35th Annual International Conference of the IEEE EMBS
Osaka, Japan, 3 - 7 July, 2013

Likelihood Gradient Ascent (LGA): A Closed-Loop Decoder Adaptation Algorithm for Brain-Machine Interfaces

Siddharth Dangi[1†], *Student Member, IEEE*, Suraj Gowda[1†], *Student Member, IEEE*,
and Jose M. Carmena[1, 2], *Senior Member, IEEE*,

***Abstract*— Closed-loop decoder adaptation (CLDA) is an emerging paradigm for improving or maintaining the online performance of brain-machine interfaces (BMIs). Here, we present Likelihood Gradient Ascent (LGA), a novel CLDA algorithm for a Kalman filter (KF) decoder that uses stochastic, gradient-based corrections to update KF parameters during closed-loop BMI operation. LGA's gradient-based paradigm presents a variety of potential advantages over other "batch" CLDA methods, including the ability to update decoder parameters on any time-scale, even on every decoder iteration. Using a closed-loop BMI simulator, we compare the LGA algorithm to the Adaptive Kalman Filter (AKF), a partially gradient-based CLDA algorithm that has been previously tested in non-human primate experiments. In contrast to the AKF's separate mean-squared error objective functions, LGA's update rules are derived directly from a single log likelihood objective, making it one step towards a potentially optimal continuously adaptive CLDA algorithm for BMIs.**

I. INTRODUCTION

Brain-Machine Interfaces (BMIs) aim to restore motor function to those suffering from stroke, spinal cord injury, amyotrophic lateral sclerosis, and other disabling conditions. Researchers have exhibited compelling proof-of-concept BMI demonstrations of rodents, monkeys, and humans using neural activity to control both real and artificial actuators [1]–[3]. However, significant improvements in both reliability and performance are still needed to achieve clinically viable neuroprostheses for humans [4].

Closed-loop decoder adaptation (CLDA) is one mechanism for achieving and/or maintaining high performance BMI systems. CLDA algorithms aim to make the decoder's output accurately reflect the user's intended movements by adapting or updating the decoder's parameters during closed-loop BMI operation (i.e., while the subject is using the BMI). A variety of CLDA algorithms have been developed that operate on different types of BMI decoders, including the population vector algorithm [1], the Wiener filter [5], neural networks [6] and the Kalman filter [7]–[10]. Here we focus our analysis on CLDA for the purpose of discovering high-performance decoder parameter settings without prior knowledge of neural tuning properties.

One CLDA paradigm that has not yet been fully explored is that of continuous adaptation. Previous work developed a continuously adaptive, partially gradient-based CLDA algorithm known as the Adaptive Kalman Filter (AKF) and tested it in non-human primate experiments [10], [11]. Here, we introduce Likelihood Gradient Ascent (LGA), a novel KF CLDA algorithm whose gradient-based paradigm presents potential advantages over other methods. Using a closed-loop BMI simulator, we demonstrate that applying the LGA algorithm leads to higher performance than the AKF in a 2-D center-out cursor control task. In contrast to the AKF, LGA's update rules are derived directly from a single log likelihood objective function rather than separate mean-squared error objectives, making LGA a step towards a potentially optimal continuously adaptive CLDA algorithm for BMIs.

II. THE LIKELIHOOD GRADIENT ASCENT (LGA) ALGORITHM

Closed-loop decoder adaptation (CLDA) is an emerging paradigm for improving or maintaining the online performance of brain-machine interfaces (BMIs). Here, we propose a new CLDA algorithm called Likelihood Gradient Ascent (LGA) for adapting the parameters of a Kalman filter (KF) decoder during closed-loop BMI operation. To be concrete, we consider the example case of BMI cursor control, where the KF state represents the position and velocity (in the horizontal and vertical directions) of a computer cursor. It should be noted, however, that the LGA algorithm and the KF itself still apply more generally to any BMI prosthetic device. First, we briefly review the Kalman filter model and different methods for estimating the user's intended movements that could be used as part of LGA.

A. The Kalman filter

Let $x_n \in \mathbb{R}^k$ and $y_n \in \mathbb{R}^m$ represent a hidden state and a noisy observation of that state, respectively, at iteration n. In the BMI context, for example, x_n represents the user's intended movements of a prosthetic device (e.g., position and velocity of a computer cursor), while y_n represents features extracted from the user's neural activity (e.g., binned firing rates). The Kalman filter assumes the following state transition and state observation models:

$$x_n = Ax_{n-1} + w_n \quad (1)$$

$$y_n = Cx_n + q_n \quad (2)$$

†S. Dangi and S. Gowda contributed equally to this work.
[1]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720 USA
[2]Helen Wills Neuroscience Institute, University of California, Berkeley, CA 94720 USA
{sdangi,sgowda,carmena}@eecs.berkeley.edu
This work was supported by the Defense Advanced Research Projects Agency contract N66001-10-C-2008 (JMC) and the National Science Foundation grant CBET-0954243 (JMC).

where $w_n \sim \mathcal{N}(0, W)$, $q_n \sim \mathcal{N}(0, Q)$, and $\{w_n\}$ and $\{q_n\}$ are each independent and identically distributed over time (and independent of each other). From these equations, we note that the KF model is parametrized by the matrices $\{A, W, C, Q\}$. Under these model assumptions, the Kalman filter is a statistically optimal algorithm for estimating the hidden state as it evolves over time when given only the noisy observations of the state. Using knowledge of the underlying model parameters $\{A, W, C, Q\}$, the Kalman filter implements an optimal linear, recursive algorithm for estimating the unknown intended movement states $\{x_n\}$ from the neural observations $\{y_n\}$. We refer the reader to [12] for the actual KF equations used to estimate the state x_t at each filter iteration.

B. Estimating intended movements

When using a KF decoder for BMI cursor control, the KF outputs a sequence $\{\hat{x}_n\}$ that represents the decoded cursor kinematics over time. Unless the decoder's parameters are already optimized or the BMI user has already learned to use the decoder for accurate cursor control, the user will likely make movement errors when attempting to control the cursor. For instance, if the user is attempting to type on a virtual keyboard, the cursor may not always move towards the letter that the user intends.

During a training or calibration phase, one could use additional task knowledge (e.g., the sequence of letters that the user intended to type) to generate an estimate of the user's intended cursor kinematics over time, which we will denote as $\{\tilde{x}_n\}$. For instance, one might utilize Shpigelman et al.'s method, which would estimate intended cursor trajectories using a time-varying linear combination of $\{\hat{x}_n\}$ and the known letter locations [13]. Alternatively, one could use Gilja et al.'s method ("innovation 1" of the ReFIT-KF algorithm), which assumes that the user always intends to move the cursor in a straight line towards the next letter [7]. Gilja et al.'s method has previously been used to estimate intended cursor kinematics as part of other CLDA algorithms [7], [8]. Other methods of estimating the user's intended movements have also been developed, including Li et al.'s unsupervised method of performing Kalman smoothing to "correct" previously decoded outputs [9].

Given an estimate $\{\tilde{x}_n\}$ of the user's intended cursor kinematics and simultaneously recorded neural features $\{y_n\}$, one can feed this data into a CLDA algorithm (such as LGA). The CLDA algorithm can then update the KF decoder's parameters during closed-loop BMI operation, with the goal of making the decoder's future outputs more accurately reflect the user's intended movements.

C. The LGA algorithm

Here, we introduce Likelihood Gradient Ascent (LGA), a new gradient-based CLDA algorithm for a KF decoder. Let $\theta^{(i)} = \left\{A^{(i)}, W^{(i)}, C^{(i)}, Q^{(i)}\right\}$ represent the current KF decoder parameters. The LGA algorithm first collects a batch of N consecutive pairs of $(\tilde{x}_n, y_n)$, where $\tilde{x}_n$ and y_n are intended cursor kinematics and neural observations (respectively), and $N \geq 1$ is a free algorithmic parameter which we denote as the "batch size". Let $\mathcal{D}$ represent the collection of these N pairs of data. The log likelihood of a set of parameters θ given $\mathcal{D}$ — $l(\theta; \mathcal{D})$ — is defined as the log probability of observing the data $\mathcal{D}$, under the assumption that this data originated from a model parametrized by θ. In order to update the current KF parameter setting $\theta^{(i)}$, the LGA algorithm calculates the gradients of $l(\theta; \mathcal{D})$ with respect to each of $\{A, W, C, Q\}$, evaluates these gradients at the current parameter values $\left\{A^{(i)}, W^{(i)}, C^{(i)}, Q^{(i)}\right\}$, and then updates these parameter values using one step of gradient ascent. For example, C is updated by the rule

$$C^{(i+1)} = C^{(i)} + \mu_C \left(\nabla_C l\left(\theta^{(i)}; \mathcal{D}\right)\right), \qquad (3)$$

where μ_C is a step-size parameter. The update rules for A, W, and Q are analogous. The updated parameter values are then used in the KF decoder for subsequent closed-loop control, and LGA begins collecting the next N pairs of data. Note that it is not necessary to adapt all four KF matrices. For instance, in our simulations we chose to only adapt the KF observation model parameters (C and Q), and kept the state transition model parameters (A and W) fixed over time, consistent with prior experimental demonstrations involving CLDA [8], [9], [14].

To derive the actual update rules of the LGA algorithm, we first write the log likelihood as:

$$\begin{aligned}
l(\theta; \mathcal{D}) &= \log p(\mathcal{D}|\theta) \\
&= \log\left[\left(\prod_{n=2}^{N} p(\tilde{x}_n|\tilde{x}_{n-1})\right) \cdot \left(\prod_{n=1}^{N} p(y_n|\tilde{x}_n)\right)\right] \\
&= \sum_{n=2}^{N} \log p(\tilde{x}_n|\tilde{x}_{n-1}) + \sum_{n=1}^{N} \log p(y_n|\tilde{x}_n) \\
&= \text{constant} - \frac{N-1}{2}\log|W| - \frac{N}{2}\log|Q| \\
&\quad -\frac{1}{2}\sum_{n=2}^{N} (\tilde{x}_n - A\tilde{x}_{n-1})^T W^{-1} (\tilde{x}_n - A\tilde{x}_{n-1}) \\
&\quad -\frac{1}{2}\sum_{n=1}^{N} (y_n - C\tilde{x}_n)^T Q^{-1} (y_n - C\tilde{x}_n) \qquad (4)
\end{aligned}$$

where we used the fact that $x_n|x_{n-1} \sim \mathcal{N}(Ax_{n-1}, W)$ and $y_n|x_n \sim \mathcal{N}(Cx_n, Q)$. We can then calculate the gradients of the log likelihood with respect to each of the parameters $\{A, W, C, Q\}$ as:

$$\begin{aligned}
\nabla_A l(\theta; \mathcal{D}) &= W^{-1}\left(\tilde{X}_2 - A\tilde{X}_1\right)\tilde{X}_1^T \\
\nabla_W l(\theta; \mathcal{D}) &= -\frac{N-1}{2}W^{-1} \\
&\quad + \frac{1}{2}W^{-1}\left(\tilde{X}_2 - A\tilde{X}_1\right)\left(\tilde{X}_2 - A\tilde{X}_1\right)^T W^{-1} \\
\nabla_C l(\theta; \mathcal{D}) &= Q^{-1}\left(Y - C\tilde{X}\right)\tilde{X}^T \\
\nabla_Q l(\theta; \mathcal{D}) &= -\frac{N}{2}Q^{-1} \\
&\quad + \frac{1}{2}Q^{-1}\left(Y - C\tilde{X}\right)\left(Y - C\tilde{X}\right)^T Q^{-1}
\end{aligned}$$

where we have expressed the gradients compactly by defining matrices $\tilde{X}$ and Y whose columns are composed of $\{\tilde{x}_n\}_{n=1}^N$ and $\{y_n\}_{n=1}^N$, respectively. These gradients are then evaluated at the current KF parameter setting $\theta^{(i)}$, and used to determine the next parameter setting $\theta^{(i+1)}$. For comparison, note that the update rule for C performed by the AKF CLDA algorithm [10], which performs gradient descent based on a different objective function, can be written as:

$$C^{(i+1)} \quad = \quad C^{(i)} + \mu_C \left(y_t - C^{(i)}\tilde{x}_t\right)\tilde{x}_t^T,$$

For LGA, an alternate (and simpler) set of updates rules for W and Q can be obtained by calculating the gradients of the log likelihood with respect to W^{-1} and Q^{-1}:

$$\nabla_{W^{-1}} l\left(\theta; \mathcal{D}\right) = \frac{N-1}{2}W - \frac{1}{2}\left(\tilde{X}_2 - A\tilde{X}_1\right)\left(\tilde{X}_2 - A\tilde{X}_1\right)^T$$

$$\nabla_{Q^{-1}} l\left(\theta; \mathcal{D}\right) = \frac{N}{2}Q - \frac{1}{2}\left(Y - C\tilde{X}\right)\left(Y - C\tilde{X}\right)^T$$

One can then instead update $\left(W^{-1}\right)^{(i)}$ and $\left(Q^{-1}\right)^{(i)}$ directly, using a rule similar to Eq. (3).

III. RESULTS

We compared the performance of the LGA algorithm to the AKF in a 2-D center-out cursor control task using a closed-loop BMI simulator. This method of simulating neural spiking activity was used previously in a BMI simulation in which human subjects generated the intended velocity with natural arm movements [15]. In our simulation, at each time step, the simulated subject observed the current position of the cursor and calculated a velocity that would direct it towards the target. A noisy version of this velocity direction was then used to generate spike counts from $m = 20$ simulated neurons. The spike rate $\lambda_i(n)$ of neuron i at time step n was given by:

$$\lambda_i(n) = \max\left(0, \langle PD_i, v_n^{sub}\rangle + b_i\right)$$

where b_i and PD_i are the baseline firing rate and preferred direction vector of neuron i, and v_n^{sub} is the noise-corrupted velocity direction generated by the subject. Baseline firing rates were chosen between 5–10 Hz, and preferred directions were randomly sampled from the distribution:

$$\angle PD_i \sim \text{Uniform}[0, 2\pi), \quad \|PD_i\| = \frac{14\,\text{spikes/sec}}{20\,\text{cm/sec}},$$

Binned spike counts were then sampled from a Poisson distribution:

$$y_{n,i} \sim \text{Poisson}\left(\lambda_i(n) \cdot \text{dt}\right)$$

and fed into a KF decoder (dt was set to be 100 ms). A and W were set to obey physical kinematics, such that integrating the velocity from one KF iteration perfectly explains position at the next iteration [7]. Elements of C were initialized by sampling from a standard normal distribution and Q was initialized to $Q = 10 \cdot I_m$. Then, either the LGA or AKF CLDA algorithm was used to adapt C and Q while the simulated subject performed a center-out task. CLDA was performed only during the first 8 trials, after which decoder parameters were held fixed for the rest of the simulation. To evaluate performance, we used 2 metrics that have previously been used to measure the accuracy of cursor trajectories: 1) Movement Error (ME; average deviation perpendicular to the reach direction), and 2) Movement Variability (MV; standard deviation of movement errors perpendicular to the reach direction) [8], [16].

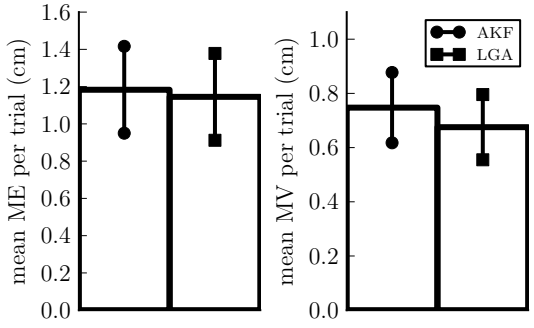

Fig. 1. Simulated closed-loop performance of the AKF and LGA CLDA algorithms when all neuronal baseline firing rates were equal (10 Hz). LGA generated decoder parameters resulting in less movement variability than AKF ($p < 0.05$, Kruskal-Wallis test) but the difference in movement error was insignificant in this simulation condition.

Figure 1 shows the subject's closed-loop performance during 100 simulated sessions in which baseline firing rates were set to 10 Hz for all simulated neurons. Cursor trajectories showed less movement variability ($p < 0.05$, Kruskal-Wallis test) when the LGA algorithm was used to perform CLDA instead of the AKF algorithm. The difference in movement error was insignificant ($p > 0.05$, Kruskal-Wallis test).

Closer inspection of both CLDA algorithms' updates rules for C reveals that LGA's update rule contains an "extra" Q^{-1} term. This difference suggests that LGA potentially has greater advantages over the AKF when the noise variances of neurons, which are modeled by Q, may vary across the BMI neural population. To test this hypothesis, we simulated the more realistic condition where modulation depths and baseline firing rates of the simulated neurons varied (evenly distributed between 5 and 10 Hz). In 100 simulated task sessions, cursor trajectories showed 10% less movement error ($p = 0.03$, Kruskal-Wallis test) and 13% less movement variability ($p = 0.0001$, Kruskal-Wallis test) when the LGA algorithm was used to perform CLDA instead of the AKF (Figure 2). These differences suggest that LGA's advantages over the AKF algorithm are more pronounced when the neural ensemble being used for control has a wide dynamic range of firing rate statistics.

IV. DISCUSSION

The LGA algorithm is a fully gradient-based CLDA algorithm for a Kalman filter decoder. Previous work developed and tested a partially gradient-based CLDA algorithm known as the Adaptive Kalman Filter (AKF). Although the AKF's update rules for the A and C matrices are gradient-based, its update rules for the W and Q covariance matrices are instead of heuristic form. Moreover, the AKF's update rules for A and C are based on separate mean-squared error objective functions. In contrast, all of LGA's update rules are gradient-

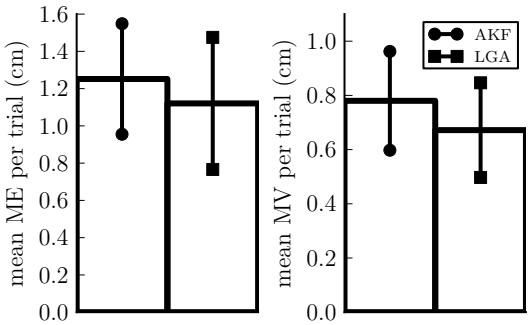

Fig. 2. Simulated closed-loop performance of the AKF and LGA CLDA algorithms for the more realistic conditions when baseline firing rates differ between neurons (chosen between 5 and 10 Hz). The smaller movement error and movement variability of decoders trained using LGA versus AKF is statistically significant ($p < 0.05$, Kruskal-Wallis test).

based and are derived directly from a single, unified log likelihood objective function. Since the KF assumes probabilistic state transition and observation models, a gradient-based KF CLDA algorithm should ideally be based on an objective function that accurately captures the models' probabilistic nature. With respect to this goal, the AKF's mean-squared error objective functions are not ideal because they do not have a probabilistic interpretation. On the other hand, the log likelihood function has a clear probabilistic interpretation as the (log) probability of observed data under a given set of parameters, and therefore in this sense, the LGA algorithm's choice of a log likelihood objective is optimal. For instance, unlike the more heuristic AKF method, LGA automatically rescales updates to neural tuning parameters based on the noise covariance of the neural population.

The LGA algorithm's gradient-based paradigm presents many potential advantages in the context of CLDA. For instance, LGA's update rules aim to "correct" rather than completely overwrite the current KF parameter setting, and the resulting smoothness in parameter updates can potentially make the algorithm robust to sporadic periods of unreliable data (e.g., when an animal subject briefly stops attending to the BMI task). Additionally, by choosing appropriate step-size values, the algorithm can be customized to achieve any desired level of smoothness or aggressiveness in parameter updates. Fine-grained control over CLDA with a single parameter may be important in settings where the experimenter must manage the rate of CLDA in real-time to maximize perfomance. Furthermore, by setting an appropriate value of the batch size N, the LGA algorithm can be applied to adapt a KF decoder on any time-scale.

The LGA algorithm operates within the CLDA paradigm of using an estimate of the user's intended movements to update the decoder. Depending on the particular method used for calculating this estimate, LGA could potentially be applied in multiple different settings. For instance, LGA could use Gilja et al.'s supervised method ("innovation 1" of the ReFIT-KF) [7] as part of a dedicated training session (i.e., initial decoder training or periodic recalibration). Alternatively, LGA could use Li et al.'s unsupervised method (Kalman smoothing of previously decoded outputs) [9] to update decoder parameters during everyday BMI operation.

One common shortcoming of both LGA and AKF is due to the stochastic nature of the gradient updates, as single time-point estimates of the gradient can be quite noisy and inaccurate. Although larger batches can be used in both algorithms, the smoothness benefits of using larger batches are not well characterized. In future work, we will explore alternative ways in which accurate updates may be generated while still adapting KF parameters on every iteration.

References

[1] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829 –1832, June 2002.

[2] J. M. Carmena, M. A. Lebedev, R. E. Crist, J. E. O'Doherty, D. M. Santucci, D. F. Dimitrov, P. G. Patil, C. S. Henriquez, and M. A. L. Nicolelis, "Learning to control a brain–machine interface for reaching and grasping by primates," *PLoS Biology*, vol. 1, no. 2, p. e42, Oct. 2003.

[3] L. R. Hochberg, D. Bacher, B. Jarosiewicz, N. Y. Masse, J. D. Simeral, J. Vogel, S. Haddadin, J. Liu, S. S. Cash, P. van der Smagt, and J. P. Donoghue, "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm," *Nature*, vol. 485, no. 7398, pp. 372–375, May 2012.

[4] V. Gilja, C. A. Chestek, I. Diester, J. M. Henderson, K. Deisseroth, and K. V. Shenoy, "Challenges and opportunities for next-generation intracortically based neural prostheses," *IEEE Transactions on Bio-Medical Engineering*, vol. 58, no. 7, pp. 1891–1899, July 2011.

[5] R. Héliot, S. Venkatraman, and J. M. Carmena, "Decoder remapping to counteract neuron loss in brain-machine interfaces," *IEEE EMBS Conference*, vol. 2010, pp. 1670–1673, 2010.

[6] B. Mahmoudi and J. C. Sanchez, "A symbiotic brain-machine interface through value-based decision making," *PLoS ONE*, vol. 6, no. 3, p. e14760, Mar. 2011.

[7] V. Gilja, P. Nuyujukian, C. A. Chestek, J. P. Cunningham, B. M. Yu, J. M. Fan, M. M. Churchland, M. T. Kaufman, J. C. Kao, S. I. Ryu, and K. V. Shenoy, "A high-performance neural prosthesis enabled by control algorithm design," *Nature Neuroscience*, vol. 15, no. 12, pp. 1752–1757, 2012.

[8] A. Orsborn, S. Dangi, H. Moorman, and J. Carmena, "Closed-loop decoder adaptation on intermediate time-scales facilitates rapid BMI performance improvements independent of decoder initialization conditions," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 20, no. 4, pp. 468–477, 2012.

[9] Z. Li, J. E. O'Doherty, M. A. Lebedev, and M. A. L. Nicolelis, "Adaptive decoding for brain-machine interfaces through bayesian parameter updates," *Neural Computation*, vol. 23, no. 12, pp. 3162–3204, 2011.

[10] S. Dangi, S. Gowda, R. Héliot, and J. M. Carmena, "Adaptive kalman filtering for closed-loop brain-machine interface systems," in *IEEE EMBS Conference on Neural Engineering*, May 2011, pp. 609–612.

[11] A. L. Orsborn, S. Dangi, H. G. Moorman, and J. M. Carmena, "Exploring time-scales of closed-loop decoder adaptation in brain-machine interfaces," *IEEE EMBS Conference*, vol. 2011, pp. 5436–5439, 2011.

[12] W. Wu, Y. Gao, E. Bienenstock, J. P. Donoghue, and M. J. Black, "Bayesian population decoding of motor cortical activity using a kalman filter," *Neural Computation*, vol. 18, no. 1, pp. 80–118, Jan. 2006.

[13] L. Shpigelman, H. Lalazar, and E. Vaadia, "Kernel-ARMA for hand tracking and brain-machine interfacing during 3D motor control," *Advances in Neural Information Processing Systems*, vol. 21, 2008.

[14] S. Dangi, A. L. Orsborn, H. G. Moorman, and J. M. Carmena, "Design and analysis of closed-loop decoder adaptation algorithms for brain-machine interfaces," *Neural Computation*, in press.

[15] J. P. Cunningham, P. Nuyujukian, V. Gilja, C. A. Chestek, S. I. Ryu, and K. V. Shenoy, "A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces," *Journal of Neurophysiology*, vol. 105, no. 4, pp. 1932–1949, Apr. 2011.

[16] S.-P. Kim, J. D. Simeral, L. R. Hochberg, J. P. Donoghue, and M. J. Black, "Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia," *Journal of Neural Engineering*, vol. 5, no. 4, pp. 455–476, Dec. 2008.