}.

V. Discussion

We have presented a POMDP-based optimization of flash control in the P300 speller system, which significantly outperforms the conventional random flash control. The POMDP-based flash strategy achieves a significant reduction in the number of flashes while maintaining high accuracy by avoiding unnecessary flashes on unlikely rows or columns.

Previous studies on improving the performance of P300 BCIs were mostly focused on the signal processing filters and the P300 classifiers [2], with a few exceptions that address stimulus paradigm. Luo and Min [23] proposed a method for improving the P300 detection accuracy by computing the set of letters to flash simultaneously, extending the traditional row and column flashes. Hill et al. [24] presented a method that computes the flash schedule as well as the set of letters to flash, using a heuristic based on the error correcting code design. However, this approach is still limited in the sense that the flash schedule is fixed a priori: in the context of control theory, the optimized flash schedule is an open-loop control since the P300 detection results within trials are not used and the system simply follows a prescribed flash schedule. In contrast, our optimized flash controls are inherently closed-loop since they determine where to flash based on the history of the P300 detection results. The POMDP is a systematic and principled framework for obtaining an optimized closed-loop flash control.

Several advances were made in this study from our own prior work [6]. First, we no longer depend on the parametric assumption regarding the observation probabilities. Instead of assuming the beta distribution for the observation probabilities and pre-computing optimal policies for different parameter settings, we directly used the empirical distribution from the training data used for the P300 classifier and the histogram of its output values. Second, the computation time for obtaining optimal flash strategies was reduced by orders of magnitude, taking almost the same time as training the P300 classifier. We achieved this speedup by exploiting the symmetries in the POMDP model [18], [19]. Hence, we were able to compute the optimal flash strategy in each human subject session, rather than choosing one of the pre-computed flash strategies. Our implementation of the algorithm takes only a few minutes to find the optimal flash strategy.

Cautious readers may question why we adopted the two-phase trial completing the row trial before starting the column trial, rather than mixing them. In principle, we could have an integrated POMDP model that allows executing row or column flashes at any time. However, we noted that an optimal flash strategy will nonetheless follow the two-phase behaviour based on the following reasoning: since a row flash provides information only related to rows, there is a chance of consecutively flashing the target letter whenever a column flash immediately follows a row flash. Hence, in order to minimize the chance of information loss due to the repetition blindness, an optimal flash strategy should minimize the interchange between row and column flashes.

There are additional ideas for the future work that may further improve the performance of our approach. First, it would be interesting to adopt POMDPs with continuous observations to take the P300 classifier output values without discretization. There are a few algorithms for POMDPs with continuous observations [25], but we have not yet tested whether these algorithms can yield the optimal flash strategy in a fast manner or such strategy significantly outperforms the one from the discrete observation POMDP. Second, although the mixing parameters q_O and q_T were set to constants in our current approach, it would be beneficial to have a procedure that automatically adjust the values through the interactions with the human subject, as the characteristics of EEGs change over time due to the subject fatigue and attentiveness. Third, especially in the case of $\pi_{\text{POMDP-BIGRAM}}$, rather than identifying the sequential patterns offline, it would be helpful to estimate them online through the interaction with the human subject, since the repositories such as novels may not be available for other task domains (e.g., navigating to places using a BCI-controlled wheelchair). Fourth, we could incorporate recent results on optimizing the set of letters to flash [23], [24] into the actions of the POMDP, extending the row and column flashes. Lastly, we hope to eliminate the pilot session where we collect the subject-dependent training data for obtaining the P300 classifier and constructing the POMDP model. A naive way that may achieve this is to use a single global P300 classifier trained against a number of human subjects and use reinforcement learning algorithms to learn the POMDP model while interacting with the human subject. However, even a higher computational complexity remains as a challenge.

Appendix A
POMDP with Observation Delay

The standard POMDP assumes that an observation is obtained before executing the action at the next time step. But in some cases, the observation is not available for a number of time steps even though we have to execute actions at every time step. We refer this situation to the POMDP with observation delays [15]. In general, the time steps in the observation delay can be constant or probabilistic. In this article, we only consider the constant time steps for the observation delays. In this case, a history of actions and observations is given by $a_0, a_1, \ldots, a_d, z_{d+1}, a_{d+1}, z_{d+2}, \ldots$ where d is the number of time steps in the observation delay. Note that the z_{d+1} is the observation relevant to the environment state and action at time step 0. Hence, we revise the definition of the belief state b_t so that it is the probability distribution of the states d time steps ago in the past, and the belief state is updated by $b_{t+1} = \tau(b_t, z_{t+1}, a_{t-d})$. This update requires keeping track of the actions executed during the observation delay.

The optimal value function of the POMDP with d-step observation delay is now defined by the equation

$$\begin{aligned} &V^*(b, a_1, \ldots, a_d) \\ &\quad = \max_a \Bigg[\sum_{s_0,\ldots,s_d} b(s_0) [\prod_{i=0}^{d-1} T^{a_{i+1}}_{s_i,s_{i+1}}] R^a_{s_d} \\ &\qquad\qquad + \gamma \sum_z Pr(z|b, a_1) V^*(\tau(b, a_1, z), a_2, \ldots, a_d, a) \Bigg], \end{aligned}$$

where a_d is the most recent action executed during the delay and a_1 is the oldest action executed during the delay. The first term in the right-hand side represents the expected immediate reward. The second term represents the expected return after executing action a. Note that the above value function depends on the state distribution of d time steps before and the actions executed during the observation delay.

To simplify the notation let $a_{1:d}$ be the series of actions $a_1, \ldots, a_d$ so that we denote the optimal value function as $V^*(b, a_{1:d})$. The optimal value function V^* can be obtained by iterative dynamic programming backup

$$\begin{aligned} V_n(b, a_{1:d}) = \max_a \Bigg[&\sum_{s_{0:d}} b(s_0) [\prod_{i=0}^{d-1} T^{a_{i+1}}_{s_i,s_{i+1}}] R^a_{s_d} \\ &\qquad + \gamma \sum_z Pr(z|b, a_1) V_{n-1}(\tau(b, a_1, z), a_{2:d}, a) \Bigg], \end{aligned}$$

for every belief state $b \in \Delta S$ and actions during the delay $a_{1:d} \in A^d$. Moreover, it can be shown that the value function $V_n(b, a_{1:d})$ is piecewise linear and convex in belief space [15]. We can therefore represent the value function using the set of α-vectors for each possible actions executed during the delay: $\Gamma^{a_{1:d}}_n$, $\forall a_{1:d} \in A^d$. Using this representation, the value of the belief state b with actions $a_{1:d}$ executed during the delay is calculated by

$$V_n(b, a_{1:d}) = \max_{\alpha \in \Gamma^{a_{1:d}}_n} b \cdot \alpha.$$

The set $\Gamma^{a_{1:d}}_n$ is computed iteratively. Given $\Gamma^{a_{1:d}}_{n-1}$ for all $a_{1:d} \in A^d$, we generate intermediate sets $\Gamma^{a_{1:d},a,*}_n$ and $\Gamma^{a_{1:d},a,z}_n$, $\forall a \in A$, $\forall z \in Z$ such that

$$\Gamma^{a_{1:d},a,*}_n = \left\{ \alpha | \alpha(s_0) = \sum_{s_{1:d}} [\prod_{i=0}^{d-1} T^{a_{i+1}}_{s_i,s_{i+1}}] R^a_{s_d} \right\},$$

$$\Gamma^{a_{1:d},a,z}_n = \left\{ \alpha | \alpha(s) = \gamma \sum_{s' \in S} T^{a_1}_{s,s'} O^z_{s',a_1} \alpha'(s') \right\}, \forall \alpha' \in \Gamma^{a_{2:d},a}_{n-1}.$$

Then $\Gamma_n^{a_{1:d}} = \bigcup_{a\in A} \Gamma_n^{a_{1:d},a}$ where $\Gamma_n^{a_{1:d},a} = \Gamma_n^{a_{1:d},a,*} \oplus \Gamma_n^{a_{1:d},a,z_1} \oplus \Gamma_n^{a_{1:d},a,z_2} \oplus \cdots$. Hence every $\alpha_n^{a_{1:d},a} \in \Gamma_n^{a_{1:d},a}$ is computed by the equation

$$\alpha_n^{a_{1:d},a}(s_0) = \sum_{s_{1:d}} \Big[\prod_{i=0}^{d-1} T_{s_i,s_{i+1}}^{a_{i+1}} \Big] R_{s_d}^a + \gamma \sum_{s_1} T_{s_0,s_1}^{a_1} \sum_z O_{s_1,a_1}^z \alpha_{n-1,z}^{a_{2:d},a}(s_1),$$

where the α-vector $\alpha_{n-1,z}^{a_{2:d},a}$ for each observation z comes from the set $\Gamma_{n-1}^{a_{2:d},a}$

Appendix B
Permutable POMDP with Observation Delay

We can exploit the symmetries in the model to speed up algorithms for finding an optimal policy [18], [19]. Let the state permutation $\pi : S \to S$ be a one-to-one and onto function. The π maps one state ordering to another one. Let action permutation $\rho : A \to A$ and observation permutation $\tau : Z \to Z$ be defined in the same manner. After applying the state permutation π to the belief state b, we get reordered belief state b^π, i.e. $b^\pi(\pi(s)) = b(s)$. We can also apply the state permutation to the α-vector α, which gives reordered α-vector α^π, i.e. $\alpha^\pi(\pi(s)) = \alpha(s)$. To simplify the notations, let $\rho(a_{1:d})$ be the permuted action sequence $\rho(a_1), \rho(a_2), \ldots, \rho(a_d)$.

Theorem 1: Given a state permutation π, if there exists an action permutation ρ and observation permutation τ such that

$$(1)\ T_{s,s'}^a = T_{\pi(s),\pi(s')}^{\rho(a)} \qquad \forall s, s' \in S, a \in A$$

$$(2)\ O_{s,a}^z = O_{\pi(s),\rho(a)}^{\tau(z)} \qquad \forall s \in S, a \in A, z \in Z$$

$$(3)\ R_s^a = R_{\pi(s)}^{\rho(a)} \qquad \forall s \in S, a \in A$$

Then for every α-vector $\alpha \in \Gamma_n^{a_{1:d}}$, there exists $\alpha^\pi \in \Gamma_n^{\rho(a_{1:d})}$ for all $n \in \{0, 1, 2, \cdots\}$.

Proof: For $n = 0$, the theorem holds if we initialize $\Gamma_0^{a_{1:d}} = \{\alpha = (\frac{R_{\min}}{1-\gamma}) \cdot \vec{1}\}$ for all $a_{1:d} \in A^d$ so that every $\Gamma_0^{a_{1:d}}$ is initialized to be a singleton with a constant vector. This is the standard way of initializing α-vectors in PBVI and other POMDP algorithms.

For $n \geq 1$, suppose that the theorem holds for α-vector sets after the $(n-1)$th value backup. We also have that every α-vector $\alpha_n^{a_{1:d},a} \in \Gamma_n^{a_{1:d},a}$ is computed by the equation

$$\alpha_n^{a_{1:d},a}(s_0) = \sum_{s_{1:d}} \Big[\prod_{i=0}^{d-1} T_{s_i,s_{i+1}}^{a_{i+1}} \Big] R_{s_d}^a + \gamma \sum_{s_1} T_{s_0,s_1}^{a_1} \sum_z O_{s_1,a_1}^z \alpha_{n-1,z}^{a_{2:d},a}(s_1)$$

where the α-vector $\alpha_{n-1,z}^{a_{2:d},a}$ for each observation z comes from the set $\Gamma_{n-1}^{a_{2:d},a}$. Hence, we can show that

$$\begin{aligned}
&\alpha_n^{a_{1:d},a}(s_0)\\
&=\sum_{s_{1:d}}\Big[\prod_{i=0}^{d-1} T_{\pi(s_i),\pi(s_{i+1})}^{\rho(a_{i+1})}\Big] R_{\pi(s_d)}^{\rho(a)}\\
&\qquad+\gamma\sum_{s_1} T_{\pi(s_0),\pi(s_1)}^{\rho(a_1)} \sum_z O_{\pi(s_1),\rho(a_1)}^{\tau(z)} \alpha_{n-1,z}^{a_{2:d},a}(s_1)\\
&=\sum_{s_{1:d}}\Big[\prod_{i=0}^{d-1} T_{\pi(s_i),\pi(s_{i+1})}^{\rho(a_{i+1})}\Big] R_{\pi(s_d)}^{\rho(a)}\\
&\qquad+\gamma\sum_{s_1} T_{\pi(s_0),\pi(s_1)}^{\rho(a_1)} \sum_z O_{\pi(s_1),\rho(a_1)}^{\tau(z)} \alpha_{n-1,z}^{\rho(a_{2:d}),\rho(a)}(\pi(s_1))\\
&=\alpha_n^{\rho(a_{1:d}),\rho(a)}(\pi(s_0)).
\end{aligned}$$

The first equality is obtained after applying the conditions for the permutability. The second equality follows from the inductive assumption. The last equality follows from the definition. Therefore, for every α-vector $\alpha \in \Gamma_n^{a_{1:d}}$, there exists $\alpha^\pi \in \Gamma_n^{\rho(a_{1:d})}$ because $\Gamma_n^{a_{1:d}} = \bigcup_a \Gamma_n^{a_{1:d},a}$. By mathematical induction, this theorem holds. ■

Theorem 1 implies an important property that, if there exists at least one permutation π satisfying all conditions on theorem 1, we only need to maintain either $\Gamma_n^{a_{1:d}}$ or $\Gamma_n^{\rho(a_{1:d})}$ because we can reproduce one α-vector set from the other set. In other words, given belief b and actions $\rho(a_{1:d})$ during the delay, $V_n(b,\rho(a_{1:d}))$ can be computed by using the set $\Gamma_n^{a_{1:d}} = \{\alpha^\pi | \alpha \in \Gamma_n^{a_{1:d}}\}$. Note that this explicit reproduction of the set by permuting every α-vector is not necessary. We can compute the value by

$$V_n(b,\rho(a_{1:d})) = \max_{\alpha\in\Gamma_n^{a_{1:d}}} b^{\pi^{-1}}\cdot\alpha$$

where $b^{\pi^{-1}}$ is the reordering of belief b using the inverse mapping of permutation π.

Corollary 2: Let A^d be the set of all action sequences that can be executed during the observation delay and $\sim$ be a relation where $a_{1:d} \sim a'_{1:d}$ if and only if there exists a state permutation π with corresponding ρ and τ that satisfy the conditions in Theorem 1 and $\rho(a_{1:d}) = a'_{1:d}$. Then A^d can be partitioned into equivalence classes induced by the relation $\sim$.

Proof: We show that $\sim$ is an equivalence relation by showing its reflexivity, symmetry, and transitivity. Reflexivity: the identity mapping for π, ρ, and τ satisfies all the conditions in Theorem 1. Symmetry: there always exists the inverse mapping of π, ρ, and τ since they are permutations and hence one-to-one and onto functions. Transitivity: suppose that $a_{1:d} \sim a'_{1:d}$ and $a'_{1:d} \sim a''_{1:d}$. This implies that there exist $\langle\pi,\rho,\tau\rangle$ with $\rho(a_{1:d}) = a'_{1:d}$ and $\langle\pi',\rho',\tau'\rangle$ with $\rho'(a'_{1:d}) = a''_{1:d}$ that satisfy the conditions in Theorem 1.

Let $\pi'' = \pi' \circ \pi$, $\rho'' = \rho' \circ \rho$, and $\tau'' = \tau' \circ \tau$. It is quite straightforward to show that π'', ρ'', and τ'' satisfy the conditions in Theorem 1 with $\rho''(a_{1:d}) = a''_{1:d}$. ■

Corollary 2 implies that we only need to maintain one α-vector set for all the α-vector sets whose action sequences during observation delay are within the same equivalence class.

APPENDIX C
PERMUTABLE POMDP ALGORITHM FOR P300 BCIS

Now consider the POMDP model for the P300 BCI where the observation delay is 2 time steps. Since each action in the POMDP model is relevant to only one state, a state permutation readily yields the corresponding action permutation. For example, given action $a = \mathrm{flash}_{\mathrm{row}\#1}$, state permutation π maps the action to $a^{\pi} = \mathrm{flash}_{\pi(\mathrm{row}\#1)}$. We specifically consider the following set of permutations:

Theorem 3: Let $\Psi = \{\langle \pi, \rho, \tau \rangle |$ for any $\pi, \rho(a) = a^{\pi}, \forall a \in A, \tau(z) = z, \forall z \in Z\}$. $\langle \pi, \rho, \tau \rangle \in \Psi$ satisfies all the conditions in Theorem 1.

Proof: $\langle \pi, \rho, \tau \rangle \in \Psi$ does not change the transition probabilities and rewards, since ρ does not change the type of the action between flash and select. It also does not change the observation probabilities, since π and ρ do not change the action between target flash and non-target flash, and observations do not change via τ. ■

A minor note about the above theorem is that Ψ can be a subset of all the permutations that satisfy the conditions in Theorem 1. However, Ψ is straightforward to obtain without using an algorithm [19], and we found it to be sufficient to speed up the POMDP algorithm.

Using permutations in Ψ, the set of all action sequences during the delay (a total of 144 action sequences) is partitioned into equivalence classes by Corollary 2. We now need to maintain only one α-vector set for each equivalence class. There are 8 equivalence classes: $(\mathrm{flash}_{\mathrm{x}}, \mathrm{flash}_{\mathrm{x}})$, $(\mathrm{flash}_{\mathrm{x}}, \mathrm{flash}_{\mathrm{y}})$, $(\mathrm{select}_{\mathrm{x}}, \mathrm{flash}_{\mathrm{x}})$, $(\mathrm{select}_{\mathrm{x}}, \mathrm{flash}_{\mathrm{y}})$, $(\mathrm{flash}_{\mathrm{x}}, \mathrm{select}_{\mathrm{x}})$, $(\mathrm{flash}_{\mathrm{x}}, \mathrm{select}_{\mathrm{y}})$, $(\mathrm{select}_{\mathrm{x}}, \mathrm{select}_{\mathrm{x}})$, $(\mathrm{select}_{\mathrm{x}}, \mathrm{select}_{\mathrm{y}})$ where $x, y \in S$ and $x \neq y$. Therefore we have only to maintain 8 α-vector sets for representing the whole 144 α-vector sets. For each equivalence class, we used the α-vector set with the lexicographic minimum, e.g. $\Gamma^{\mathrm{flash}_{\mathrm{row}\#1}, \mathrm{flash}_{\mathrm{row}\#2}}$ for the equivalence class $(\mathrm{flash}_{\mathrm{x}}, \mathrm{flash}_{\mathrm{y}})$ and the corresponding action sequence is referred to as the representative action sequence of the equivalence class, e.g. $\mathrm{flash}_{\mathrm{row}\#1}, \mathrm{flash}_{\mathrm{row}\#2}$ for $(\mathrm{flash}_{\mathrm{x}}, \mathrm{flash}_{\mathrm{y}})$. To compute value $V_n(b, a_{1:d})$, we identify $\langle \pi^{\mathrm{eq}}, \rho^{\mathrm{eq}}, \tau^{\mathrm{eq}} \rangle \in \Psi$ such that $\rho^{\mathrm{eq}}(a_{1:d}) = a'_{1:d}$ where $a'_{1:d}$ is the representative action sequence of the equivalence class of $a_{1:d}$. There are more than one such permutations, and we only have to identify one. We then have

$$V_n(b, a_{1:d}) = \max_{\alpha \in \Gamma_n^{a'_{1:d}}} b^{\pi^{\text{eq}}} \cdot \alpha.$$

We can further reduce the sizes of α-vector sets $\Gamma_n^{a_{1:d}}$'s using a technique similar to permutable POMDPs [18]: Let $\Psi^{a_{1:d}} \subseteq \Psi$ be the set of all permutations such that $a_{1:d} \sim a_{1:d}$. According to Theorem 3 and 1, for any $\langle \pi, \rho, \tau \rangle \in \Psi^{a_{1:d}}$ and $\alpha \in \Gamma_n^{a_{1:d}}$, we have $\alpha^{\pi} \in \Gamma_n^{a_{1:d}}$ as well. Specifically, let $S^{-a_{1:d}}$ be the set of states irrelevant to actions in $a_{1:d}$. For example, $S^{-\{\text{flash}_{\text{row}\#1}, \text{flash}_{\text{row}\#2}\}} = \{\text{row}\#3, \text{row}\#4, \text{row}\#5, \text{row}\#6\}$ since the actions are only relevant to row#1 and row#2. Note that $\Psi^{a_{1:d}}$ is the set of all permutations in Ψ that can only possibly change the state in $S^{-a_{1:d}}$. This is because, for $\langle \pi, \rho, \tau \rangle \in \Psi$, if π changes states relevant to actions in $a_{1:d}$, then $\langle \pi, \rho, \tau \rangle \notin \Psi^{a_{1:d}}$ since $a_{1:d} \neq \rho(a_{1:d})$. Hence, if α is in $\Gamma_n^{a_{1:d}}$, then all its possible permutations on the states $S^{-a_{1:d}}$ are also in $\Gamma_n^{a_{1:d}}$. Therefore, rather than computing and storing all such α-vectors in $\Gamma_n^{a_{1:d}}$, we maintain a single α-vector that represents the rest, which is obtained by sorting on the states $S^{-a_{1:d}}$ in the descending order. This technique reduces the size of $\Gamma_n^{a_{1:d}}$ by a factor of $|\Psi^{a_{1:d}}|$, i.e. $|\Gamma_{n,\text{sort}}^{a_{1:d}}| = |\Gamma_n^{a_{1:d}}|/|\Psi^{a_{1:d}}|$. Using the parsimonious set $\Gamma_{n,\text{sort}}^{a_{1:d}}$, we can compute the value by

$$V_n(b, a_{1:d}) = \max_{\alpha \in \Gamma_{n,\text{sort}}^{a_{1:d}}} b_{\text{sort}} \cdot \alpha$$

where b_{sort} is the belief sorted on the states $S^{-a_{1:d}}$ in the descending order.

Combining the two techniques is quite straightforward. We only maintain α-vector sets for representative action sequences and each representative α-vector set maintains α-vectors that are sorted on the state $S^{-a_{1:d}}$. Given belief b and actions $a_{1:d}$ during the delay, we compute $V_n(b, a_{1:d})$ by the following method: We first identify $\langle \pi^{\text{eq}}, \rho^{\text{eq}}, \tau^{\text{eq}} \rangle \in \Psi$ of which ρ^{eq} maps $a_{1:d}$ to the representative action sequence in its equivalence class. We then identify $\langle \pi^{\text{sort}}, \rho^{\text{sort}}, \tau^{\text{sort}} \rangle \in \Psi^{\rho^{\text{eq}}(a_{1:d})}$ by sorting $b^{\pi^{\text{eq}}}$ on the states $S^{-\rho^{\text{eq}}(a_{1:d})}$ in the descending order. We then have

$$V_n(b, a_{1:d}) = \max_{\alpha \in \Gamma_{n,\text{sort}}^{\rho^{\text{eq}}(a_{1:d})}} b^{(\pi^{\text{sort}} \circ \pi^{\text{eq}})} \cdot \alpha \quad ,$$

and the α-vector that gives the maximum value can be obtained by

$$\alpha^* = \left[\operatorname*{argmax}_{\alpha \in \Gamma_{n,\text{sort}}^{\rho^{\text{eq}}(a_{1:d})}} b^{(\pi^{\text{sort}} \circ \pi^{\text{eq}})} \cdot \alpha \right]^{(\pi^{\text{sort}} \circ \pi^{\text{eq}})^{-1}} .$$

Note that the identifications of the permutations can be done implicitly without enumerating and examining all the permutations in Ψ and $\Psi^{a_{1:d}}$. For example, if $a_{1:d} = (\text{flash}_{\text{row\#5}}, \text{flash}_{\text{row\#3}})$, we can easily identify that $\rho^{\text{eq}}(a_{1:d}) = (\text{flash}_{\text{row\#1}}, \text{flash}_{\text{row\#2}})$ since we have to map the action sequence to its lexicographic minimum. We then map the states $\pi(\text{row\#5}) = \text{row\#1}$ and $\pi(\text{row\#3}) = \text{row\#2}$, and sort b on the rest of the states in the descending order, which results in $b^{(\pi^{\text{sort}} \circ \pi^{\text{eq}})}$, and evaluate its inner-product with the α-vectors in $\Gamma_{n,\text{sort}}^{\rho^{\text{eq}}(a_{1:d})}$ to compute the value.

Figures 3, 4, and 5 show the pseudocode of the PBVI algorithm for finding an optimal policy of the POMDP model for P300 BCI. $B^{a_{1:d}}$ in the pseudocode represents the set of belief states used for the point-based backup where the action sequence executed during the delay was $a_{1:d}$. The belief states were collected using the breadth-first traversal starting from b_0. However, we do not exhaustively collect all the reachable belief states since there can be an infinite number of them - we stop the traversal when we have collected a prescribed number of belief states. In addition, when we add a belief state b into the set, we instead add b_{sort} which is sorted on the states $S^{-a_{1:d}}$, following the idea discussed in the previous paragraph.

The algorithm produces the set $\Gamma^{a_{1:d}}$ of α-vectors that represents the optimal value function for each representative action sequence $a_{1:d}$. Each α-vector is associated with the optimal action to execute, which is a^* in Fig. 4. Hence, the execution of the optimal policy is carried out in a similar manner as the optimal value function is computed, using the equation

$$V^*(b, a_{1:d}) = \max_{\alpha \in \Gamma^{a_{1:d}}} b \cdot \alpha. \tag{5}$$

and selecting the action associated with the best α-vector.

Note that when we execute the policy, the action sequence during the delay is of length less than d for the initial d time steps. In this case, we maximize over the action sequences that agree with the actions that are executed so far. For example, when d=2 and only one action a was executed so far, we use the value

$$V^*(b, a) = \max_{a'} \sum_{s_0, s_1} b(s_0) \left[T^a_{s_0,s_1} R^{a'}_{s_1} \right] + V^*(b, a, a'). \tag{6}$$

When no action was executed so far, we use

$$V^*(b) = \max_{a'} \sum_{s_0} b(s_0) R^{a'}_{s_0} + V^*(b, a'). \tag{7}$$

Appendix D
Execution of POMDP Flash Strategies

We briefly explain how we execute the π_{POMDP} using an example. Suppose that the first target letter is M which is in row#3 (see Fig. 1). We start the row flash strategy with

$$b_0 = [0.1667, 0.1667, 0.1667, 0.1667, 0.1667, 0.1667]$$

Using Eqn. 7, we determine the action that maximizes the value function for b_0. Suppose that the action is $a_0^* =$ flash$_{\text{row\#3}}$. We thus flash row#3 and move on to the next flash epoch $t = 1$. Since the P300 detection result of a_0^* is not available until $t = 3$, the belief state remains the same:

$$b_1 = [0.1667, 0.1667, 0.1667, 0.1667, 0.1667, 0.1667]$$

Using Eqn. 6, we determine the action that maximizes the value function for b_1. Suppose that the action is $a_1^* =$ flash$_{\text{row\#6}}$. We thus flash row#6 and move on to the next flash epoch $t = 2$. Since the first P300 detection result is still not available, the belief state remains the same:

$$b_2 = [0.1667, 0.1667, 0.1667, 0.1667, 0.1667, 0.1667]$$

Using Eqn. 5, suppose that the action that maximizes the value function for b_2 is $a_2^* =$ flash$_{\text{row\#1}}$. We thus flash row#1 and move on to the next flash epoch $t = 3$. The P300 detection result for flashing row#3 at $t = 0$ is now available, and suppose that it is 0.9991 (z_{10}). We update the belief state using Eqn. 1 so that

$$b_3 = [0.0669, 0.0669, 0.6654, 0.0669, 0.0669, 0.0669]$$

Note that the probability of row#3 being the target row is increased by the detection result, although the exact numerical values may vary depending on the observation probabilities of the POMDP obtained from the subject. Suppose that the action maximizes the value function for b_3 is $a_3^* =$ flash$_{\text{row\#3}}$. We thus flash row#3 and move on to the next flash epoch $t = 4$. The P300 detection result for flashing row#6 at $t = 1$ is now available, and suppose that it is 0.002087 (z_1). The updated belief state becomes

$$b_4 = [0.0712, 0.0712, 0.7082, 0.0712, 0.0712, 0.0069]$$

Note that the detection result effectively lowers the probability of row#6 being the target row, while increasing other rows by a small amount.

We continue alternating between updating the belief state using Eqn. 1 and executing the best action using Eqn. 5. If the best action is one of the select actions, we terminate the flash strategy by selecting the corresponding row as the target row, and then start the column flash strategy in a similar manner.

ACKNOWLEDGMENT

The authors would like to thank Sungho Jo and Yoon-Kyu Song for their helpful comments regarding the early version of the work. This work was supported by National Research Foundation of Korea (NRF) grant 2009-0069702 and KAIST-Microsoft Research Collaboration Center.

REFERENCES

[1] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan, “Brain-computer interfaces for communication and control,” *Clinical Neurophysiology*, vol. 113, pp. 767–791, 2002.

[2] S. G. Mason, A. Bashashati, M. Fatourechi, K. F. Navarro, and G. E. Birch, “A comprehensive survey of brain interface technology designs,” *Annals of Biomedical Engineering*, vol. 35, pp. 137–169, 2007.

[3] D. J. Krusienski, E. W. Sellers, D. J. McFarland, T. M. Vaughan, and J. R. Wolpaw, “Toward enhanced P300 speller performance,” *Journal of Neuroscience Methods*, vol. 167, pp. 15–21, 2008.

[4] L. A. Farwell and E. Donchin, “Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials,” *Electroencephalography and Clinical Neurophysiology*, vol. 70, pp. 510–523, 1988.

[5] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, “Planning and acting in partially observable stochastic domains,” *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[6] J. Park, K.-E. Kim, and S. Jo, “A POMDP approach to P300-based brain-computer interfaces,” in *Proceedings of the ACM International Conference on Intelligent User Interfaces (IUI)*, 2010, pp. 1–10.

[7] J. Park, K.-E. Kim, and Y.-K. Song, “A POMDP-based optimal control of P300-based brain-computer interfaces,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), NECTAR Track*, 2011, pp. 1559–1562.

[8] U. Hoffmann, J.-M. Vesin, and T. Ebrahimi, “Spatial filters for the classification of event-related potentials,” in *Proceedings of the European Symposium on Artificial Neural Networks*, 2006, pp. 47–52.

[9] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: a library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[10] J. Pineau, G. Gordon, and S. Thrun, “Anytime point-based approximations for large POMDPs,” *Journal of Artificial Intelligence Research*, vol. 27, pp. 335–380, 2006.

[11] T. Smith and R. Simmons, “Heuristic search value iteration for POMDPs,” in *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004, pp. 520–527.

[12] ——, “Point-based pomdp algorithms: Improved analysis and implementations,” in *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005, pp. 542–459.

[13] E. Charniak, *Statistical Language Learning*. MIT Press, 1996.

[14] W. Min and G. Luo, “Medical applications of EEG wave classification,” *Chance*, vol. 22, no. 4, 2009.

[15] J. L. Bander and C. C. W. III, “Markov decision processes with noise-corrupted and delayed state observations,” *Journal of the Operational Research Society*, vol. 50, pp. 660–668, 1999.

[16] R. Fazel-Rezai, “Human error in p300 speller paradigm for brain-computer interface,” in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2007, pp. 2516–2519.

[17] N. G. Kanwisher, “Repetition blindness: Type recognition without token individuation,” *Cognition*, vol. 27, pp. 117–143, 1987.

[18] F. Doshi and N. Roy, “The permutable POMDP: fast solutions for POMDPs for preference elicitation,” in *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008, pp. 493–500.

[19] K.-E. Kim, “Exploiting symmetries in POMDPs for point-based algorithms,” in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2008, pp. 1043–1048.

[20] H. Serby, E. Yom-Tov, and G. F. Inbar, “An improved P300-based brain-computer interface,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 13, pp. 89–98, 2005.

[21] J. R. Wolpaw *et al.*, “Brain-computer interface technology: A review of the first international meeting,” *IEEE Transactions on Rehabilitation Engineering*, vol. 8, pp. 164–173, 2000.

[22] G. Townsend *et al.*, “A novel P300-based brain-computer interface stimulus presentation paradigm: Moving beyond rows and columns,” *Clinical Neurophysiology*, vol. 121, pp. 1109–1120, 2010.

[23] G. Luo and W. Min, “Distance-constrained orthogonal latin squares for brain-computer interface,” *Journal of Medical Systems*, pp. 1–8, 10.1007/s10916-010-9455-6. [Online]. Available: http://dx.doi.org/10.1007/s10916-010-9455-6

[24] J. Hill, J. Farquhar, S. Martens, F. Bießmann, and B. Schölkopf, “Effects of stimulus type and of error-correcting code design on BCI speller performance,” in *Proceedings of the Neural Information Processing Systems (NIPS)*, 2009, pp. 665–672.

[25] J. Hoey and P. Poupart, “Solving POMDPs with continuous or large discrete observation spaces,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 1332–1338.

Jaeyoung Park received his BS degree with double major in Computer Science and Mathematical Sciences from KAIST, Korea, in 2008, and MS degree in Computer Science from KAIST in 2010. He is currently a PhD candidate in Computer Science at KAIST. His primary research interest is in the application of decision-theoretic algorithms to brain-computer interfaces.

Kee-Eung Kim received the BS degree in Computer Science from KAIST, Korea, in 1995, and the ScM and PhD degrees in Computer Science from Brown University, USA, in 1998 and 2001, respectively. From 2001 to 2006, he was with Samsung SDS and Samsung advanced institute of technology, Korea. In 2006, he joined the Computer Science Department at KAIST, where he is currently an associate professor. His research interests are representations and algorithms for sequential decision making problems in artificial intelligence and machine learning.

List of Figures

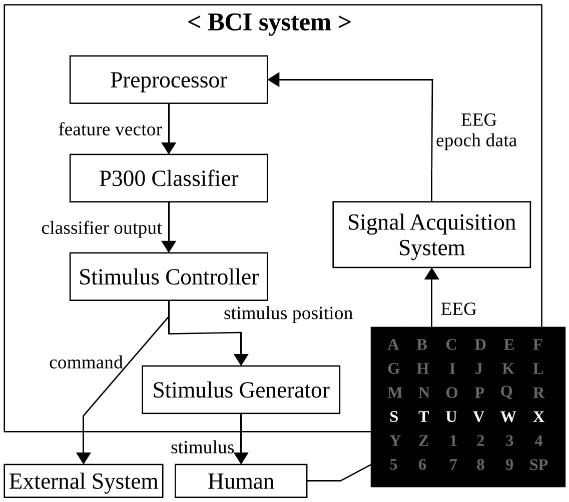
< BCI system >
Preprocessor
feature vector
P300 Classifier
classifier output
Stimulus Controller
stimulus position
command
Stimulus Generator
stimulus
External System
Human
EEG
epoch data
Signal Acquisition System
EEG
A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Y Z 1 2 3 4
5 6 7 8 9 SP

Fig. 1

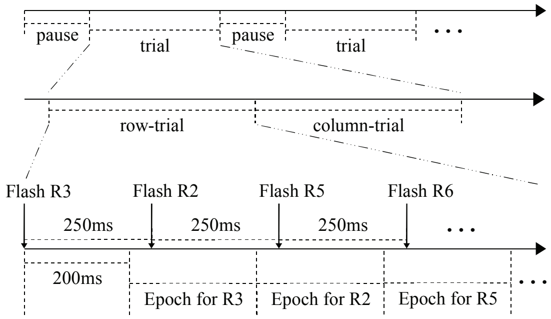
pause
trial
pause
trial
. . .
row-trial
column-trial
Flash R3
Flash R2
Flash R5
Flash R6
250ms
250ms
250ms
. . .
200ms
Epoch for R3
Epoch for R2
Epoch for R5
. . .

Fig. 2

```
Procedure: PERMUTABLE-BCI-PBVI($B$, $K$, $d$, $\epsilon$)
  Inputs: the set of belief state sets $B = \{B^{a_{1:d}} | \text{representative action sequence } a_{1:d}\}$; repetition blindness length $K$; observation delay $d$; error bound $\epsilon$
  for all representative action sequence $a_{1:d}$ do
    Initialize $\Gamma^{a_{1:d}} = \{\left(\frac{R_{\text{min}}}{1-\gamma}\right) \cdot \vec{1}\}$
  end for
  repeat
    for all representative action sequence $a_{1:d}$ do
      $\Gamma^{a_{1:d}}_{\text{next}} = \text{BACKUP}(\langle a_{1:d} \rangle, K, d, B^{a_{1:d}}, \Gamma)$
    end for
    $\delta = \text{difference}(B, \Gamma, \Gamma_{\text{next}})$
    $\Gamma = \Gamma_{\text{next}}$
  until $\delta < \epsilon$
  return $\Gamma$
```

Fig. 3

```
Procedure: BACKUP($(a_{1:d}, K, d, B^{a_{1:d}}, \Gamma_{n-1})$)
  Inputs: the action sequence $a_{1:d}$; repetition blindness length $K$; observation delay length $d$; belief state set $B^{a_{1:d}}$; set of $\alpha$-vector sets for ($n$-1)-step value function $\Gamma_{n-1} = \{\Gamma_{n-1}^{a_{1:d}} |$representative action sequence $a_{1:d}\}$
  $A_{\text{allow}} = A - \{a_{d-K+1:d}\}$ //avoid repetition blindness
  for all belief state $b \in B^{a_{1:d}}$ do
    for all action $a \in A_{\text{allow}}$ do
      $\alpha^a = \vec{0}$
      for all observation $z \in Z$ do
        $b' = \tau(b, a_1, z)$
        $\alpha^{a,z} =$ FIND_MAX_ALPHA$(b', a_{2:d}; a, \Gamma_{n-1})$
        for all state $s \in S$ do
          $\alpha^a(s) = \alpha^a(s) + \gamma \sum_{s' \in S} T^a_{s,s'} O^z_{s',a} \alpha^{a,z}(s')$
        end for
      end for
      for all state $s_0 \in S$ do
        $\alpha^a(s_0) = \alpha^a(s_0) + \sum_{s_{1:d}} \left( \prod_{i=0}^{d-1} T^{a_{i+1}}_{s_i,s_{i+1}} \right) R^a_{s_d}$
      end for
    end for
    $a^* = \text{argmax}_{a \in A_{\text{allow}}} b \cdot \alpha^a$
    $\alpha^* = \alpha^{a^*}$
    $\alpha_{\text{sort}} =$ sort $\alpha^*$ on states $S^{-a_{1:d}}$ in descending order
    $\Gamma_n^{a_{1:d}} = \Gamma_n^{a_{1:d}} \cup \{\alpha_{\text{sort}}\}$
  end for
  return $\Gamma_n^{a_{1:d}}$
```

Fig. 4

```
Procedure: FIND_MAX_ALPHA$(b, a_{1:d}, \Gamma_{n-1})$
    Inputs: belief state $b \in B^{a_{1:d}}$; action sequence $a_{1:d}$; set of $\alpha$-vector sets for ($n$-1)-step value function $\Gamma_{n-1} = \{\Gamma_{n-1}^{a_{1:d}} | \text{representative action sequence } a_{1:d}\}$
    $a'_{1:d}$ = representative action sequence of $a_{1:d}$
    compute $\langle \pi^{\text{eq}}, \rho^{\text{eq}}, \tau^{\text{eq}} \rangle \in \Psi$, and $\langle \pi^{\text{sort}}, \rho^{\text{sort}}, \tau^{\text{sort}} \rangle \in \Psi^{a_{1:d}}$ such that
    (1) $\rho^{\text{eq}}(a_{1:d}) \sim a'_{1:d}$, and
    (2) $\pi^{\text{sort}}$ is sorting $b^{\pi^{\text{eq}}}$ on states $S^{-a'_{1:d}}$ in descending order
    $\alpha^{a'_{1:d}} = \text{argmax}_{\alpha}\, b^{(\pi^{\text{sort}} \circ \pi^{\text{eq}})} \cdot \alpha$ where $\alpha \in \Gamma_{n-1}^{a'_{1:d}}$
    $\alpha^{a_{1:d}} = \alpha^{a'_{1:d}}{}^{(\pi^{\text{sort}} \circ \pi^{\text{eq}})^{-1}}$
    return $\alpha^{a_{1:d}}$
```

Fig. 5

LIST OF TABLES

TABLE I

Subject	π_{RAND}				π_{POMDP}				$\pi_{\text{POMDP-BIGRAM}}$			
	PBR	BR	Acc	Flashes	PBR	BR	Acc	Flashes	PBR	BR	Acc	Flashes
A	36.483	36.483	100.00	24	38.965	38.965	100.00	21.9±4.6	41.751	41.751	100.00	19.8±6.1
B	9.541	9.541	100.00	120	26.452	26.452	100.00	36.9±13.8	31.437	31.437	100.00	29.5±7.9
C	4.051	6.481	81.25	120	6.718	8.957	87.50	96.8±26.3	10.246	10.246	100.00	111.1±54.3
D	0.447	3.575	56.25	120	5.736	9.177	81.25	81.9±24.7	12.912	12.912	100.00	86.1±72.4
E	6.765	9.020	87.50	96	6.136	8.181	87.50	106.9±51.0	8.201	9.373	93.75	105.6±70.0
F	17.720	17.720	100.00	60	25.833	25.833	100.00	38.1±9.6	32.359	32.359	100.00	28.4±5.8
G	1.057	4.230	62.50	120	2.116	4.232	75.00	164.5±108.1	2.551	4.081	81.25	196.6±68.0
H	9.080	14.527	81.25	48	12.380	14.149	93.75	66.6±17.4	13.713	15.672	93.75	59.1±22.7
I	16.331	18.664	93.75	48	27.212	27.212	100.00	35.6±12.8	30.851	30.851	100.00	30.3±14.7
J	20.592	23.533	93.75	36	38.288	38.288	100.00	22.4±4.8	43.873	43.873	100.00	18.3±7.5
Avg.	12.206	14.377	85.63	-	18.984	20.145	92.50	-	22.790	23.256	96.88	-
s.d.	10.968	10.120	15.60	-	13.990	12.838	0.09	-	14.889	14.410	0.06	-