Closed-loop decoder adaptation algorithms for brain-machine interface systems

Siddharth Dangi

Closed-loop decoder adaptation algorithms for brain-machine interface systems

by

Siddharth Dangi

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Jose Carmena, Chair
Professor Anant Sahai
Professor Bruno Olshausen

Spring 2015

Closed-loop decoder adaptation algorithms for brain-machine interface systems

Abstract

Closed-loop decoder adaptation algorithms for brain-machine interface systems

by

Siddharth Dangi

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Jose Carmena, Chair

Brain-machine interfaces (BMIs) aim to assist patients suffering from neurological injuries and disease by enabling them to use their own neural activity to control external devices such as computer cursors or robotic arms, or even drive movements of their own body via muscle stimulation. At the heart of a BMI system is the decoding algorithm, or "decoder", that translates recorded neural activity into control signals for a prosthetic device. Decoders are often initialized offline by first recording neural activity while a subject performs real movements, or observes or imagines movements, and then fitting a decoder to predict these movements from the neural activity. However, BMIs are fundamentally closed-loop systems, since BMI users receive performance feedback (e.g. by visual observation of the prosthetic's movements), and the prediction power of decoders trained offline does not directly correlate with closed-loop performance. In other words, a high level of BMI performance can not necessarily be achieved solely by optimizing decoder parameters in an open-loop setting.

Two different mechanisms have been leveraged in the closed-loop regime to facilitate performance improvements. The first mechanism, neural plasticity or brain adaptation, is the ability of neurons to adapt their receptive fields and tuning properties to facilitate performance improvements. The second, a newly-emerging paradigm known as Closed-Loop Decoder Adaptation (CLDA), aims to update decoder parameters during closed-loop BMI operation in order to make the decoder's output more accurately reflect the user's intended BMI movements. In this work, we leverage the power of CLDA to both improve and maintain BMI performance. First, we introduce a CLDA algorithm that can rapidly improve BMI performance independent of method by which the decoder is seeded, which may be crucial for clinical applications where patients have limited movement and sensory abilities due to motor deficits. We then present a general framework for the design and analysis of CLDA algorithms, and demonstrate that mathematical convergence analysis can be a useful paradigm for evaluating the convergence properties of a prototype CLDA algorithm before experimental testing. Next, we then apply the CLDA technique to demonstrate high-performance, proficient BMI control based on local field potential signals instead of spikes, and demonstrate that there is broad flexibility in the frequencies that could potentially be used for LFP-based BMI control. Finally, we introduce a new CLDA algorithm called Recursive Maximum Likelihood that adapts decoder parameters very rapidly and efficiently, and possesses a variety of useful properties and practical algorithmic advantages. We test all of our algorithms and methods in closed-loop experiments by training macaque monkeys to perform a center-out reaching task using either spiking activity or local field potentials to control a 2D computer cursor. Overall, our work makes important progress towards demonstrating the power of closed-loop decoder adaptation as a useful tool for developing high-performance brain machine interface systems.

To Sebastian, Jeeves, and Cartman.

Acknowledgments

There were many times early on when I wanted to quit. Don't get me wrong — working on algorithms for brain-controlled prosthetic devices was awesome...but what if I had chosen a different research area that I would have liked even more? What if my PhD took forever to complete? What if I didn't want to continue BMI research forever because I had found lots of other things interesting too, and instead wanted to explore other areas after my PhD? Whenever I ever walked into his office feeling uncertain with these questions or others on my mind, my advisor Jose Carmena always made sure that walked out feeling reassured, motivated, determined, and most of all — excited! (I still haven't found anyone else who talks as excitedly — and rapidly — when discussing something they are passionate about!) I certainly wouldn't have made it to the end of my PhD without his support, but more importantly, he helped me come to the realization that getting a PhD is not about the final destination, but about the journey. It's indeed been a fun ride, and I know that even as I explore other fields and develop new interests and passions in the years ahead, the things I've learned and the skills I've developed along the way during my PhD will be invaluable to me. I have Jose to thank for being such a good advisor, mentor, and friend.

I would also like to thank Anant Sahai, Bruno Olshausen, and Jan Rabaey for serving on my qualifying exam committee, and Anant and Bruno for serving on my dissertation committee as well. Their advice and support has meant a lot to me.

As a graduate student, I couldn't have asked for better lab-mates and colleagues — directly or indirectly, they've all had an impact on me and helped me along the way. I'm very grateful to Simon Overduin and Amy Orsborn for selflessly helping to teach me to work with our monkeys, all while expecting nothing in return. Even after being trained to run my own monkey experiments, I was incredibly fortune that Amy, Simon, Helene Moorman, and Suraj Gowda had already spent countless hours training Sebastian, Jeeves, and Cartman — the monkeys whose data enabled the research presented in this dissertation. Vivek Athalye, Preeya Khanna, and Kelvin So are all great friends, and they made our 4-person Sutardja Dai office a lively and exciting place to come into work every day. (Vivek — no matter how many times I secretly practiced at night while you weren't in the office, you always had my number in our mini-basketball battles.) Ryan Neely would wake up at 4:30am, run 20 miles or jog up a mountain, and then come into work at the same time as the rest of us and do awesome science — whether he knew it or not, he was an inspiration to me and everyone else in the lab. Aaron Koralek and I never hung out much and didn't have the opportunity to work together, but his distinctive laugh alerted all of us when he was nearby and always brightened everyone's day. Andrea and Nerea — thanks for being such good friends, showing me around San Sebastian and Tubingen, and even teaching me some Spanish! Ander — I truly admire your passion for what you do, your determination to get it done, and your leadership to make it happen. Thanks for welcoming me into your home and hosting me, during times of both good and bad health. Simon, Ryan Canolty, Maryam Shanechi, and Samantha Summerson — you were an incredibly talented group of post-docs, and we were all lucky to have you in our lab as colleagues and friends. Ale Dominguez — you were the best lab manager one could ask for, one of the friendliest and kindest people I've met, and also one of the funniest – you always confused me, Suraj, and Vivek without knowing it!

I've collaborated with so many people over the years. In particular, Amy Orsborn and I partnered together on much of the work and experiments involving SmoothBatch, and I was fortunate to have joined the lab at the perfect time that enabled us to work together. Kelvin So and I also collaborated very closely on the closed-loop LFP BMI work, and there are few other people in the world with whom running monkey experiments together or co-writing papers could be so smooth, painless, and enjoyable. Finally, Suraj and I worked together on so many things that I don't even know where to start. It was truly fantastic that I had someone as smart and talented, and as good of a friend as him that I could ask a question to or discuss an idea with at any time, in lab or in our apartment. My time at Berkeley would not have been nearly as fun without him to share much of it with.

Graduate school is a roller coaster of highs and lows, and I wouldn't have made it to the end of the ride without my friends. Dan, Cam, Arjun, Suraj, Julian, and Kevin – our 3-on-3 basketball battles at Live Oak Park were something I looked forward to every single week, and know I will miss dearly once we leave this place and move on to other things. Arjun — you've been a really great friend (and even better doppelganger) since undergrad, and you could always be counted on to grab food with me on a moment's notice and talk about anything at any time of the day or night. Fulbert Chan — it's eerie how pretty much every time my research wasn't going well, you would start a funny conversation with me on gchat, which would make me crack a smile and help me take my mind off of things. Shervin, Julian, Kevin, and Suraj — our Bonita Ave. apartment wouldn't have won any awards for appearance or earthquake safety readiness, but you all helped make it such a memorable place, were terrific roommates, and will be life-long buddies. To all my friends — the birthday celebrations at Jupiter, Cal football games, trips to Tahoe, Super Bowl parties, and all the other little things that we did together helped keep me sane and happy, and I hope we will have many more of them in the years to come.

Finally, I would like to thank my family. To my sister, Shalini – I know the 8-year-old me had begged mom and dad for a brother, but despite all the teasing and tough love, it turns out I couldn't have asked for a better sibling and I can't imagine what my life would be like without you. To my parents, Salil and Vinita — you were born and raised in India without wealth or priviledge, yet somehow, through years and years of hard work, managed to leave home behind, make it to America, and start your family here. My "accomplishments" pale in comparison to what you both have done, and for your own continual sacrifices to make sure that Shalini and I had the opportunity for a better life, I am eternally grateful.

Contents

Chapter 1

Introduction

Brain-machine interfaces (BMIs) aim to assist patients suffering from neurological injuries and disease by enabling them to use their own neural activity to control external devices such as computer cursors or robotic arms, or even drive movements of their own body via muscle stimulation. Various forms of BMI control have been demonstrated in rodents [1–3], non-human primates [4–15], and humans [16–19]. However, significant improvements in both reliability (lifetime usability of the interface) and performance (achieving control and dexterity comparable to natural movements) are still needed to achieve clinically viable neuroprostheses for humans [20, 21].

A fundamental component of any BMI is the decoding algorithm, or "decoder", that translates recorded neural activity into control signals for a prosthetic device (see Figure 1.1)[1]. The Kalman filter is one popular decoding algorithm choice for brain-machine interface systems; in Chapter 2, we review the Kalman filter model and equations and its application to the BMI setting. Decoders such as Kalman filters are often initialized offline by recording neural activity while a subject performs real movements [4–8], observes movements [22], or imagines moving [9, 14, 16, 17], and fitting a decoder to predict these movements from the neural activity. However, BMIs are fundamentally closed-loop systems, since BMI users receive performance feedback (e.g. by visual observation of the prosthetic's movements). Recent research shows that the prediction power of decoders trained offline does not directly correlate with closed-loop performance, perhaps due to the inherent feedback-related differences between open- and closed-loop systems [23, 24]. These results suggest that performance improvements cannot be achieved solely by finding an optimal open-loop decoding algorithm, and therefore highlight the importance of treating BMIs as closed-loop systems.

[1]Figure courtesy of Amy Orsborn.

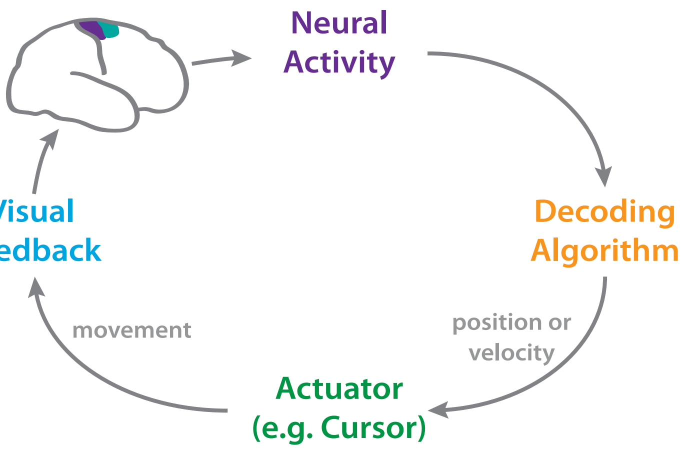


Figure 1.1: Example closed-loop diagram for a brain-machine interface system.

After the decoder has been initialized, two different mechanisms have been leveraged in the closed-loop regime to facilitate performance improvements. The first mechanism, neural plasticity or brain adaptation, is the ability of neurons to adapt their receptive fields and tuning properties to facilitate performance improvements. For instance, previous work from Ganguly and Carmena demonstrated that when monkeys practiced BMI control with a “stable circuit” consisting of a fixed decoder and stable neurons, they developed a stable neural “map” of the decoder that enabled performance improvements, was readily recalled across days, and was robust to interference from a second learned map [13]. The second, a newly-emerging paradigm known as Closed-Loop Decoder Adaptation (CLDA), aims to update decoder parameters during closed-loop BMI operation in order to make the decoder's output more accurately reflect the user's intended BMI movements.

Developing CLDA algorithms capable of rapidly improving performance, independent of initial performance, may be crucial for clinical applications where patients have limited movement and sensory abilities due to motor deficits. Given the subject-decoder interactions inherent in closed-loop BMIs, the decoder adaptation time-scale may be of particular importance when initial performance is limited. In Chapter 3 we present SmoothBatch, a CLDA algorithm for a Kalman filter decoder which updates decoder parameters on a 1–2 min time-scale using an exponentially weighted sliding average. The algorithm was experimentally tested with one nonhuman primate performing a center-out reaching BMI task. SmoothBatch was seeded four ways with varying offline decoding power: 1) visual observation of a cursor, 2) ipsilateral arm movements, 3) baseline neural activity, and 4) arbitrary weights. Our results show that SmoothBatch rapidly improved performance regardless of seeding, and that after decoder adaptation ceased, the subject maintained high performance. Moreover, performance improvements were paralleled by SmoothBatch convergence, suggesting that CLDA involves a co-adaptation process between the subject and the decoder.

Designing the SmoothBatch CLDA algorithm required making multiple important decisions, including choosing the time-scale of adaptation, selecting which decoder parameters to adapt, crafting the corresponding update rules, and designing CLDA parameters. These design choices, combined with the specific settings of CLDA parameters, will directly affect the algorithm's ability to make decoder parameters converge to values that optimize performance. In Chapter 4, we present a general framework for the design and analysis of CLDA algorithms, and support our results with experimental data of two monkeys performing a BMI task. First, we analyze and compare existing CLDA algorithms to highlight the importance of four critical design elements: the adaptation time-scale, selective parameter adaptation, smooth decoder updates, and intuitive CLDA parameters. Second, we introduced mathematical convergence analysis, using measures such as mean-squared error and KL divergence, as a useful paradigm for evaluating the convergence properties of a prototype CLDA algorithm before experimental testing. By applying these measures to an existing CLDA algorithm, we demonstrated that our convergence analysis is an effective analytical tool that can ultimately inform and improve the design of CLDA algorithms.

Intracortical brain–machine interfaces (BMIs) have predominantly utilized spike activity as the control signal — however, an increasing number of studies have shown the utility of local field potentials (LFPs) for decoding motor related signals. Yet currently, it is unclear how well different LFP frequencies can serve as features for continuous, closed-loop BMI control. In Chapter 5, using closed-loop decoder adaptation as a tool to adapt decoder parameters to subject-specific LFP feature modulations during BMI control, we demonstrate 2D continuous LFP-based BMI control. We trained two macaque monkeys to control a 2D cursor in a center-out task by modulating LFP power in the 0–150 Hz range. While both monkeys attained control, they used different strategies involving different frequency bands. One monkey primarily utilized the low-frequency spectrum (0–80 Hz), which was highly correlated between channels, and obtained proficient performance even with a single channel. In contrast, the other monkey relied more on higher frequencies (80–150 Hz), which were less correlated between channels, and had greater difficulty with control as the number of channels decreased. We then restricted the monkeys to use only various sub-ranges (0–40, 40–80, and 80–150 Hz) of the 0–150 Hz band. Interestingly, although both monkeys performed better with some sub-ranges than others, they were able to achieve BMI control with all sub-ranges after decoder adaptation, demonstrating broad flexibility in the frequencies that could potentially be used for LFP-based BMI control. Overall, our results demonstrate proficient, continuous BMI control using LFPs and provide insight into the subject-specific spectral patterns of LFP activity modulated during control.

In some applications, the time required for initial decoder training and any subsequent decoder recalibrations could be potentially reduced by performing continuous adaptation, in which decoder parameters are updated at every time step during these procedures, rather than waiting to update the decoder at periodic intervals in a more batch-based process. In Chapter 6, we present recursive maximum likelihood (RML), a CLDA algorithm that performs continuous adaptation of a Kalman filter decoder's parameters. We demonstrate that RML possesses a variety of useful properties and practical algorithmic advantages. First, we show how RML leverages the accuracy of updates based on a batch of data while still adapting parameters on every time step. Second, we illustrate how the RML algorithm is parameterized by a single, intuitive half-life parameter that can be used to adjust the rate of adaptation in real time. Third, we show how evenwhen the number of neural features is very large, RML's memory-efficient recursive update rules can be reformulated to also be computationally fast so that continuous adaptation is still feasible. To test the algorithm in closed-loop experiments, we trained three macaque monkeys to perform a center-out reaching task by using either spiking activity or local field potentials to control a 2D computer cursor. RML achieved higher levels of performance more rapidly in comparison to a previous CLDA algorithm that adapts parameters on a more intermediate timescale. Overall, our results indicate that RML is an effective CLDA algorithm for achieving rapid performance acquisition using continuous adaptation.

Finally, in Chapter 7, we conclude by summarizing the main contributions of this dissertation and proposing some directions for future work.

Chapter 2

Decoding Methods

2.1 Kalman Model and filter equations

The Kalman filter (KF) is a linear, recursive estimator for estimating an unknown state x_t as it evolves over time, given noisy observations y_t of the state. Specifically, the standard form of the filter assumes the following state evolution and state observation models:

$$\begin{aligned} x_t &= Ax_{t-1} + w_t, \qquad w_t \sim \mathcal{N}(0, W) && (2.1) \\ y_t &= Cx_t + q_t, \qquad q_t \sim \mathcal{N}(0, Q) && (2.2) \end{aligned}$$

where w_t and q_t are additive Gaussian noise terms with covariance matrices W and Q, respectively. The inference problem that the Kalman filter solves (optimally, assuming that the models above are accurate) is computing the minimum mean squared error estimate $\hat{x}_t$ of the state x_t at each time step, given the history of observations $\{y_t, y_{t-1}, y_{t-2}, ...\}$. The Kalman filter performs this estimation at recursively at each filter iteration using an efficient, linear algorithm — in other words, it computes the estimate $\hat{x}_t$ using only: 1) the most recent observation, y_t, and 2) and its own previous estimate $\hat{x}_{t-1}$.

In particular, on each iteration, the Kalman filter uses first its previous kinematic state estimate $\hat{x}_{t-1}$ (and estimate covariance P_{t-1}) to generate the *a priori* estimate $\hat{x}_{t|t-1}$ (and estimate covariance $P_{t|t-1}$) of x_t:

$$\begin{aligned} \hat{x}_{t|t-1} &= A\hat{x}_{t-1} \\ P_{t|t-1} &= AP_{t-1}A^T + W \end{aligned}$$

Secondy, it calculates the Kalman gain K_t, and uses the observation y_t to generate the *a posteriori* estimate $\hat{x}_t$ (and estimate covariance P_t) by making an additive correction to $\hat{x}_{t|t-1}$:

$$K_t = P_{t|t-1}C^T \left(CP_{t|t-1}C^T + Q\right)^{-1}$$

$$\begin{aligned}\hat{x}_t &= \hat{x}_{t|t-1} + K_t\left(y_t - C\hat{x}_{t|t-1}\right)\\ P_t &= \left(I - K_t C\right) P_{t|t-1}\end{aligned}$$

The estimate covariance matrix P_t at each time-step represents the filter's confidence in its own state estimates. Specifically, the diagonal terms of P_t represent the KF's variances for its estimates of the state vector variables (i.e., larger values reflect lower confidence). Note that during standard operation of the filter, P_t quickly converges to a steady-state matrix P that depends only the model parameters $\{A, W, C, Q\}$.

2.2 BMI Decoder Implementation

The Kalman filter is a common decoding algorithm choice for BMI applications [25, 26]. In the context of a BMI, x_t is a vector representing the intended state of prosthetic device (e.g., a computer cursor, prosthetic limb, robotic wheelchair, etc.) and y_t is a vector of features of recorded neural activity. In the work presented in subsequent chapter, all experimental sessions used a Kalman filter (KF) as the decoding algorithm during closed-loop control. For simplicity, we focused in particular on BMI cursor control — however, it is important to note that the Kalman filter and our presented closed-loop decoder adaptation methods do apply more generally to other types of BMI systems. For BMI cursor control, we used a position-velocity KF, in which the KF state vector x_t is defined to include the position (p) and velocity (v) of the cursor at time t, along both the horizontal (x) and vertical (y) directions of the screen:

$$x_t = \begin{bmatrix} p_{\mathrm{x},t} & p_{\mathrm{y},t} & v_{\mathrm{x},t} & v_{\mathrm{y},t} & 1 \end{bmatrix}^T.$$

The constant term 1 and a corresponding extra column of C are added to account for neural observations y_t with non-zero means.

2.3 Decoder Seeding

From the above equations, it is evident that the KF model is parametrized by the matrices $\{A, W, C, Q\}$. Hence, in order to use a KF as a decoding algorithm for a brain-machine interface, these matrices must be seeded with initial values.

2.3.1 Initializing A and W

In our experiments, the state evolution model parameters (A and W matrices, describing cursor dynamics) were defined to model the system physics, with position components set as the integral of velocity, as in [27]:

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & dt & 0 \\ 0 & 0 & a_v^{\mathrm{xx}} & a_v^{\mathrm{xy}} & 0 \\ 0 & 0 & a_v^{\mathrm{yx}} & a_v^{\mathrm{yy}} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \\ W &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_v^{\mathrm{xx}} & w_v^{\mathrm{xy}} & 0 \\ 0 & 0 & w_v^{\mathrm{yx}} & w_v^{\mathrm{yy}} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

where dt is the time between successive Kalman filter iterations. The velocity state transition model (A_v and W_v) were fit using their maximum-likelihood estimates

$$\begin{aligned} A_v &= \begin{bmatrix} a_v^{\mathrm{xx}} & a_v^{\mathrm{xy}} \\ a_v^{\mathrm{yx}} & a_v^{\mathrm{yy}} \end{bmatrix} = V_2 V_1^T \left(V_1 V_1^T \right)^{-1} \\ W_v &= \begin{bmatrix} w_v^{\mathrm{xx}} & w_v^{\mathrm{xy}} \\ w_v^{\mathrm{yx}} & w_v^{\mathrm{yy}} \end{bmatrix} = \frac{1}{N-1} \left(V_2 - AV_1 \right) \left(V_2 - AV_1 \right)^T \end{aligned}$$

where N is the total number of measured time-points, and V_1 and V_2 are matrices that are formed by tiling recorded velocity kinematics for times $[1, N-1]$ and $[2, N]$, respectively. In order to allow for BMI control that would mimic natural arm movements as much as possible, A and W were fit for all seeding methods using a data set of arm movements collected while the subject moved the cursor directly using movements of his arm.

2.3.2 Initializing C and Q

In our closed-loop BMI experiments, the state observation model parameters (C and Q matrices) were initialized ("seeded") using one of five different methods:

1. Visual Feedback, or VFB: C and Q were trained using neural activity recorded as the subject passively viewed a video of a cursor moving on a screen.

2. Ipsi: C and Q were trained using neural activity recorded as the subject moved the cursor using ipsilateral arm movements.

3. Contra: C and Q were trained using neural activity recorded as the subject moved the cursor using contralateral arm movements.

4. Baseline: C and Q were trained using neural activity recorded during quiet sitting.

5. Shuffled: A VFB decoder was first trained, and the rows of C and rows/columns of Q were then randomly shuffled in order to randomize the assignment of decoder weights to neural features.

For seeding methods 1–4, the observation model (C and Q matrices, describing neural activity's relation to cursor movement) were fit using their maximum-likelihood estimate

$$C = YX^T\left(XX^T\right)^{-1} \tag{2.3}$$

$$Q = \frac{1}{N}(Y - CX)(Y - CX)^T \tag{2.4}$$

where the Y and X matrices are formed by tiling recorded neural activity and cursor kinematics, respectively. For VFB seeding, a visual cursor was moved through artificially generated trajectories (straight trajectories, Gaussian speed profiles, 800 ms reach duration). The subject viewed the cursor movement while seated in the primate chair, receiving pseudo-random rewards on 25% of all trials. While no effort was made to constrain subject behavior, the animal typically sat quietly and looked at the display. Recorded neural activity and the artificially generated cursor kinematics were used for decoder estimation. Baseline seeding was performed using the same protocol as VFB, but with the display turned off so that the subject did not see anything. Ipsi and contra decoder seeds were created with neural data ipsilateral or contralateral to the arm kinematics recorded as the subject performed the center-out task with his arm.

Chapter 3

SmoothBatch: Improving BMI performance independent of initialization

The work presented in this chapter was performed in collaboration with Amy L. Orsborn, Helene G. Moorman, and Jose M. Carmena, and was published in IEEE Transactions on Neural Systems and Rehabilitation Engineering [28].

3.1 Introduction

Brain-machine interfaces (BMIs) have great potential to restore function to patients with motor disabilities including amputees and those suffering from spinal cord injury, stroke, and amyotrophic lateral sclerosis. Early work has provided a solid proof of concept for BMIs, with several groups showing demonstrations of rodents [1], nonhuman primates [4–14], and humans [16, 17] controlling artificial actuators using neural activity. However, in order to make BMI systems widely clinically viable, significant improvements in reliability (lifetime usability of the interface) and performance (achieving control and dexterity comparable to natural movements) are needed [20, 21].

BMI systems use an algorithm (the "decoder") to translate recorded neural activity (e.g., spike trains) into a control signal (e.g., position) for an external actuator such as a computer cursor. The BMI user receives performance feedback, typically in the form of visual observation, creating a closed feedback system. Thus, BMIs allow a user to modulate their neural activity to achieve a desired goal. BMI decoders are usually created in open-loop by first recording neural activity as a subject makes movements [4–8, 12, 27, 29, 30] or imagines moving [9, 14, 16, 17], and then training a decoder to predict these movements from the neural activity. However, open-loop decoder prediction power does not directly correlate with closed-loop performance [23, 24], suggesting that improvements in BMI performance cannot be achieved solely by finding an optimal open-loop decoding algorithm. Instead, recent work shows that significant improvements in performance can come from insights into the closed-loop BMI system, in which brain and machine adaptation play pivotal roles. For instance, Ganguly and Carmena [13] showed that when a subjects practiced BMI control with a fixed decoder, they learned a stable neural representation of the decoder, and the development of these stable representations paralleled improvements in control. In other words, the brain can adapt to improve performance. Other researchers have taken the opposite approach, investigating methods of closed-loop decoder adaptation (CLDA) to improve performance [2, 5, 27, 30, 31]. These studies show that closed-loop BMI performance can be significantly improved by using known or inferred task goals, or evaluative feedback, during closed-loop BMI control to modify the decoder.

CLDA algorithms typically have two components: 1) a method to infer a subject's intended movement goals during closed-loop control, and 2) a rule to update the decoder's parameters. One particularly interesting aspect of candidate decoder update algorithms is the time-scale on which they update the decoder. Gilja et al. [27] used a batch-based algorithm that applies one discrete decoder update 10–15 min after the initial seeding, while Shpigelman et al. [30] used a real-time update rule that adjusts the decoder at every decoder iteration. Given that closed-loop BMI performance involves an inherent interplay between the subject and the decoder, the rate at which the decoder changes will likely influence performance and the algorithm's ability to improve control. Moreover, this time-scale of adaptation may be paramount in situations where initial closed-loop performance may be severely limited, such as clinical applications for patients that cannot enact natural movement because of spinal cord injury or other neurological disorders [32]. It is thus worthwhile to identify the most appropriate time-scale of decoder adaptation to yield efficient CLDA algorithms that rapidly and robustly improve BMI performance regardless of initial closed-loop performance. Previous work [32] suggests that an algorithm using an intermediate (1–2 min) timescale may be ideal for situations with limited initial performance. Here, we present a new CLDA algorithm called SmoothBatch, which updates the decoder on this timescale. This algorithm implements a sliding average of decoder parameters estimated on small batches of data. SmoothBatch uses the method developed by Gilja et al. [27] to infer the subject's intended kinematics in a center-out task. We present experimental validation using data from one nonhuman primate subject. We also explore the algorithm's ability to improve closed-loop BMI performance independent of the initial decoder performance (seed) by comparing SmoothBatch's performance using four different decoder seeding methods: 1) visual observation of cursor movement, 2) ipsilateral arm movement, 3) neural activity during quiet sitting, and 4) arbitrary weights.

3.2 Methods

3.2.1 Electrophysiology

One adult male rhesus macaque (macaca mulatta) was used in this study. The subject was chronically implanted with microwire electrode arrays for neural recording. One array of 128 teflon-coated tungsten electrodes (35 m diameter, 500 μm wire spacing, 8×16 array configuration; Innovative Neurophysiology, Durham, NC) was implanted in each brain hemisphere, targeting the arm areas of primary motor cortex (M1) and dorsal premotor cortex (PMd). Localization was performed using stereotactic coordinates from rhesus brain anatomy [33]. Each array was positioned targeting M1, and due to the size of the array, extended rostrally into PMd. All procedures were conducted in compliance with the National Institutes of Health Guide for Care and Use of Laboratory Animals and were approved by the University of California, Berkeley Institutional Animal Care and Use Committee. Unit activity was recorded using a combination of two 128- channel MAP and OmniPlex recording systems (Plexon Inc, Dallas, TX). Single and multiunit activity was sorted using an online sorting application (Plexon, Inc, Dallas, TX), and only neural activity with well-identified waveforms were used for BMI control.

3.2.2 Behavioral Task

The subject was trained to perform a self-paced delayed 2-D center-out reaching task to eight targets (1.7 cm radius) uniformly spaced about a 14-cm-diameter circle. The animal sat in a primate chair, head restrained, and observed reach targets displayed via a computer monitor projecting to a semi-transparent mirror parallel to the floor. Figure 3.1 shows an illustration of the task setup and trial time-line. Trials are initiated by moving the cursor to the center target and holding for 400 ms. Upon entering the center, the reach target appears. After the center-hold period ends, the subject is cued to initiate the reach (via target flash), after which he must move the cursor to the peripheral target within a given 3 s time-limit and hold for 400 ms to receive a liquid reward. If the subject makes an error, defined as failure to hold at the center or target, or failure to reach the target within the time-limit, the trial is aborted and must be restarted. The subject has an unlimited amount of time to enter the center target to initiate a trial. Reach targets were presented in a block structure, with pseudo-randomized order within each block of eight targets. Initial task-training was conducted with the animal making arm movements. The arm was placed in a 2D KINARM exoskeleton (BKIN Technologies, Kingston, ON, Canada), which constrained movements to the horizontal plane parallel to and just under the reach-target display. A cursor co-localized with the center of the subject's hand was displayed on the screen to provide visual feedback of hand location. During BMI operation, the subject performed the center-out task by moving the cursor under neural control. The subject's arm were removed from the KINARM and confined within the primate chair. The subject was proficient (overtrained) in the center-out task with arm movements before BMI experiments commenced.

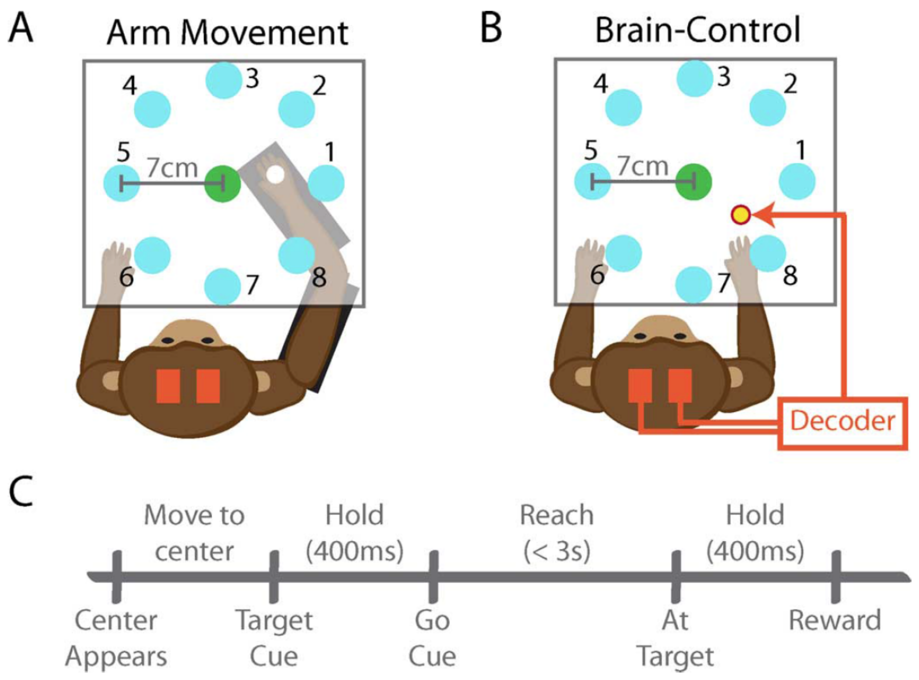


Figure 3.1: Behavioral task. (A) and (B) show a top-view of the experimental setup. The nonhuman primate subject sat in a primate chair viewing a 2-D center-out task projected in front of them. Initial training (and ipsi decoder seeding) was performed with the animal performing the task with his arm inside a 2-D exoskeleton (A). In BMI, the task was performed with the animal's arm removed from the exoskeleton and the cursor position controlled via BMI predictions (B). (C) Time-line of task events.

3.2.3 BMI Decoder

Online closed-loop BMI control was implemented using a Kalman filter (KF) decoder (see Chapter 2) with binned neuron spike counts as the choice of neural features y_t. Online BMI control was implemented using PlexNet (Plexon Inc., Dallas, TX) to stream neural data on a local intranet from the Plexon recording system to a dedicated computer running Matlab (The Mathworks, Natick, MA). Neural firing rates were estimated with a 100 ms bin width. Neural ensembles of 16–36 (26.5±4.45; mean±STD) neurons were used. Units were selected only based on waveform quality. Decoders were initialized using four of the methods described in Section 2.3:

1. Visual Feedback ($n = 20$)
2. Ipsi ($n = 8$)

3. Baseline ($n = 17$)
4. Shuffled ($n = 11$)

3.2.4 Closed-Loop Decoder Adaptation Algorithm: SmoothBatch

Once a seed decoder was created, CLDA was used to improve performance. The algorithm used, SmoothBatch, updates the decoder on an intermediate (1–2 min) time-scale. The subject performed the center-out task under BMI control as the algorithm updated the decoder. The subject's intended kinematics were inferred from the observed BMI performance using the technique developed by Gilja et al. [34, 35], which assumed that the subjects intends to reach straight to the target at all times and rotates observed cursor velocities to point towards the current task target. The observed neural activity and intended kinematics were collected for a short (1–2 min) interval. Each batch of data was used to construct a new estimate of the C and Q matrices (using 2.3 and 2.4), $\hat{C}$ and $\hat{Q}$. The BMI decoder was then updated using a weighted sum:

$$\begin{aligned} C^{(i+1)} &= \alpha C^{(i)} + (1-\alpha)\hat{C} \\ Q^{(i+1)} &= \beta Q^{(i)} + (1-\beta)\hat{Q} \end{aligned}$$

where i indexes discrete decoder iterations and $\alpha, \beta \in (0, 1)$ determine the speed of adaptation. We report α and β in terms of the half-life of the update process — i.e., how long it takes for the influence of a $\hat{C}$ estimate in the BMI decoder to reduce by half. The half-lives for C and Q, denoted by h_c and h_q, are related to α and β on the open interval (0, 1) as follows:

$$\alpha^{h_c/b} = 1/2 \quad \text{and} \quad \beta^{h_q/b} = 1/2$$

where b is the batch size. Note that and values depend upon both the half-life and batch size. To avoid conducting a vast parameter search of both batch size and half-life experimentally, SmoothBatch was first implemented in a BMI simulator that utilizes an empirically verified model of the user's learning process during closed-loop BMI operation [36]. Preliminary results from the simulator allowed were used to narrow down the search space, which was then explored in experiments. Batch sizes of 40–100 s and C and Q half-lives of 90–300 s were tested experimentally. Rough optimization showed that batch sizes of 60–100 s and half-lives of 90–210 s produced the most rapid performance improvements. All presented data use parameters within this range; the majority ($n = 46$) use an 80 s batch and 120 s half-life. In all cases, C and Q half-lives were set to be equal so that these matrices were updated at the same time. Also note that the A and W matrices (which parameterize the cursor dynamics model) were held fixed. As soon as a new decoder update was calculated, it was used for subsequent BMI predictions.

3.2.5 Data Analysis

3.2.5.1 Behavioral Performance Metrics

Only successfully initiated trials (i.e., entering the center and successfully holding until the go-cue) were considered for analysis, allowing for three possible outcomes: a successful reach, a target-hold error, or a reach time-out. The subject would typically leave the center early (a false-alarm) on approximately 10% of all attempts to initiate a reach, both during neural and arm control. The task structure did not penalize these errors, likely leading to the high false-alarm rate. Thus, they were excluded from analysis. Task behavior was quantified by analyzing trial outcomes both by percentage and event rates. The success, reach time-out, and error percentages over time were calculated using a sliding average (75 trial window) to estimate the evolution of performance within the session. The corresponding rates for these different trial outcomes were quantified by binning the event times (60 s wide, non-overlapping bins). The event percentage metric provides an estimate of overall task performance, while the event rates provide more temporal information about task performance (e.g., periods of inactivity when the subject is unable to initiate a trial will not be reflected in the trial-based percentage). A threshold of 8 successes/min was used to assess SmoothBatch's improvement time. The task required the subject to hit each presented reach target before moving to the next target. Therefore, achieving 8 successes/min typically corresponds to the subject successfully acquiring all targets, demonstrating control across the entire workspace. Reach times were quantified as the time between the cursor leaving the center target and entering the peripheral target or occurrence of a time-out error. Evolution of trajectory quality were estimated using metrics adopted from literature for pointing device evaluation and previously used to evaluate BMIs [17]. Average movement error (ME) and movement variability (MV) around the task-relevant axis (i.e., perpendicular to the reach target) were quantified. ME and MV are defined as

$$\mathrm{ME} = \frac{\sqrt{\sum |y_i|}}{n} \quad \text{and} \quad \mathrm{MV} = \sqrt{\frac{\sum (y_i - \bar{y})^2}{n}}$$

where y_i is the perpendicular distance from the task-axis at time-point i and $\bar{y}$ is the mean of y across the trajectory. ME and MV were computed for each trajectory using the time from leaving the center to arriving at the distal target or occurrence of a time-out error.

3.2.5.2 Decoder Metrics

The changes in KF matrices C and Q were computed as the decoder was updated using SmoothBatch. The Frobenius norm of the differences of consecutive update step were used to look at overall matrix adaptation. The velocity terms (in x and y directions) in the C matrix and their evolution in time were also analyzed. While the Kalman filter decoder estimated both position and velocity, analysis of the Kalman gains obtained showed that neural activity makes the strongest contributions to the cursor velocity, making these terms of particular importance for the decoder. Our Kalman filter assumes

$$y_i \approx C_{i,1}p_x + C_{i,2}p_y + C_{i,3}v_x + C_{i,4}v_y + C_{i,5}$$

where y_i is the firing rate of unit i. Ignoring position terms, this is a standard velocity tuning model [30]. A unit's velocity preferred-direction (PD) and modulation-depth (MD) are given by

$$\begin{aligned} \mathrm{PD}_i &= \tan^{-1}\left(\frac{C_{i,4}}{C_{i,3}}\right) \\ \mathrm{MD}_i &= \sqrt{C_{i,3}^2 + C_{i,4}^2} \end{aligned}$$

To assess how the decoder direction-tuning model evolved, the mean change in PD across neurons was quantified. More strongly tuned neurons (larger MD) contribute more to BMI performance than non-tuned units (small MD). Hence, changes in PDs of neurons with high MD more directly influence decoder performance. To capture this, the magnitude of PD change for a given unit was weighted by that unit's MD in the final decoder. The average weighted change in PD for the full decoder (ΔwPD) was calculated by averaging this weighted PD change across units. ΔwPD is defined as

$$\Delta w\mathrm{PD}\left(t\right) = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{\mathrm{MD}_i\left(T\right)}{\max_i\left(\mathrm{MD}\left(T\right)\right)}\left|\mathrm{PD}_i\left(t\right) - \mathrm{PD}_i\left(t-1\right)\right|\right)$$

where M is the total number of units in the decoder, and$t \in [1, T]$ indexes each decoder during adaptation.

3.2.5.3 Offline Seed Decoder Prediction Power

The offline prediction power of seed decoders were computed using the R^2 correlation coefficient between measured kinematics and predicted kinematics (x and y components of position and velocity). In ipsi seeds, measured kinematics corresponded to observed arm movements. In VFB and baseline seeds, the display cursor's kinematics were used as the measured kinematics. Note that in baseline seeding, the cursor was not viewed by the subject. Finally, for shuffled decoders, the decoder was used to predict kinematics using neural activity recorded during a VFB condition recorded within the same session, and those predictions were compared with the display cursor kinematics.

3.3 Results

3.3.1 SmoothBatch BMI Performance Improvements

SmoothBatch CLDA rapidly and reliably improved closed-loop BMI performance, despite being seeded with decoders constructed in the absence of contralateral arm movements. Figure 3.2 shows the evolution of task performance for a representative baseline seed session. As seen in the events per minute (quantified with a 120 s bin-width to reduce behavioral noise; Figure 3.2A), the subject was not readily able to perform the task with the seed decoder (only a few trials were initiated). After a few minutes (1–2 decoder updates), the subject was able to initiate trials but with limited control as shown by the increased rate of reach time-out events (Figure 3.2A) and the highly irregular reach trajectories in the first attempted reaches (3.2C). Performance improved gradually until approximately 10 min, when success percentage and rate both markedly improved.

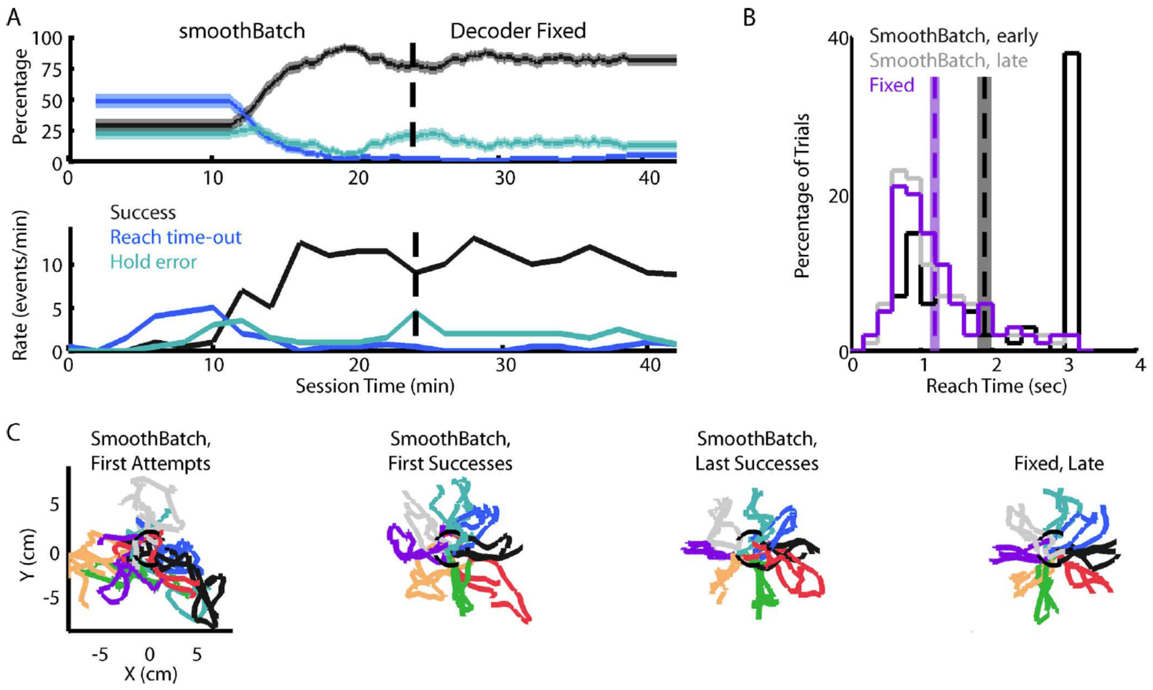


Figure 3.2: BMI task performance improvements using SmoothBatch closed-loop decoder adaptation for one representative baseline seeded session. (A) Sliding average (75 trial window) of task performance rates (top) and binned event hit-rates (bottom, 120 s bin width) during SmoothBatch adaptation and upon fixing the decoder. (B) Distribution of reach-times for the first and last 100 reaches during SmoothBatch operation, and the first 100 reaches upon fixing the decoder. Dashed lines and shaded regions indicate mean and standard error of the mean, respectively, for each. (C) Progression of reach trajectories during the session. Three reaches to each target are shown for the first attempted reaches of the session (left), first successful reaches during SmoothBatch (left middle), last successful reaches during SmoothBatch (right middle), and the last successful reaches with a fixed decoder (right).

These rapid, nonlinear performance improvements were observed across all sessions (including all decoder seed types). Initial performance started at 0.018±0.133 successes/min and exceeded 8 successes/min (which roughly corresponds to attaining successful reaches to all targets) within 13.1±5.5 min. Performance increased up to maximum rates of 14.3±1.37 successes/minute (maximum 88.04±5.3% trial success percentage) within 20.75±5.93 min (all mean±std; $n = 56$, all seed types). Reach trajectories (Figure 3.2C) showed improvement between the first success and late in SmoothBatch adaptation, becoming more stereotyped. Similarly, reach times showed a marked reduction from the first to last 100 trials during SmoothBatch, and remained low after the decoder was held fixed after adaptation ceased

(Figure 3.2B). Across all sessions, SmoothBatch significantly improved reach trajectory kinematics. The mean reach times, ME, and MV for the last 100 trials during SmoothBatch were significantly smaller than those of the first 100 trials in the session (Wilcoxon paired test, $p > 0.05$). At the end of SmoothBatch adaptation, the subject achieved average reach times of 1.23$\pm$0.16 s, ME of 0.771$\pm$0.088 cm, and MV of 0.593$\pm$0.067 cm (all mean$\pm$std; $n = 56$, all seed types).

As seen in the example session in Figure 3.2, performance typically improved gradually, requiring multiple batch updates before sufficient task performance was achieved. Many decoder seeds, particularly arbitrary seeds like baseline and shuffled, yielded decoders the subject could not readily use—even limiting the ability to initiate trials. On average, 1.23$\pm$1.1 batch updates were required before the subject successfully initiated a trial, 3.03$\pm$2.1 updates occurred before the first successful reach, and 7.43$\pm$3.6 updates occurred before the subject was able to successfully reach all eight targets. SmoothBatch improved the decoder gradually and did not jump to a high-performance solution in a single iteration.

The ultimate goal of CLDA is to find a decoder that will allow the subject to readily gain proficient control of the BMI to perform a variety of tasks. In many situations, particularly unstructured tasks, it may not always be possible to infer the user's movement goals. Hence, it is highly desirable for task performance to be maintained after decoder adaptation has ceased. To test this, after the subject attained adequate performance (≥ 10 trials/min, approximated by the experimenter), SmoothBatch adaptation was stopped and the subject performed the BMI task with a fixed decoder. As seen in Figure 3.2, the subject was able to maintain high task performance and accurate reaches after the decoder was held fixed. Across all sessions, no significant changes in performance were found after fixing the decoder. Because the subject used fixed decoders for varying periods of time across sessions, behavioral measures were compared between the final moments of SmoothBatch adaptation (last estimate of successes per minute; success percentage for the last 75 trials) and early fixed-decoder performance (successes per minute for the first 15 min; success percentages for the first 100 initiated trials). Neither the successes per minute nor the success percentages were significantly different between the end of SmoothBatch adaptation and fixing the decoder (Wilcoxon paired test, $p = 0.733$ and $p = 0.1$, respectively; $n = 54$, two sessions with insufficient data with a fixed decoder were excluded from analysis). Furthermore, mean reach times for the last 100 trials during SmoothBatch showed no significant difference from that of the first 100 trials with the decoder fixed (Wilcoxon paired test, $p = 0.55$). Mean ME and MV, however, showed a small (approximately 12%), significant increase between the last 100 trials in SmoothBatch and the first 100 trials with a fixed decoder (Wilcoxon paired test, $p < 0.05$), suggesting a very slight drop in trajectory precision.

3.3.2 Kalman Filter Decoder Evolution During SmoothBatch

The SmoothBatch algorithm showed convergence towards a stable decoder solution during adaptation. Figure 3.3A displays the Frobenius norm in the element-wise change in consecutive C and Q decoder matrices across time, showing that the change rapidly decreased for both matrices. This convergence was also seen in the velocity-weights in C, with the preferred directions assigned to each unit stabilizing over time. Fig. 3(B) shows the average magnitude of PD change weighted by neuron modulation depth (ΔwPD) during SmoothBatch adaptation. Task performance (successes/min) is overlaid, showing that convergence of the velocity PDs was strongly correlated with behavioral improvements. Polar plots of all decoder units' tuning (PD and MD) for example decoders during SmoothBatch adaptation are shown in Figure 3.3C (corresponding time-points are indicated in Figure 3.3B). The initial baseline-seeded decoder assigned weak tuning to all units. Near the point of significant behavioral improvement, as the decoder weights were converging, the MDs of units had significantly increased and a few units had PDs near those of the final decoder. Once performance significantly improved, the tuning model very closely resembled that of the final decoder. Across all sessions, the average ΔwPD between the decoder being used when the subject was first able to successfully reach all eight targets and the final decoder was $6.52 \pm 4.48°$, as compared to $23.79 \pm 12.17°$ for the initial seed decoder (mean$\pm$std; $n = 56$, all seed types).

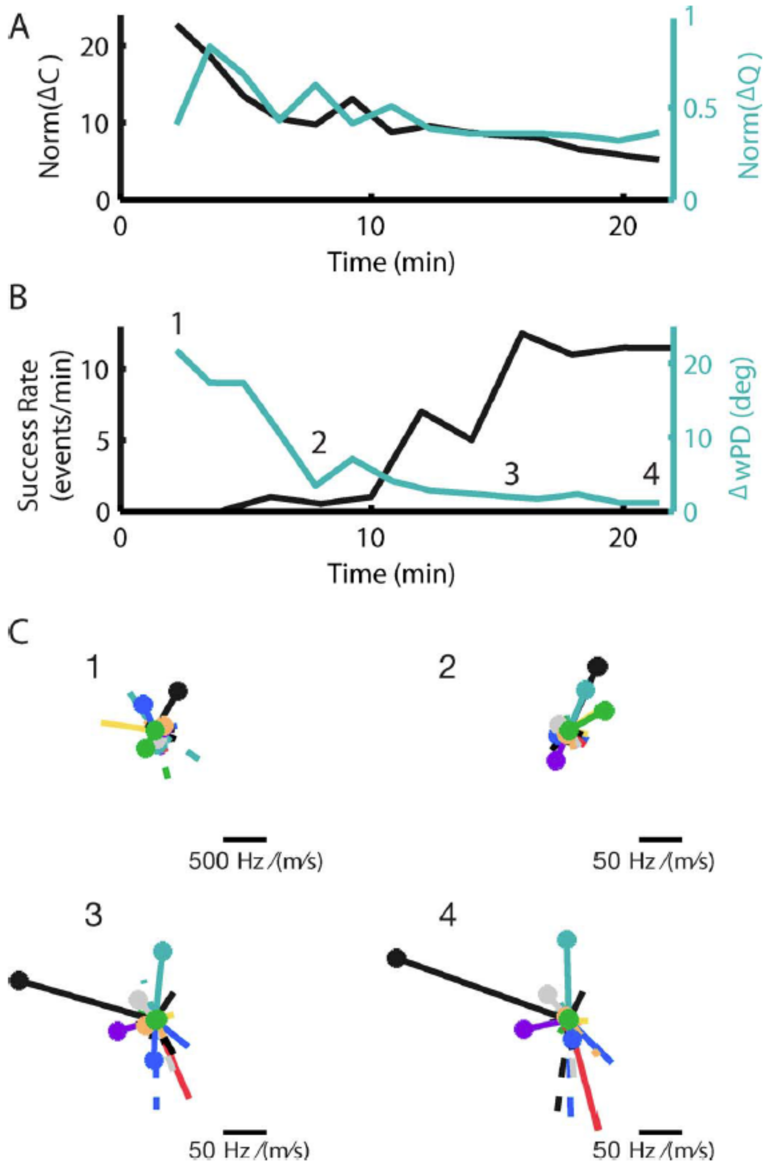


Figure 3.3: Kalman filter decoder evolution during SmoothBatch closed-loop decoder adaptation for one representative session seeded with baseline activity (same session as shown in Fig. 2). (A) The matrix norm of changes in the C and Q matrices between decoder updates converges as SmoothBatch continues. (B) The mean change in units' velocity preferred directions weighted by relative tuning strength (ΔwPD) also converges. The convergence of the velocity tuning solution parallels behavioral performance improvements. (C) Velocity tuning decoder evolution for example time-points (1–4, indicated in B). The tuning of each unit is represented in polar coordinates, where the line length indicates modulation-depth and its orientation shows the preferred direction. Note that 1 (upper left) is on a different scale than 2–4.

3.3.3 Performance Improvement's Dependence on Decoder Seeding

SmoothBatch CLDA improved BMI performance independent of the initial decoder seeding method. Each seeding method had a varying amount of offline predictive power, as summarized in Figure 3.4A. VFB and ipsi seedings contained the most information about all kinematic variables while baseline and shuffled seeds had little to no predictive power. Despite differences in offline power of the decoders, all seeds reached similar final performance after adaptation. Figure 3.4B compares early success rates (first estimate within the session) to the maximum rate achieved, separated by seed type. A Kruskal–Wallis analysis of variance (KW-ANOVA) showed no significant effect of seed type on maximum performance ($p > 0.05$). All seeds achieved similar reach trajectory precision as well. Figure 3.4C shows the average ME for the first and last 100 trials within the SmoothBatch session, as well as average estimates from arm movements for comparison. The final reach kinematics parameters (MV, ME, nor reach time) showed a significant dependence on seed type (KW-ANOVA, $p > 0.05$). Performance improvements did, however, occur on different time-scales depending on the initial decoder seed. Figure 3.4D shows the amount of time it took for performance to exceed a threshold of 8 successful trials/min (left) and reach the maximum rate (right) for each session, sorted by seed decoder type. A KW-ANOVA showed that time to reach the 8 trials/min threshold depended on seed type. VFB seedings, which contained the strongest offline decoding power, improved significantly faster than all other types ($p < 0.05$). No other significant differences among groups were found. The offline R^2 power of the seed decoder showed weak correlations with time to reach threshold (Figure 3.4E). Pooling all data, significant correlations were only found for p_x and p_y ($R = -0.329$, $p = 0.014$; $R = -0.393$, $p = 0.003$, respectively). The time to reach threshold performance also showed no clear relation to the distance (ΔwPD) between the seed and final decoders (Figure 3.4F), suggesting that algorithmic convergence/step size alone does not limit improvement. The time to achieve maximum performance, however, showed no significant differences across groups (Fig. 4(D); KW-ANOVA, $p > 0.05$) and no significant correlation with offline prediction power or ΔwPD (not shown; $p > 0.05$).

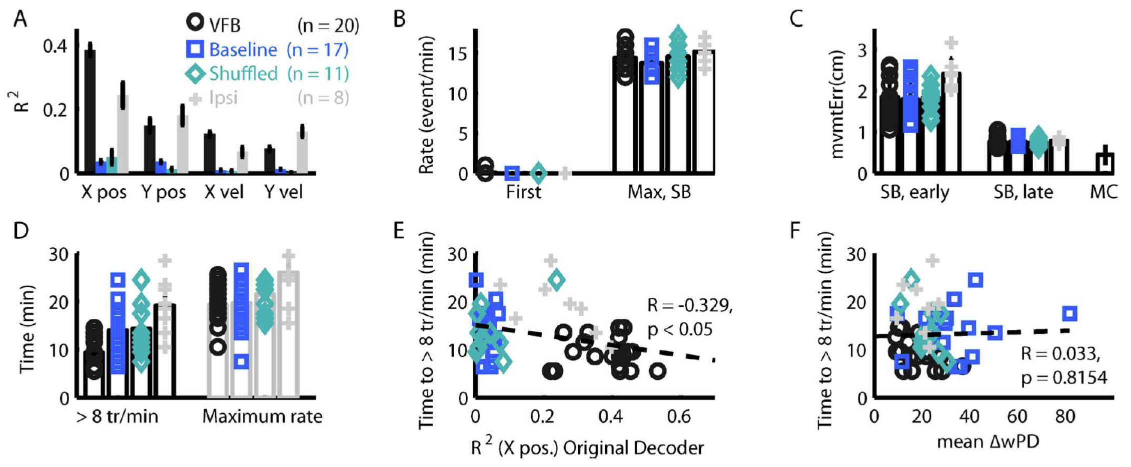


Figure 3.4: SmoothBatch performance improvement across different decoder seeding types. (A) Offline decoder prediction power for endpoint position and velocity (x and y components) for all decoder seed types. Bars indicate mean across all performed sessions, error bars show standard error of the mean. (B) Initial and maximum performance rates of SmoothBatch sessions, separated by seed types. Points show individual sessions; bars and error bars show mean and standard error of the mean, respectively. (C) Average movement error (ME) for the first (early) and last (late) 100 trials during SmoothBatch adaptation, separated by seed condition. Average ME across eight arm movement sessions (arm) are shown at right for comparison. Format as in panel C. (D) SmoothBatch performance improvement times (time for successes per minute to exceed 8 trials/min) and time to achieve the maximum performance rate for all sessions and all decoder seed types. Format as in panel C. (E) Relationship between seed decoder predictive power (R^2 for x-component of position) and time to reach threshold performance. Points are individual sessions, color-coded by seed type, and line shows linear regression ($R = -0.329$, $p < 0.05$). (F) Relationship between relative distance between seed decoder and final decoder's velocity tuning models (ΔwPD) and time to reach threshold performance. Points are individual sessions, color-coded by seed type, and line shows linear regression (not significant).

As seen by the wide spread in ΔwPD (3.4F), few initial seed decoders, even ones that contained significant offline decoding power, had velocity PD assignments close to those of the final SmoothBatch solution (note that on average, significant performance improvements were not seen until $\Delta w\text{PD} < 6.52\pm4.48°$). A KW-ANOVA showed that ΔwPDs were significantly larger for baseline seeds than VFB ($p < 0.05$); no other groups showed statistically significant differences.

3.4 Discussion

We found that the SmoothBatch CLDA algorithm, which updates on an intermediate timescale (1–2 min), was able to improve closed-loop BMI performance rapidly and robustly, independent of initial decoder seeding. The subject was readily able to achieve proficient center-out task performance (88.04±5.3% success rate, 1.23±0.16 s reach times, 0.771±0.088 cm ME, and 0.593±0.067 cm MV) across 56 experimental sessions. Importantly, improvements were rapid (success rates exceeding 8 trials/min in an average of 13.1±5.5 min), and occurred reliably in all sessions despite decoders being seeded using four different methods. Performance was also maintained upon ceasing adaptation and holding the decoder fixed, with no drop in task performance or reach times, and only slight (12%) increases in ME and MV.

Comparison across BMI studies represent a current challenge in the field [21]. Studies use incongruous task designs and different types of neural activity, obscuring direct comparisons. The use of different animal models (e.g., animals with arms free to move versus partial or full movement restriction) across studies may further confound cross-study comparisons. Furthermore, the field has not established standard metrics for performance evaluation, limiting the ability to quantitatively compare results. In an effort to make this work more readily comparable to future studies, we report performance metrics commonly used to evaluate pointing-devices like computer mice (MV and ME). Comparisons to one human BMI study (not explicitly utilizing CLDA or across-session learning) reporting MV and ME metrics shows that SmoothBatch has markedly reduced mean ME and MV relative to their best results ([17] MV: 1.2 cm, SmoothBatch 0.587 cm; [17] ME: 2.3 cm, SmoothBatch: 0.759 cm). Moreover, MV and ME results for SmoothBatch approach those of natural movements (MV: 0.35±0.17 cm, ME: 0.44±0.20 cm; mean±std across 2025 trials in eight arm movement session).

The success of SmoothBatch is a clear testament to the power of the CLDA principle [2, 5, 30, 31, 34, 35]. In the limit of long batch sizes and $\alpha = \beta = 0$, Gilja et al.'s batch algorithm is a special case of SmoothBatch. The longer data batches used by Gilja et al. allow for more accurate parameter estimation, eliminating the need for averaging. However, waiting to collect sufficient data for a single model update greatly reduces the update frequency, thus reducing the rate at which the user sees performance improvements. In the work of Gilja et al., using decoders seeded from contralateral arm movements and BMI performed during overt arm movements, their subjects were able to attain $> 90\%$ task performance with the seed decoder before any CLDA intervention. This initial high level of performance allows for a significant improvement in reach kinematics after a single batch-based decoder update [34, 35].

SmoothBatch, however, is ideally suited to address a different problem—how can we use CLDA to generate a high-performance BMI given an initial decoder with severely limited closed-loop performance? CLDA may be a particularly useful tool in clinical applications, where many factors could limit initial closed-loop performance (see below). In these applications, collecting sufficient data for a batch-based decoder update is challenging. We recently showed that when starting from poorly performing seed decoders, as many as 3–5 6-min batch updates were required to produce adequate closed-loop BMI performance [32]. Furthermore, the slow update rates reduce subject motivation, since subjects were required to use poorly performing decoders for 6–10 min [32]. Though SmoothBatch sacrifices accuracy in each parameter estimation step due to a smaller batch-size than the Gilja et al. algorithm, the increased decoder update frequency provides the user with more rapid feedback and may facilitate faster improvement.

The progress of improvements seen in SmoothBatch also further suggest that using CLDA may involve a bootstrapping or co-adaptation process between the brain and decoder when starting from limited closed-loop BMI performance. SmoothBatch required multiple decoder update steps before performance improved significantly, progressing to gradually let the subject initiate trials, then reach a few targets, and finally reach all targets. This progression may be due in part to the exponentially weighted averaging and short data-batches used. However, the batch-sizes and half-lives used here correspond to relatively aggressive adaptation rates ($\alpha = \beta \in [0.55, 0.77]$) and on average, more than one half-life's worth of updates occurred before adequate performance was achieved. Moreover, we recently showed a similar progression using purely batch-based algorithms [32]. This suggests that creating an optimal decoder in a single update step may be highly infeasible when starting from severely limited closed-loop performance. Instead, intermediate adaptation time-scale algorithms like SmoothBatch provide the subject with more rapid feedback and gradually adapt the decoder, facilitating a co-adaptation process and yielding rapid performance improvements.

SmoothBatch uses a sliding average of model parameter estimates based on small amounts of data, which is similar to the approach used by Taylor et al. [5]. They also show that their co-adaptive algorithm is able to yield proficient closed-loop BMI control when seeded with randomly initialize parameters. This suggests that an intermediate time-scale adaptation approach may ideal for such applications.

One concern in using an adaptive algorithm is convergence. Our previous work [32] showed that a real-time adaptive CLDA algorithm [37] improved performance and increased subject motivation via rapid updates, but temporally overfit the data causing performance degradation after decoder adaptation ceased. Here, we show that SmoothBatch rapidly converges (Figure 3.3). The subject was also readily able to maintain task performance after decoder adaptation ceased, showing that the algorithm converged towards a general task solution and avoided overfitting. C and Q matrices did exhibit small fluctuations (i.e., changes did not fully converge to zero, see Figure 3.3), however these did not noticeably effect task performance, suggesting they may only be driven by behavioral and neural variability. It may also be possible to optimize batch size and half-life parameters to further improve convergence. For instance, increasing the batch size as a function of task performance could allow the algorithm to take large, rapid steps while the decoder is “far” from a high-performance decoder, and smaller steps to fine-tune the model as it nears the final solution. Additional work to fully optimize the batch and half-life parameters may also yield yet faster improvement speeds.

The idea of CLDA involving a co-adaptation between the subject and decoder is further implied by the fact that SmoothBatch's improvement rate depended upon the initial decoder seed. While SmoothBatch was able to improve performance regardless of the seed, decoders seeded with neural activity during VFB improved more rapidly than arbitrary (baseline and shuffled) or ipsilateral arm-movement (ipsi) seeds. VFB seeds had the highest level of offline decoding power, consistent with observations that imagined/viewed movements evoke motor cortex activity [38–40]. Times to achieve a threshold of eight successful trials/min were weakly correlated with the decoder's offline prediction power, suggesting that the algorithm can more readily improve performance if initiated with decoder with some movement decoding power.

However, our results also suggest that offline prediction power is not the sole factor in how quickly SmoothBatch can improve performance. Ipsi decoder seeds had moderate offline prediction power, greater than that of baseline and shuffled seeds, consistent with findings of population-level representations of ipsilateral arm movements in motor cortex [29]. Yet, no statistically significant differences in improvement times were detected between ipsi, baseline or shuffled seeds. Furthermore, the times to achieve maximum performance rates were uncorrelated with offline prediction power. This is consistent with the Taylor et al. finding that the speed of adaptation did not differ when their co-adaptive population vector algorithm was initialized with PD estimates from arm movements or with random assignments [5].

Interestingly, we also found that on average the velocity tuning models for all seeds were "far" from that of the final decoder, despite containing varying degrees of offline prediction Figure 3.4D; only baseline seeds had significantly larger ΔwPD than VFB). This combined with the observation that offline prediction power alone is not indicative of SmoothBatch improvement rates, is consistent with the emerging notion that closed-loop BMI performance is not necessarily related to open-loop predictions [23, 24]. The many differences between BMI and natural movement — for instance congruent proprioceptive feedback is absent during BMI — may significantly alter the neural activity of the motor networks [14, 40]. Performance improvements in SmoothBatch did not appear to be limited by relative distance between initial and final decoders (Figure 3.4D). However, the difference between the subject's evoked neural activity between arm movements and closed-loop control would influence his performance in BMI, possibly affecting this co-adaptation process by influencing user strategy (e.g., causing frustration). We did see noticeable variability in improvement times across all decoder seed types, which could be related to subject strategies/motivation and the co-adaptation process. Additional work is needed to explore the exact mechanisms underlying performance improvements and the aspects of a seed decoder that influence final improvement. Such insights could yield further reductions in performance improvement times.

In conclusion, here we show that SmoothBatch can readily improve performance in a relatively short time, even with ill-conditioned seed decoders. Additional work to explore SmoothBatch's operation—the influence of adaptation parameters and the subject-decoder co-adaptation process—could also further improve the algorithm's operation. The ability to quickly and robustly generate high-performance BMIs independent of the subject's initial closed-loop BMI performance could be useful for clinical applications. For paralyzed patients, seeding decoders with arm movements is not feasible. Many patients will also lack proprioceptive feedback during closed-loop BMI operation, potentially limiting their performance [14]. Furthermore, noninvasive human BMI studies have shown significant inter-subject variability in the ability to volitionally modulate neural activity, which greatly impacts their ability to operate BMIs [41]. All of the decoder seeding methods presented here are compatible with many patient populations: VFB, baseline, and shuffled decoders could be created for any patient regardless of motor disability, and ipsi seedings could be used for unilaterally paralyzed patients such as stroke victims. Our work shows that SmoothBatch CLDA, an algorithm that updates the decoder on an intermediate time-scale during closed-loop operation, can reliably improve BMI performance in all of these cases.

Chapter 4

Design considerations for CLDA algorithms

The work presented in this chapter was performed in collaboration with Amy Orsborn, Helene G. Moorman, and Jose M. Carmena, and was published in Neural Computation [42].

4.1 Introduction

The concept of Closed-Loop Decoder Adaptation (CLDA) has shown great promise as a mechanism to both improve and/or maintain closed-loop BMI performance. A wide range of CLDA algorithms have been developed and tested for various purposes (see Table 4.1 for a sampling of prior work). While some CLDA algorithms strive to improve control accuracy when limited information is available to create an initial decoder [28], other algorithms aim to maintain the control accuracy of already high-performing decoders [43]. While these algorithms all represent implementations of CLDA, they are designed to operate in characteristically different ways. Indeed, when developing a CLDA algorithm, there are many different design choices to be made, including choosing the time-scale of adaptation, selecting which decoder parameters to adapt, crafting the corresponding update rules, and designing CLDA parameters. These decisions may significantly impact an algorithm's performance. For instance, a CLDA algorithm's ability to improve closed-loop BMI performance may be particularly sensitive to the time-scale at which it adapts decoder parameters [28, 32]. To date, little work has been done to explore how algorithms' design choices influence their performance. Similarly, few techniques have been established to assess algorithms' convergence properties. Ideally, CLDA algorithms should be designed to make the decoder's parameters converge rapidly to or maintain values that are optimal for high-performance BMI control. While conducting closed-loop BMI experiments is ultimately the only conclusive way to evaluate a CLDA algorithm's convergence properties, such experiments are lengthy and costly. Although closed-loop simulation methods [36, 44, 45] could potentially be used to study convergence, these tools have not yet been used for this purpose. Aside from these tools, there are currently limited methods for exploring the parameter space and predicting the convergence properties of a prototype CLDA algorithm before experimental testing. Better understanding how different design elements of a CLDA algorithm influences its closed-loop performance and convergence will be critical to developing high-performance BMIs.

In this chapter, we present a general framework for the design and analysis of CLDA algorithms. First, we identify and explore a core set of important design considerations that frequently arise when designing CLDA algorithms. For instance, we evaluate the effects of an algorithm's time-scale of adaptation, which can significantly impact both the subject's level of engagement and the rate of performance improvements. Next, we underscore the importance of selective parameter adaptation by demonstrating why adapting certain decoder parameters can lead to undesired effects on performance. We then illustrate how smooth parameter updates can help mitigate the impact of unreliable batches of data, especially when initial performance is limited. Finally, we stress the importance of designing CLDA parameters that are readily interpretable, a property which can help avoid conducting large parameter sweeps and simplify parameter selection in experimental work.

We then introduce mathematical convergence analysis, using measures such as mean-square error (MSE) or KL divergence, as a useful precursor to closed-loop experiments that can further inform CLDA design and constrain the necessary experimental testing. We choose the SmoothBatch CLDA algorithm [28] as a case study to demonstrate the utility of our analysis. Our analysis predicts that SmoothBatch's MSEs converge exponentially to steady-state values, and that both these values and the rate of convergence are independent of the decoder's seeding method. Experimental data from two non-human primate subjects across 72 sessions serves to support our predictions and validate our convergence measures. Finally, guided by our convergence predictions, we propose a specific method to improve the SmoothBatch algorithm by allowing for time-varying CLDA parameters. Overall, our study of CLDA design considerations sheds light on important aspects of the design process, while our mathematical convergence analysis serves as a useful tool that can inform the design of future CLDA algorithms prior to conducting closed-loop experiments.

Table 4.1: Sampling of prior work on CLDA algorithms. The CLDA focus represents how each particular algorithm was used in its respective study.

CLDA Algorithm	Decoder	CLDA focus	Form of CLDA updates
[5]	Population vector algorithm	improving from low performance	iterative refinement of tuning properties based on current performance, errors from most recent block, and best weights from a past block
[2]	Kalman filter	improving from low performance	batch maximum likelihood estimation using data from a trial-by-trial sliding block
[30]	Kernelized ARMA	maintaining performance	replacement of old examples in training set with new examples
[35]	Kalman filter	improving from moderate-to-high performance	batch maximum likelihood estimation
[46]	Wiener filter	maintaining performance	remapping of a lost neuron's decoder weights using next closest neuron's weights
[31]	Time-Delayed Neural Network	improving and maintaining performance	decoder adapted within a reinforcement learning (actor-critic) framework using an error signal representing the gradient of user's reward expectation
[43]	Kalman filter	maintaining performance	Bayesian regression self-training updates
[37] Adaptive KF	Kalman filter	improving from low performance	stochastic gradient descent
[28] SmoothBatch	Kalman filter	improving from low performance	weighted average of current parameters with new batch maximum likelihood estimates
[18]	Kalman filter	improving from low performance	iteratively updated filter parameters during successive closed-loop calibration blocks

4.2 Methods

4.2.1 Experimental procedures

CLDA algorithms were experimentally tested using intracortically recorded neural activity from two non-human primates. Electrophysiology and behavioral protocols have been previously described in [32] and [28]. Briefly, 128-electrode microwire arrays (35 μm diameter, 500 μm wire spacing, 8×16 array configuration; Innovative Neurophysiology, Durham, NC) were implanted in both brain hemispheres of two adult male rhesus macaque monkeys. Arrays were positioned to target the arm areas of primary motor cortex (M1), and due to their size, extended rostrally into dorsal premotor cortex (PMd). Unit activity was recorded using a combination of two 128-channel MAP and OmniPlex recording systems (Plexon Inc, Dallas, TX). Single and multi-unit activity was sorted using an online sorting application (Plexon Inc, Dallas, TX), and only units with well-identified waveforms were used for BMI control.

Monkeys were head restrained in a primate chair and observed a task display via a computer monitor projecting to a semi-transparent mirror parallel to the floor. They were trained to perform a self-paced 2D center-out reaching task to 8 circular targets uniformly spaced around a circle. During BMI operation, the subjects' arms were confined within the primate chair as they performed the task by moving the cursor under neural control. Figure 4.1 shows an illustration of the task set-up and trial timeline. Reach targets were presented in a block structure of 8 targets with pseudo-randomized order within each block. Targets were spaced 7 cm (subject 1) or 6.5 cm (subject 2) away from the center. Target sizes of 1.2–1.7 cm radius, center and target holds of 250–400 ms, and reach times of 3–5 s were used. All procedures were conducted in compliance with the National Institutes of Health Guide for Care and Use of Laboratory Animals, and were approved by the University of California, Berkeley Institutional Animal Care and Use Committee.

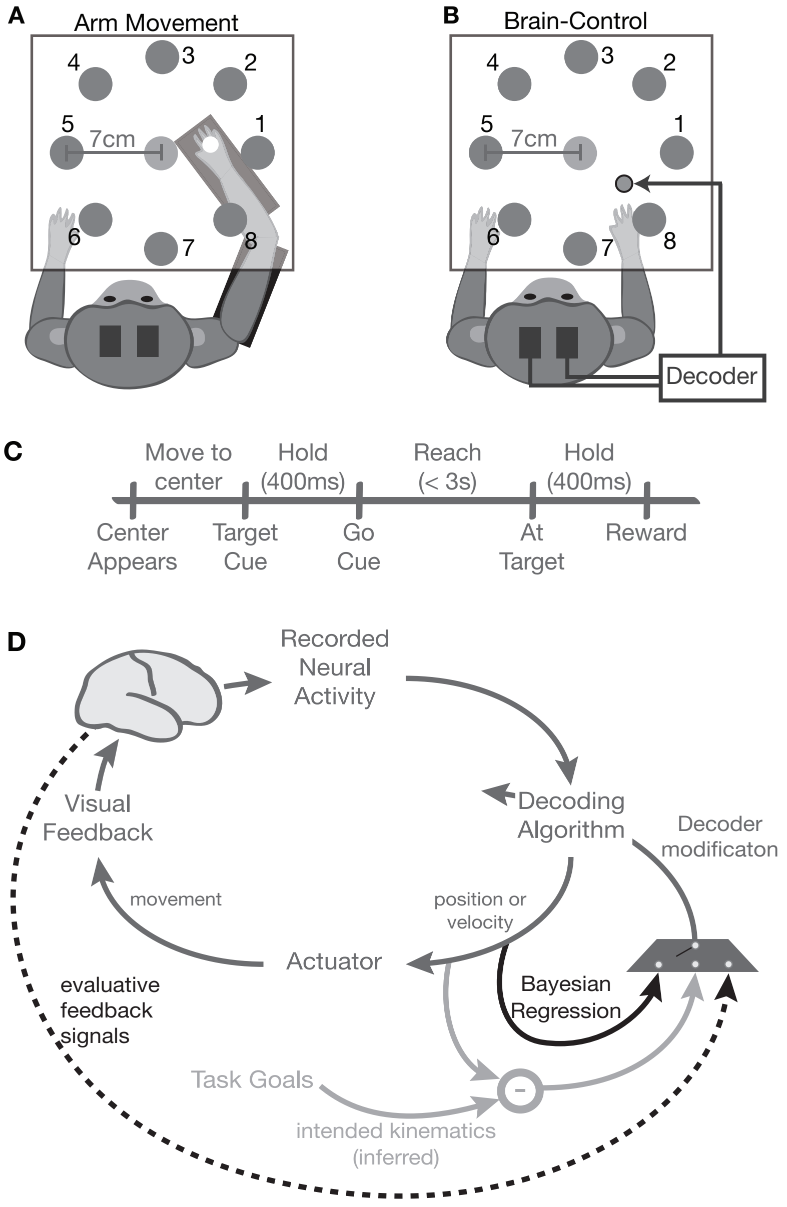


Figure 4.1: **Experimental set-up and Closed-Loop Decoder Adaptation (CLDA).** Illustration of the experimental task set-up for arm movements (A) and closed-loop BMI experiments (B), and trial timeline for the center-out task (C). Panel D illustrates the concept of Closed-Loop Decoder Adaptation. The decoder is modified as a subject uses it in closed-loop control (gray). The signals used to modify the decoder can be attained in several ways, including using 1) task-goals to infer a subject's intentions (orange), 2) Bayesian methods to self-train the decoder (green), or 3) neural signals (purple).

4.2.2 Decoding Algorithm

All experimental sessions used a Kalman filter (KF) as the decoding algorithm during closed-loop control, with binned neural firing rates as the choice of neural features. Online BMI control was implemented using PlexNet (Plexon Inc, Dallas TX) to stream neural data on a local intranet from the Plexon recording system to a dedicated computer running MATLAB (The Mathworks, Natick, MA). Neural firing rates were estimated with a 100 ms bin width. Neural ensembles of 16-36 (26.5 ± 4.45; mean±STD) neurons were used. Units were selected only based on waveform quality.

Each test of a CLDA algorithm used a decoder initialized using one of the five of methods described in Section 2.3:

1. Visual Feedback ($n = 32$)
2. Contra ($n = 2$)
3. Ipsi ($n = 8$)
4. Baseline ($n = 17$)
5. Shuffled ($n = 13$)

The structures of the KF state transition model parameters (A and W) were constrained during fitting to obey physical kinematics, such that integrating the velocity from one KF iteration perfectly explains position at the next iteration (see [27] for further details). As discussed below, these parameters were typically held fixed during CLDA. In order to allow for BMI control that would mimic natural arm movements as much as possible, A and W were fit for all seeding methods using a 30-minute data set of arm movements collected while the subject performed the center-out task in manual control. In experiments where A and W were adapted (see Section 4.3; *VFB* and *contra* seeds only), these matrices were initialized to their maximum-likelihood estimates calculated from the recorded kinematic data collected specifically for seeding. For seeding methods 1–4, the KF observation model parameters, C and Q, were seeded using batch maximum-likelihood estimates based on 8 minutes of neural and kinematic data.

4.2.3 KF CLDA Algorithms

In the following sections, we review some existing CLDA algorithms for KF decoders that will be compared within our analyses. These algorithms update the decoder as the subject performs the center-out task in closed-loop BMI control. For each algorithm, the subject's intended cursor kinematics are inferred from the observed kinematics using the method developed by Gilja et al. [27], which assumes that the subject always intends to move in a straight line towards the target, and rotates observed cursor velocities accordingly. For reasons discussed in Section 4.3, we typically only update the KF decoder's observation model parameters (see Section 4.3). As such, update rules for only the C and Q matrices are given below for each CLDA algorithm. (The Batch algorithm's update equations for A and W, when these parameters are also adapted, are presented in [28]). Each time new values for these decoder parameters are calculated, they are immediately used in the decoder as part of subsequent BMI control.

4.2.3.1 Batch maximum likelihood estimation

One paradigm for KF CLDA algorithms entails collecting data and processing the entire batch at once to update the decoder's parameters. In this approach, the updated C and Q matrices are set to their maximum likelihood estimates based on the batch of data. We refer to this method as the Batch algorithm. The Batch algorithm's update rules are

$$\hat{C} = YX^T\left(XX^T\right)^{-1} \tag{4.1}$$

$$\hat{Q} = \frac{1}{N}\left(Y-\hat{C}X\right)\left(Y-\hat{C}X\right)^T \tag{4.2}$$

where the Y and X matrices are formed by tiling N columns of recorded neural activity and kinematics, respectively. The size of the data batch is parametrized by the batch period $T_\mathrm{b} \triangleq N \cdot \mathrm{dt}$ (where dt is the time between KF decoder iterations).

4.2.3.2 Adaptive Kalman Filter

The Adaptive Kalman Filter (Adaptive KF or AKF; [37]) is a CLDA algorithm designed to update the KF's observation model parameters at every decoder iteration, which corresponds to every 100 ms in our experiments. If we let i index discrete decoder iterations, the Adaptive KF's update rules are

$$\begin{aligned} C^{(i+1)} &= C^{(i)} - \mu\left(C^{(i)}x_t - y_t\right)x_t^T \\ Q^{(i+1)} &= \alpha Q^{(i)} + (1-\alpha)\left(y_t - C^{(i+1)}x_t\right)\left(y_t - C^{(i+1)}x_t\right)^T \end{aligned}$$

where y_t and x_t represent neural firing rates and an estimate of the user's intended kinematics, respectively, at time t[1]. The update rule for C is derived by writing this matrix as the solution to an optimization problem, and then using stochastic gradient descent with step-size μ to iteratively make small corrections to it at every decoder iteration. The Adaptive KF's update rule for Q, on the other hand, is of heuristic form, and effectively represents a weighted average (with parameter $\alpha \in [0,1]$) of the current value of Q with a single-iteration estimate of Q.

[1]Note that here, t actually equals i. However, we choose to maintain both as separate indices in order to be consistent with other algorithms such as SmoothBatch, where they are different because the decoder is not updated at every iteration.

4.2.3.3 SmoothBatch

The SmoothBatch CLDA algorithm [28] periodically updates the KF decoder's observation model (C and Q matrices) by performing a weighted average of the current parameters with those estimated from a new batch of data using the Batch algorithm. The observed neural activity and intended cursor kinematics are collected over one batch period. This batch of data is then used to construct new Batch estimates, $\hat{C}$ and $\hat{Q}$, of the C and Q matrices using (4.1) and (4.2). Finally, the observation model parameters are updated using a weighted average:

$$C^{(i)} = (1-\rho)\,\hat{C}^{(i)} + \rho C^{(i-1)} \tag{4.3}$$

$$Q^{(i)} = (1-\rho)\,\hat{Q}^{(i)} + \rho Q^{(i-1)} \tag{4.4}$$

where i indexes discrete batch periods and the weighting parameter $\rho \in (0,1)$ controls the influence of $\hat{C}$ and $\hat{Q}$ on the new parameter settings.

4.3 CLDA Design Principles

The development of a CLDA algorithm involves making multiple crucial design decisions such as:

- how often to apply update rules (the time-scale of adaptation)
- whether or not to update all decoder parameters
- how to update decoder parameters (the actual parameter update formulas or "update rules")
- how to set CLDA algorithmic parameters (e.g., step-sizes or learning rates)

Ideally, these design choices should enable a CLDA algorithm to make the decoder's parameters converge rapidly to or maintain values that are optimal for high-performance BMI control. In the following sections, we explore these design choices by examining various aspects of the Batch, Adaptive KF, and SmoothBatch CLDA algorithms. We focus primarily on SmoothBatch, which has been demonstrated to improve BMI control accuracy even when initial performance is severely limited due to an unfavorable seeding [28]. For instance, with patients such as paralyzed individuals or amputees that would be likely clinical targets of BMI technology, seeding methods involving overt movements wouldn't be feasible, and therefore other methods that typically result in a lower level of initial performance would need to be used. Despite our focus on SmoothBatch, however, we will see that many of its design properties are likely to be shared by other CLDA algorithms, even when their underlying purpose is characteristically different.

4.3.1 Time-scale of adaptation

A CLDA algorithm's *time-scale of adaptation* refers to the frequency with which its update rules are applied to adapt the decoder's parameters. In a clinical setting with paralyzed patients, initial closed-loop BMI performance may be limited. As such, it is critical to develop CLDA algorithms capable of rapidly improving control accuracy regardless of initial performance. Given the subject-decoder interactions inherent in closed-loop BMI systems, the time-scale of adaptation may be particularly important in these situations. Indeed, previous work comparing two CLDA algorithms (the Batch and AKF algorithms) operating on different time-scales showed that the adaptation frequency can have a significant impact on CLDA performance [32]. We recently showed that the SmoothBatch algorithm, when operating on an intermediate time-scale (1-2 minutes), successfully and robustly improves performance independent of the initial decoder [28]. Here, we extend these results by comparing experimental data from the Batch, AKF, and SmoothBatch algorithms to highlight the role of adaptation time-scale.

All three CLDA algorithms were experimentally tested with one non-human primate subject[2], starting from limited task performance (Batch and AKF using *VFB* seeds, $n = 7$ and $n = 2$, respectively; SmoothBatch using *VFB*, *Ipsi*, *Baseline*, and *Shuffled* seeds, $n = 64$). For all algorithms, the state transition model (A and W) was held fixed. Sessions selected for comparison had comparable task settings and neural ensemble sizes. Figure 4.2A shows examples of the Batch, AKF, and SmoothBatch algorithms' performance in representative sessions. The evolution of task performance is illustrated with moving averages (75 trial window) of percentages of trial outcomes (i.e. success, reach-error, or hold error; upper panels) and rates of each task event (estimated via binning events in 120 s non-overlapping bins; lower panels). Each CLDA algorithm was able to improve performance, as evidenced by increases in success percentages and rates, and corresponding decreases in errors during adaptation. However, the algorithms differ significantly in how well the subject is able to maintain performance after adaptation ceases (i.e., when using a fixed decoder). For both the Batch and SmoothBatch algorithms, the subject readily maintained performance upon fixing the decoder. However, for the AKF, task performance (both success percentage and rate) dropped significantly after CLDA was ceased. Performance changes were assessed in two ways. First, we compared the maximum success rate during adaptation with the maximum rate during first 15 minutes of using the fixed decoder. Second, we compared the success percentage during the last 50 trials during adaptation to that of the first 50 trials with the fixed decoder. Figure 4.2D depicts the percent change in the success rate for all sessions, sorted by CLDA method. Batch ($n = 7$) and SmoothBatch ($n = 64$) showed no significant changes in performance (Wilcoxon paired test, $p > 0.05$). Note that a decoder update occurs between the adapt and fixed portion for the Batch algorithm, which may explain the slight (though not statistically significant) increase in performance observed. Only one AKF

[2]The Batch and AKF algorithms were not tested in the second subject. Two SmoothBatch sessions in subject 1 did not have sufficient data with the fixed decoder to assess the post-CLDA maintenance of performance, and thus were excluded from our analysis.

session had sufficient data with a fixed decoder to allow for comparison, but this session shows a very large drop in performance. Success percentages showed identical trends (Batch and SmoothBatch had no significant change; Wilcoxon paired test, $p > 0.05$; not shown).

The adaptation time-scale also has a significant influence on the rate of performance improvements. Improvement slopes were quantified by computing the change in success rate (or percentage) from the start of adaptation to the maximum achieved during adaptation, and dividing by the time required to achieve that maximum performance. Figure 4.2E shows the success rate improvement slope for all sessions sorted by CLDA algorithm. SmoothBatch shows significantly faster improvements than both Batch and AKF algorithms (Kruskal-Wallis analysis of variance, $p < 0.05$). The AKF improves most slowly, though not statistically significant from that of Batch. Identical trends were found when comparing percentage-based slope estimates.

One key aspect of the adaptation time-scale is the frequency with which the subject sees improvements in performance. This may be particularly important when starting from limited performance when the subject has little to no ability to control the cursor. For instance, the Batch algorithm requires the subject to persist in trying to use this poorly performing decoder for long periods of time, which may cause frustration, lack of motivation, or highly variable user strategies. By contrast, the AKF and SmoothBatch give the subject feedback more rapidly, keeping them more engaged in the task and potentially avoiding these user-related complications that could hinder the algorithm's progress. Subject engagement was assessed by calculating the number of trials the subject attempted to initiate (via entering the center target) during the first 10 minutes of CLDA. Here, only sessions with *VFB* seeds were used to avoid potential differences due to the initial decoder [28]. Figure 1F shows that the subjects initiated significantly more trials in SmoothBatch sessions than Batch sessions (Kruskal-Wallis analysis of variance, $p < 0.05$). AKF sessions similarly show more trial initiations, though not significantly different from Batch or SmoothBatch. This suggests that increasing the frequency of user feedback can indeed increase user engagement. Increased engagement, in turn, likely contributes to the CLDA algorithm's ability to improve performance. Together, these analyses show the importance of the CLDA algorithm time-scale when starting from limited initial performance.

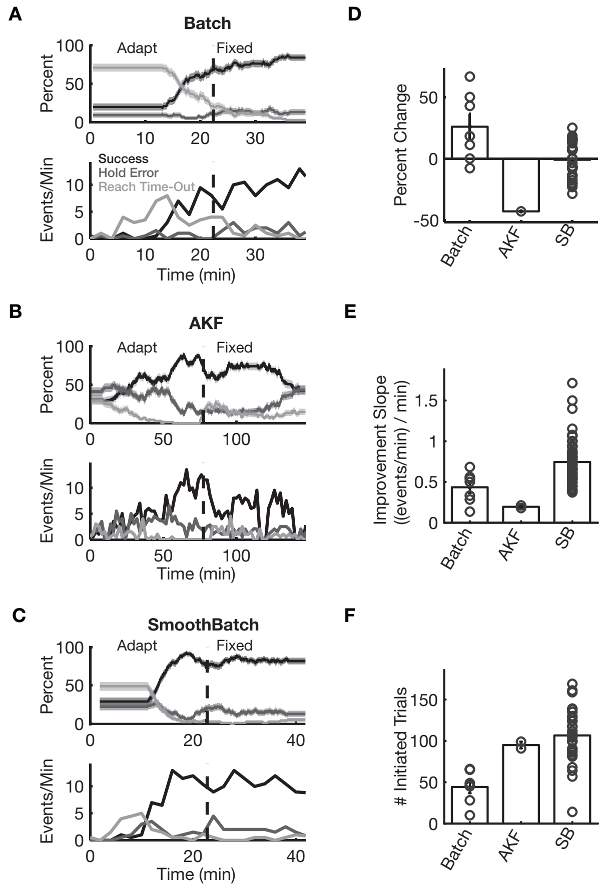


Figure 4.2: **Performance comparison of three CLDA algorithms that operate on different time-scales.** (A) Experimental performance for each algorithm calculated using both moving averages (75 trial window) of percentages of trial outcomes (i.e. success, reach-error, or hold error) and rates of each task event (estimated via binning events in 120 s non-overlapping bins). (D) Percent change in the success rate across all sessions, sorted by CLDA algorithm. (E) Improvement slope of the success rate for all sessions, sorted by CLDA algorithm. (F) Number of trials the subject attempted to initiate (via entering the center target) during the first 10 minutes of CLDA.

4.3.2 Selective parameter adaptation

BMI decoding algorithms are often characterized by several different parameters. For instance, a KF decoder is parametrized by four matrices, $\{A, W, C, Q\}$, each with different physical interpretations within a cursor control BMI system. A and W characterize the state transition model and represent cursor kinematics, while C and Q form the observation model that represents the mapping between neural activity and cursor movement. The KF combines these two models in a recursive algorithm to predict cursor movement, and its confidence in the models (related to the covariance matrices W and Q) ultimately determines their relative contributions to the final state estimate via a Kalman gain matrix, K.

A priori, a KF CLDA algorithm should adapt all four matrices since they all contribute to the final observed cursor movement. Indeed, previous work with Batch-based KF CLDA updated all KF parameters [27]. However, experimental evidence reveals that repeated adaptation of the full KF model, as is often needed when starting from limited initial performance (see Section 4.3.1), can have negative consequences. In particular, decoded cursor velocities show marked decreases with each successive update of KF parameters. Analysis of the matrices across consecutive CLDA updates reveals a clear explanation for these consequences. Figure 4.3A shows the changes (relative to their initial values) in the cursor velocity and norms of the Kalman gain (top), and KF matrices (transition model, middle; observation model, bottom) across Batch CLDA sessions where the full KF is adapted. Thick lines show the mean across all sessions ($n = 9$; 4 *VFB* seeds, 5 *Contra* seeds), and shading shows standard error. (Note that not all sessions contained the same number of Batch CLDA updates.) The mean predicted cursor speed decreased significantly with each CLDA update. Interestingly, the Kalman gain shows a parallel decrease in amplitude (estimated via the matrix norm). Inspecting the KF matrices reveals large and rapid decreases in W and significant increases in C. A and Q matrices, by contrast show no clear trends. Inspection of matrices shows that W typically decreases by an order of magnitude over the course of CLDA.

As noted above, the covariance matrix W plays a crucial role in determining the relative contributions of the state and observation models. Smaller W matrices indicate that the KF is more confident in its linear state transition model, and thus will give less weight to the neural data when forming a prediction (reflected by a smaller Kalman gain). We see that the state transition model, A, does not change significantly, but rather that the confidence in this model increases with each CLDA update. State transition models in BMIs (and models of natural arm movements) typically have damped velocity dynamics [27]. Thus, decreasing the weight on the system input — the neural data — inherently reduces cursor velocity. As such, the observed decreases in W likely contribute significantly to the cursor slowing found with CLDA updates.

To test this hypothesis, consecutive Batch updates were performed while holding the state transition model (A and W) fixed. The speed, Kalman gain, and KF matrices for these sessions are shown in figure 4.3B (formatting identical to 4.3A; $n = 19$, all *VFB* seeds). Here, we see that fixing A and W eliminates cursor slowing. In fact, cursor speeds show a slight increase, accompanied by small increase in the Kalman gain. Changes in the observation model are similar to changes in sessions where all matrices were adapted. This selective adaptation improved performance more successfully than full adaptation because it did not inherently limit cursor velocity.

While all KF matrices contribute to cursor movement, and thus are candidates for adaptation, it is important to consider their physical interpretations and the system as a whole before selecting what parameters to adapt. Given that the state transition model represents cursor kinematics, intuition suggests that the model should not change dramatically. In fact, the A matrices show little to no change, as expected. However, the cursor movements used for decoder parameter updates are themselves generated by a KF with kinematics specified by the decoder. Thus, the KF becomes increasingly confident in its model of cursor movement, driving decreases in W and corresponding decreases in cursor speed. If we instead interpret the KF as having a fixed model of cursor kinematics, and only adapt the mapping between the subject's neural activity and cursor movement, we avoid this problem of recursively refitting a dynamics model on itself. For this reason, SmoothBatch only modifies the C and Q matrices of the KF. Similarly, other CLDA algorithms for the KF update only these parameters as well [43].

BMI decoders typically contain multiple parameters, and therefore deciding which parameters to update is an essential design choice for CLDA algorithms. Our experiments have demonstrated that CLDA algorithms for KF decoders should not adapt the state transition model parameters, as this leads to reductions in cursor velocity and other undesirable effects. More generally, these results may suggest that it is undesirable to update any decoder parameters that represent a prior model on movement. Instead, CLDA algorithms should primarily focus on adapting parameters that represent the mapping between the subject's neural activity and their intended movements.

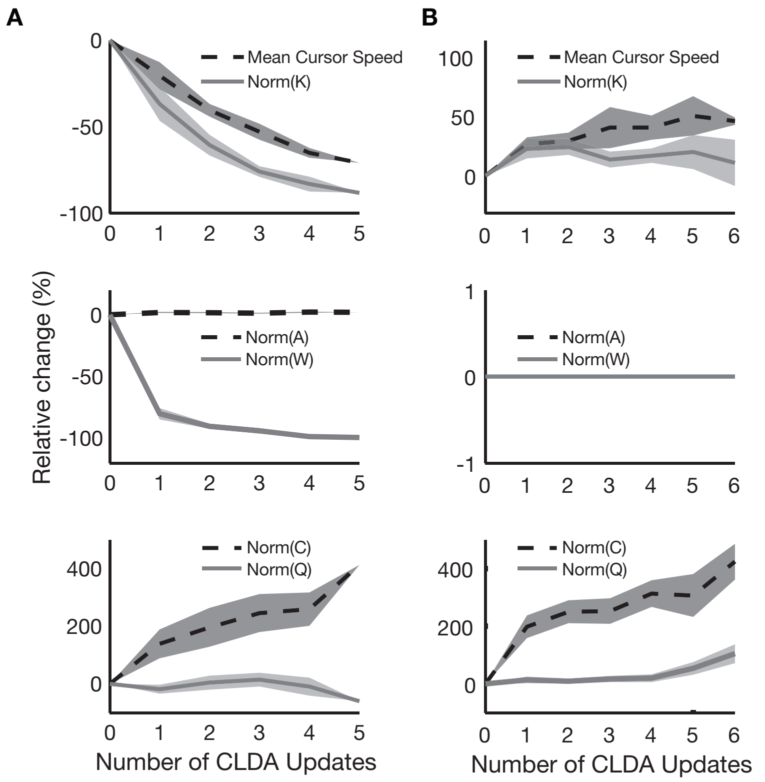


Figure 4.3: **Advantages of CLDA with selective parameter adaptation for a Kalman filter decoder.** Changes (relative to their initial values) in mean cursor speed and norm of the Kalman gain (top row), and KF model matrices (middle and bottom rows) across Batch CLDA sessions. (A) Adaptation of all KF parameters $\{A, W, C, Q\}$ leads to dramatic decreases in cursor speed. (B) Selective adaptation of only the KF state observation model parameters $\{C, Q\}$ avoids undesirable decreases in cursor speed.

4.3.3 Smooth decoder updates

CLDA algorithms typically collect and process real-time data, such as the user's neural activity and BMI cursor movements, to form parameter updates that will improve performance. However, occasionally an algorithm may encounter unreliable data that, if not properly addressed, can have a negative impact on performance. For instance, if the BMI user is distracted or not paying attention for a short time during data collection, the parameter updates formed from that data could be inaccurate. Such unreliable data can be especially concerning for certain types of CLDA algorithms, such as the Batch algorithm. In these types of algorithms, new parameter estimates completely overwrite and replace the decoder's current parameters at every CLDA update. Whenever the algorithm encounters unreliable data, current decoder parameters are replaced with new, inaccurate estimates, and the BMI user could struggle to adapt accordingly. One potential solution to this problem is to detect unreliable data and reject it, using either an automated algorithm or manual supervision. However, for CLDA algorithms that aim to improve control accuracy when starting from low performance, most initially collected data is at least to some degree "unreliable". For example, a 2D BMI cursor's movements can often appear random and not goal-directed when initial control is severely limited. In these situations, the subject could either be (unsuccessfully) attempting to control the cursor, or simply distracted and disengaged from the task. Since it is difficult to distinguish between these cases and reject the unreliable data, a CLDA algorithm has no choice but to operate on the entire dataset. In addition, since the cursor in this example may not explore the full workspace, there may not be sufficient variability in the corresponding collected batch of data to form accurate parameter estimates that could enable cursor control in all directions.

For CLDA algorithms that aim to operate from limited initial performance, one proven solution to encountering unreliable data is to perform smooth decoder updates. For instance, the SmoothBatch algorithm [28] is designed to facilitate gradual updates of parameters, rather than completely overwriting them. SmoothBatch performs a weighted average of current parameters with new parameter estimates, which helps prevent a single unreliable batch of data from causing drastic parameter changes that could impede improvements in performance. If fact, it can be shown that SmoothBatch's update rules can be formally interpreted as the solutions to the following optimization problems:

$$C^{(i)} = \arg\min_{C} \left[(1-\rho) \left\| C - \hat{C}^{(i)} \right\|_{\mathrm{F}}^{2} + \rho \left\| C - C^{(i-1)} \right\|_{\mathrm{F}}^{2} \right] \tag{4.5}$$

$$Q^{(i)} = \arg\min_{Q} \left[(1-\rho) \left\| Q - \hat{Q}^{(i)} \right\|_{\mathrm{F}}^{2} + \rho \left\| Q - Q^{(i-1)} \right\|_{\mathrm{F}}^{2} \right] \tag{4.6}$$

where $\|M\|_F$ denotes the Frobenius norm of a matrix M, which is defined as:

$$\|M\|_F \triangleq \sqrt{\sum_i \sum_j |m_{ij}|^2} = \sqrt{\mathrm{Tr}\,(M^T M)}. \tag{4.7}$$

The cost functions in (4.5) and (4.6) are designed to meet two competing objectives: producing updated parameters that are near the batch maximum likelihood estimates, yet still close to the previous parameter setting. SmoothBatch's update rules therefore achieve a balance of both objectives that can be adjusted as desired by setting the weighting parameter $\rho \in (0, 1)$. For instance, as $\rho \to 1$, the cost functions increasingly penalize deviations from the previous parameter setting, $C^{(i-1)}$ and $Q^{(i-1)}$, forcing SmoothBatch to make very smooth parameter updates. As $\rho \to 0$, the cost functions instead penalize deviations from the Batch parameter estimates, $\hat{C}^{(i)}$ and $\hat{Q}^{(i)}$. In the special case of $\rho = 0$, SmoothBatch reduces to the Batch algorithm:

$$C^{(i)} = \hat{C}^{(i)} \qquad \text{and} \qquad Q^{(i)} = \hat{Q}^{(i)}.$$

Therefore, SmoothBatch is effectively a regularized version of the Batch algorithm that directly aims to achieve smooth parameter updates. When initial BMI performance is limited and unreliable data is difficult to avoid, CLDA algorithms like SmoothBatch that ensure smoothness in their parameter updates can avoid large, disruptive parameter changes. By mitigating the impact of unreliable batches of data on performance, CLDA algorithms with smooth parameter updates can help ensure that a high level of BMI performance will eventually be reached.

4.3.4 Intuitive CLDA parameters

CLDA parameters refer to the step-sizes, learning rates, and other algorithmic parameters of a particular CLDA algorithm. The ability of a CLDA algorithm to effectively facilitate BMI performance improvements will greatly depend on the settings of these CLDA parameters. Note that these are distinct from decoder parameters such as the KF model matrices $\{A, W, C, Q\}$. A common approach to identifying a favorable setting of algorithmic parameters is to conduct a rigorous parameter sweep. While this technique works well offline, it is often infeasible in an online experimental setting, especially if any CLDA parameters lack intuitive interpretations. If there is no reference for what a parameter's order of magnitude should be, the parameter test space increases significantly, making a vast parameter sweep time-consuming or even infeasible. Given the inherent time constraints involved when performing closed-loop BMI experiments, this approach quickly becomes impractical.

In contrast, designing CLDA parameters with intuitive interpretations can greatly simplify parameter selection. As an example, the SmoothBatch algorithm's weighting parameter ρ is designed to implement a clear and intuitive linear trade-off between achieving the maximum likelihood estimates of parameters and maintaining the current parameter setting.

Since ρ is restricted to the range $[0, 1]$, the experimenter can perform a much more focused parameter sweep in order to identify an appropriate value. In addition, the clear interpretation of both boundary values helps provide an intuitive feel for the parameter. Moreover, another useful interpretation of ρ can be obtained by recursively expanding SmoothBatch's update rule (e.g., for C):

$$\begin{aligned} C^{(i)} &= (1-\rho)\,\hat{C}^{(i)} + \rho C^{(i-1)} \\ &= (1-\rho)\,\hat{C}^{(i)} + \rho\left((1-\rho)\,\hat{C}^{(i-1)} + \rho C^{(i-2)}\right) \\ &= \ldots \\ &= (1-\rho)\left(\hat{C}^{(i)} + \rho\hat{C}^{(i-1)} + \ldots + \rho^{i}\hat{C}^{(1)}\right) + \rho^{i} C^{(0)} \\ &= (1-\rho)\sum_{j=0}^{i} \rho^{j}\hat{C}^{(i-j)} + \rho^{i} C^{(0)} \end{aligned} \tag{4.8}$$

where $C^{(0)}$ represents the initial (seed) value of the matrix. From this expansion, we see that SmoothBatch's update rules effectively implement an exponentially-weighted moving average of past maximum likelihood estimates. In other words, the weights attached to past maximum likelihood estimates experience exponential decay. Thus, in addition to being a weighting parameter, ρ also has an intuitive interpretation as the weighting factor of an exponentially-weighted moving average.

Furthermore, this reformulation reveals another intuitive parametrization of the algorithm. Since the maximum likelihood estimates $\left\{\hat{C}^{(i)},\, i \geq 1\right\}$ occur T_b seconds apart, one can define a "half-life" h as

$$\rho^{h/T_\text{b}} = \frac{1}{2}. \tag{4.9}$$

The half-life h represents the time it takes for a previous maximum likelihood estimate's weight in the decoder to be reduced by a factor of two. Therefore, the SmoothBatch CLDA parameters $\{T_\text{b}, \rho\}$ can be conveniently reparametrized as $\{T_\text{b}, h\}$. Since the batch period and the half-life both have units of time, this reparametrization provides yet another intuitive representation of SmoothBatch's CLDA parameters.

Overall, the multiple interpretations of ρ and its ability to be reparametrized into the half-life demonstrate the intuitive nature of SmoothBatch's algorithmic parameters. When a CLDA algorithm is designed like SmoothBatch to have CLDA parameters with clear and intuitive interpretations, appropriate values of these parameters will be simpler to choose in an experimental setting, thereby maximizing the algorithm's ability to rapidly improve closed-loop performance.

4.4 Convergence Analysis

Ideally, CLDA algorithms should be designed to make the decoder's parameters converge rapidly to or maintain values that are optimal for high-performance BMI control. Presumably, for every decoding algorithm there exist optimal settings of decoder parameters that allow the user to maximize BMI performance. Since decoder parameters are typically continuous-valued, it is unrealistic to expect a CLDA algorithm to achieve an optimal parameter setting exactly. Instead, a CLDA algorithm should aim to make decoder parameters converge close to an optimal setting. Analyzing the design of a CLDA algorithm with respect to this objective raises multiple important questions. How close to this optimal setting will decoder parameters converge? How fast does convergence occur? How do CLDA parameters affect convergence, and what tradeoffs are inherent in the choices of these parameters? How does the decoder's seeding method affect convergence? While conducting closed-loop BMI experiments is ultimately the only definitive way to address these questions conclusively, such experiments are often lengthy and costly.

Developing a method to predict and analyze the convergence properties of CLDA algorithms, before testing them experimentally, would greatly facilitate and expedite the CLDA algorithm design process. While closed-loop simulation methods could potentially be used to study convergence, these tools have not yet been used for this purpose [36, 45]. In this section, we explore a novel paradigm — mathematical convergence analysis under reasonable model assumptions. Using mean-square errors and KL divergence to quantify a CLDA algorithm's convergence ability, we can analyze how an algorithm will perform under different settings of its CLDA parameters. To demonstrate the utility of our analysis, we apply our measures to the SmoothBatch algorithm to make predictions about and gain insight into its convergence properties. Experimental testing with two monkeys performing a BMI task across 72 sessions validates the utility of our convergence measures. These results demonstrate that mathematical convergence analysis is an effective analytical tool that can ultimately inform the design of CLDA algorithms.

4.4.1 Proposed convergence measures

Each time a CLDA algorithm's update rules are applied, some of the decoder's parameters are changed. If we consider the evolution of one of these parameters as forming a sequence, we can then investigate whether this sequence will converge near the corresponding optimal parameter value. Notationally, we use a superscript $^{(i)}$ to denote the i^{th} term in the sequence. For example, $\theta^{(0)}$ denotes the initial (seed) value of a parameter θ, $\theta^{(1)}$ denotes its value after the first CLDA update, and so on.

Let θ^* denote the optimal value of some parameter θ. To quantify convergence, we seek some measure of the deviation of $\theta^{(i)}$ from θ^*. One candidate measure is the difference between the expected value of $\theta^{(i)}$ and the optimal value θ^*:

$$\mathbb{E}\left[\theta^{(i)}\right] - \theta^*$$

Intuitively, one might expect that if a CLDA algorithm is effective at improving BMI performance, then for each decoder parameter θ, this difference should tend towards 0 as more parameter updates are performed:

$$\lim_{i\to\infty} \mathbb{E}\left[\theta^{(i)}\right] - \theta^* \;=\; 0.$$

Despite its simplicity, however, a critical shortcoming of this condition is that it is only a first-order convergence measure. In other words, it does not provide any measure of the "variance" of $\theta^{(i)}$ around θ^*, which could be very large even if the first-order condition holds. To measure this variance, we consider the following second-order measure, the (normalized) **Mean-Square Error (MSE)**[3] of a parameter θ at update iteration i:

$$\mathrm{MSE}^{(i)}(\theta) \;\triangleq\; \frac{\mathbb{E}\left[\left\|\theta^{(i)} - \theta^*\right\|_F^2\right]}{\left\|\theta^*\right\|_F^2}. \tag{4.10}$$

Note that in the above definition, the added normalization ensures that $\mathrm{MSE}^{(i)}(\theta)$ is not affected by the general scaling of the parameter θ.

While a highly performing decoder will necessarily have low MSEs, all decoders with the same MSEs may not necessarily result in the same performance. Because of this problem, we propose using an additional, more probabilistic convergence measure, **KL divergence (KLD)**, to support our analysis. Our intuition for using KL divergence is as follows. Let x_1^N and y_1^N be shorthand notation for observed intended kinematics $\{x_t\}_{t=1}^N$ and neural activity $\{y_t\}_{t=1}^N$, respectively. In addition, let $\theta^{(i)}$ and θ^*represent the set of decoder parameters at update iteration i and the optimal set of parameters, respectively (and let $p_{\theta^{(i)}}$ and p_{θ^*} be the distributions on $\left\{x_1^N, y_1^N\right\}$ induced by these parameter sets). Then the KL divergence between θ^* and $\theta^{(i)}$ is:

$$\begin{aligned} KLD\left(\theta^{(i)}\right) &= D_{KL}\left(p_{\theta^*} \| p_{\theta^{(i)}}\right) \\ &= \mathbb{E}\left[\log \frac{p\left(x_1^N, y_1^N | \theta^*\right)}{p\left(x_1^N, y_1^N | \theta^{(i)}\right)}\right]. \\ &= \mathbb{E}\left[\log p\left(x_1^N, y_1^N | \theta^*\right)\right] - \mathbb{E}\left[\log p\left(x_1^N, y_1^N | \theta^{(i)}\right)\right] \end{aligned}$$

where we have defined the shorthand notation $KLD\left(\theta^{(i)}\right)$. In other words, it is the difference between the expected log likelihood under the distributions induced by θ^* and $\theta^{(i)}$, respectively. Our intuition for using KL divergence as a convergence measure is as follows: if $KLD(\theta_1) = KLD(\theta_2)$ for two sets of parameters θ_1 and θ_2, then the averaged probability of observing $\left\{x_1^N, y_1^N\right\}$ is the same under both θ_1 and θ_2. In other words, the KL divergence has a probabilistic interpretation of equality that is not present with MSE, our first convergence measure.

[3]We have invoked the expectation operator $\mathbb{E}[\cdot]$, because $\theta^{(i)}$ is random variable *before* conducting a closed-loop CLDA experiment. However, *after* an experiment has been conducted, MSE is no longer a random quantity, but rather a sequence (indexed by i) that can be calculated from the corresponding sequence $\left\{\theta^{(i)}, i \geq 0\right\}$ (see Appendix for proposed methods of choosing θ^*). Therefore, it is important to note that experimental traces of MSEs are actually just SEs ("squared errors").

4.4.2 Case study: SmoothBatch algorithm

The convergence measures that we have proposed are general enough to apply to a large class of decoders and CLDA algorithms. However, to concretely demonstrate the application of these convergence measures to a real CLDA algorithm, we choose the SmoothBatch algorithm as a specific example in the sections that follow. SmoothBatch updates the observation model parameters of a Kalman filter decoder by periodically performing a weighted average of current parameters with maximum likelihood estimates from a new batch of data (see Section 4.2.3.3). The SmoothBatch algorithm has multiple CLDA parameters — the weighting parameter ρ, the batch period T_{b}, and the half-life h. By using SmoothBatch's parameter update rules and applying our convergence measures for $\theta \in \{C, Q\}$, we can predict the trade-offs inherent in the choices of these parameters and gain valuable insight into SmoothBatch's convergence properties.

In order to facilitate our mathematical convergence analysis, we must first make certain model assumptions. To make our calculations tractable, we will assume that the KF models hold true — i.e., that there exist underlying matrices $\{A, W, C, Q\}$ for which (2.1) and (2.2) hold true. Furthermore, we will assume that a KF decoder with its parameters set to these matrices would allow the user to maximize BMI performance. Accordingly, since these matrices then effectively represent the optimal parameter setting, we will denote them as $\{A^*, W^*, C^*, Q^*\}$. Finally, we will assume that these true, underlying matrices are unchanging over time.

With these model assumptions, we can apply our convergence measures to analyze the SmoothBatch algorithm. In the following sections, we focus our attention directly on the results of our calculations, and we refer the reader to the Appendix for the full derivation of these results. Using the Frobenius matrix norm and evaluating (4.10) for $\theta \in \{C, Q\}$ by substituting in SmoothBatch's update rules, we find that SmoothBatch's MSEs for C and Q are of the form:

$$\mathrm{MSE}^{(i)}(C) = \rho^{2i} \cdot \frac{\left\|C^{(0)} - C^*\right\|_F^2}{\left\|C^*\right\|_F^2} + \left(1 - \rho^{2i}\right) \frac{1-\rho}{1+\rho} \cdot \frac{f_C(T_{\mathrm{b}})}{\left\|C^*\right\|_F^2} \tag{4.11}$$

$$\mathrm{MSE}^{(i)}(Q) = \rho^{2i} \cdot \frac{\left\|Q^{(0)} - Q^*\right\|_F^2}{\left\|Q^*\right\|_F^2} + \left(1 - \rho^{2i}\right) \frac{1-\rho}{1+\rho} \cdot \frac{f_Q(T_{\mathrm{b}})}{\left\|Q^*\right\|_F^2} \tag{4.12}$$

where $f_C(T_{\mathrm{b}})$ and $f_Q(T_{\mathrm{b}})$ denote monotonically decreasing functions of the batch period T_{b}.

We can use the same model assumptions to apply our second convergence measure, KL divergence. For our KF decoder model, we find that the KL divergence between the probability distributions under $\theta^* = \{C, Q\}$ and $\theta^{(i)} = \left\{C^{(i)}, Q^{(i)}\right\}$ to be

$$KLD\left(\theta^{(i)}\right) = D_{KL}\left(p_{\theta^*} \| p_{\theta^{(i)}}\right)$$

$$\frac{1}{2}\log\left(\frac{\det Q^{(i)}}{\det Q^*}\right) + \frac{1}{2}\left(\mathrm{tr}\left[{Q^{(i)}}^{-1} Q^*\right] - m\right) \tag{4.13}$$

$$+\frac{1}{2}\mathrm{tr}\left[\left(C^* - C^{(i)}\right)^T {Q^{(i)}}^{-1}\left(C^* - C^{(i)}\right) M_x\right]. \tag{4.14}$$

Due to the matrix inverses in (4.14) and the particular form of SmoothBatch's updates, the KL divergence turns out not to be an ideal measure for predicting SmoothBatch's convergence properties. As a result, we will instead utilize KL divergence as a secondary convergence measure to both analyze SmoothBatch's actual experimental performance and validate predictions derived from our primary measure, MSE. It is important to note, however, that for other CLDA algorithms with different parameter update rules, KL divergence may actually be a more tractable measure than MSE, and therefore more useful for predicting convergence properties.

In the following sections, we further interpret (4.11) and (4.12) to analyze SmoothBatch's convergence properties and their dependence on the batch period T_b, the weighting parameter ρ, the half-life h, and the decoder's initial seeding.

4.4.3 Time evolution of SmoothBatch's MSE

Since a CLDA algorithm's goal is to make decoder parameters converge close to the optimal parameter setting, it is critical to determine how close these parameters can get. Can we expect an algorithm's MSEs to decrease over time to zero, or will they instead settle at non-zero values? If the latter holds true, then how to do choices of the algorithm's CLDA parameters affect these values? Our convergence analysis allows us to answer these questions and predict how SmoothBatch's MSEs evolve over time. Rearranging terms in (4.11), we can express SmoothBatch's MSE for C as:

$$\mathrm{MSE}^{(i)}(C) \;=\; \underbrace{\rho^{2i}\left(\frac{\left\|C^{(0)}-C^*\right\|_F^2}{\left\|C^*\right\|_F^2} - \frac{1-\rho}{1+\rho}\frac{f_C(T_\mathrm{b})}{\left\|C^*\right\|_F^2}\right)}_{\triangleq \mathrm{MSE}_\mathrm{tr}(C)} + \underbrace{\frac{1-\rho}{1+\rho}\frac{f_C(T_\mathrm{b})}{\left\|C^*\right\|_F^2}}_{\triangleq \mathrm{MSE}_\mathrm{ss}(C)}.$$

Our analysis predicts that SmoothBatch's MSE for C can be decomposed into the sum of a transient term, $\mathrm{MSE}_\mathrm{tr}(C)$, that decays as ρ^{2i} and a steady-state value, $\mathrm{MSE}_\mathrm{ss}(C)$, that remains unchanged over time (note that the same analysis applies to SmoothBatch's MSE for Q). If the SmoothBatch algorithm is applied for long enough, the transient terms will decay significantly, and the steady-state terms will dominate the MSEs. Therefore, our analysis predicts that SmoothBatch's MSEs for C and Q will settle to non-zero steady-state values:

$$\mathrm{MSE}_\mathrm{ss}(C) = \frac{1-\rho}{1+\rho}\frac{f_C(T_\mathrm{b})}{\left\|C^*\right\|_F^2} \qquad \text{and} \qquad \mathrm{MSE}_\mathrm{ss}(Q) = \frac{1-\rho}{1+\rho}\frac{f_Q(T_\mathrm{b})}{\left\|Q^*\right\|_F^2}. \tag{4.15}$$

From (4.15), we can analyze how SmoothBatch's steady-state MSEs are affected by changes in its CLDA parameters. For instance, since $f_C(T_\mathrm{b})$ and $f_Q(T_\mathrm{b})$ are both monotonically decreasing functions of the batch period T_b, increasing this parameter (for a fixed ρ) results in a decrease of the steady-state MSEs. Similarly, increasing the weighting parameter ρ (for a fixed T_b), or increasing the half-life h (for a fixed ρ or T_b) has the same effect. Using these relationships, we have a principled way to adjust SmoothBatch's CLDA parameters in order to achieve lower steady-state MSE values. However, as we will see, adjusting these same parameters also affects another quantity — the transient decay rate.

4.4.4 Rate of convergence

Although it is desirable for a CLDA algorithm to have low steady-state MSE values, it is equally important for it to achieve fast rates of convergences. For instance, even if one could design a CLDA algorithm with zero steady-state MSEs, the algorithm would be useless if its convergence rate was too slow for a real experiment. Fortunately, the transient terms of SmoothBatch's MSEs decay as ρ^{2i}, predicting that these MSEs exhibit *exponential* convergence to their steady-state values. In order to make this exponential convergence more clear, we can write ρ^{2i} in terms of the exponential function, thereby directly casting the transient's decrease over time as an exponential decay:

$$\mathrm{MSE}^{(i)} \;=\; \exp\left(-2i\ln\frac{1}{\rho}\right)\cdot \mathrm{MSE}_{\mathrm{tr}} + \mathrm{MSE}_{\mathrm{ss}}.$$

Note that this equation holds for both C and Q. Moreover, by introducing the batch period T_{b} into both the numerator and denominator, and using the fact that the product $T_{\mathrm{b}}i$ corresponds to the time of the i^{th} CLDA update, we can roughly express MSE directly as a function of the time t:

$$\mathrm{MSE}^{(t)} \;\approx\; \exp\left(-\frac{2\ln\frac{1}{\rho}}{T_{\mathrm{b}}}t\right)\cdot \mathrm{MSE}_{\mathrm{tr}} + \mathrm{MSE}_{\mathrm{ss}}.$$

From this expression, we observe that by decreasing either ρ or T_{b} (while keeping the other parameter fixed), one can increase the rate of convergence, and therefore allow SmoothBatch to reach its steady-state MSEs faster. Finally, using the definition of the half-life h in (4.9), we find that we can also express the rate of convergence as:

$$\mathrm{MSE}^{(t)} \;\approx\; \exp\left(-\frac{2\ln 2}{h}t\right)\cdot \mathrm{MSE}_{\mathrm{tr}} + \mathrm{MSE}_{\mathrm{ss}}.$$

This equation allows us to express the convergence rate solely in terms of h, and predicts that another way to increase the convergence rate is to simply decrease the half-life. In addition, from the properties of exponential decay, this result further implies that the time it takes for the transient MSE to decrease to a certain fraction (e.g., 0.01) of its original value is also proportional to h. Thus, our analysis predicts that h is directly tied to the convergence rate of SmoothBatch's MSEs, thereby confirming the notion of the half-life as an intuitive CLDA parameter as stated in Section 4.3.4.

4.4.5 CLDA parameter trade-offs

The predicted effects of changes to SmoothBatch's CLDA parameters on both the MSE decay rate and steady-state MSEs are summarized in Table 4.2. Note that for certain

CLDA parameter changes, the predicted effects are indeterminate or result in no change. Overall, our analysis predicts that the SmoothBatch algorithm exhibits a fundamental tradeoff between achieving a faster rate of convergence and lower steady-state MSEs. In general, increases in SmoothBatch's CLDA parameters lower its steady-state MSEs, but they also lead to a slower convergence rate, thereby increasing the total adaptation time required to reach steady-state. The opposite effect occurs when SmoothBatch's CLDA parameters are decreased.

Table 4.2: Summary of the predicted effects of changes to SmoothBatch's CLDA parameters on the MSE convergence rate and steady-state MSEs.

Increasing this	while holding	results in the following effect on the:	
parameter	this one fixed	MSE convergence rate	steady-state MSEs
ρ	T_b	decrease	decrease
	h	no change	indeterminate
T_b	ρ	decrease	decrease
	h	no change	indeterminate
h	ρ or T_b	decrease	decrease

4.4.6 Effect of decoder's seeding

Another important consideration is how decoder initialization affects different aspects of convergence. Since $C^{(0)}$ and $Q^{(0)}$ represent the initial values of the KF observation model, the decoder's seeding clearly determines the initial MSEs before any closed-loop decoder adaptation has been performed:

$$\mathrm{MSE}^{(0)}(C) = \frac{\left\|C^{(0)} - C^*\right\|_F^2}{\left\|C^*\right\|_F^2} \qquad \text{and} \qquad \mathrm{MSE}^{(0)}(Q) = \frac{\left\|Q^{(0)} - Q^*\right\|_F^2}{\left\|Q^*\right\|_F^2}.$$

However, $C^{(0)}$ and $Q^{(0)}$ do not appear in any of our expressions for either the rate of convergence or the steady-state MSEs. Therefore our analysis predicts that the decoder's seeding method does not affect either of these two quantities[4]. Since the steady-state MSE is a measure of the deviation of parameters from the optimal parameter setting — which in turn relates directly to performance — our convergence analysis predicts that the decoder's initial seeding should not affect the level of performance achievable using the SmoothBatch algorithm.

[4]Note that the seeding method can still affect the total time to achieve convergence — for example, the time required for the transient MSEs to decay below a certain level — but our analysis predicts that it will not affect the *rate* at which the transient MSEs decay.

4.4.7 Improving the SmoothBatch algorithm

Our analysis of the SmoothBatch algorithm has predicted a key trade-off between the convergence rate and the steady-state MSEs. However, with our new understanding of how SmoothBatch's CLDA parameters affect its convergence properties, we may be able to re-engineer the algorithm to overcome this seemingly fundamental limitation, and further improve the algorithm's performance. Specifically, we would like to modify SmoothBatch to achieve lower steady-state MSEs without significantly sacrificing the convergence rate. We propose allowing CLDA parameters to vary in time. Since the seed decoder's parameters may initially be far from their optimal setting, one could perform aggressive adaptation early to make large parameter updates that bring the parameters into the right "ballpark". However, as decoder parameters near their optimal values, more fine-tuned adaptation could be used to avoid potentially disruptive large parameter updates.

We propose using a time-varying version of SmoothBatch's ρ parameter, where $\rho^{(i)} \to 1$ as i increases so that parameter updates become smoother as time progresses. Letting $\rho \triangleq \rho^{(1)}$ and choosing a decay factor $\alpha < 1$, our proposed method for increasing $\rho^{(i)}$ over time is to make $1 - \rho^{(i)}$ decay exponentially as i increases:

$$1 - \rho^{(i)} \triangleq \alpha^{i-1} (1 - \rho)$$

Rearranging terms, we propose the following systematic method for increasing $\rho^{(i)}$ over time:

$$\rho^{(i)} \triangleq 1 - \alpha^{i-1} (1 - \rho), \qquad \alpha < 1 \tag{4.16}$$

Figure 4.4 illustrates the predicted effects of implementing a time-varying weighting parameter for an example case where $T_\text{b} = 80$ s and $\rho^{(1)} = 0.63$. The decay factor α governs the rate at which the weighting parameter will increase over time (Figure 4.4A). As closed-loop decoder adaptation is performed, the transient MSEs are predicted to decay differently depending on α (4.4B, log-scale). For standard SmoothBatch ($\alpha = 1$), the transient MSEs decay exponentially to zero, but when $\rho^{(i)}$ is time-varying ($\alpha < 1$), they decay at slightly slower rates. If $\rho^{(i)}$ is increased too quickly (e.g., $\alpha \leq 0.8$), lower steady-state MSEs are achieved (4.4C), but the transient MSEs no longer decay to zero (i.e., they are no longer "transient"). However, when the weighting parameter is increased at an appropriate rate (e.g., $\alpha \approx 0.9$), a significant, order-of-magnitude reduction in the steady-state MSEs can be achieved with very little difference in how the transient MSEs decay. In other words, our analysis predicts that with appropriate increases in the weighting parameter over time, we can overcome SmoothBatch's seemingly fundamental tradeoff, and achieve lower steady-state MSEs without significantly sacrificing the convergence rate.

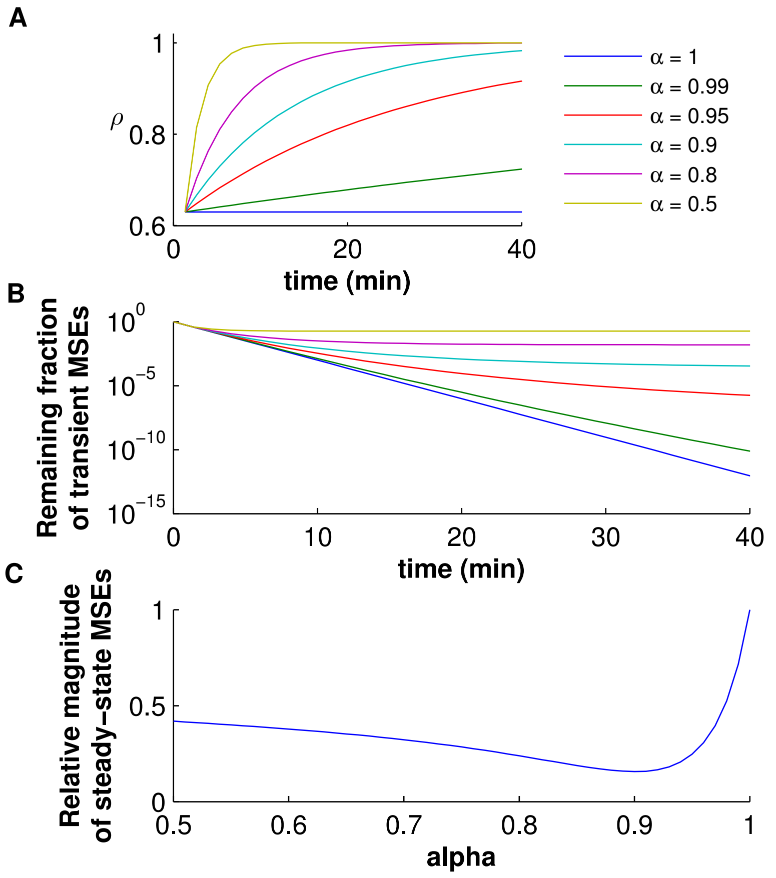


Figure 4.4: **Improving the SmoothBatch CLDA algorithm by using a time-varying ρ parameter.** (A) Trajectory of the weighting parameter ρ over time for different values of the decay factor α. (B) Predicted decay of the transient MSEs over time as closed-loop decoder adaptation is performed. (C) Predicted magnitude of the steady-state MSEs, relative to standard SmoothBatch, after 40 minutes of adaptation has been performed. For $\alpha \approx 0.9$, a significant reduction in the steady-state MSEs can be achieved with very little difference in how the transient MSEs decay. The batch period and the initial value of ρ were set to $T_\mathrm{b} = 80$ s and $\rho^{(1)} = 0.63$ for this example.

4.4.8 Experimental validation

To test our convergence predictions, we evaluated the performance of the SmoothBatch algorithm by conducting closed-loop experiments with two non-human primates performing a center-out task across 72 sessions. Figure 4.5 shows empirical traces of MSEs and the KLD, along with their exponential fits, for a sample session (see Appendix for details on the fitting procedure). As predicted by our convergence analysis, the MSEs indeed appear to decay exponentially and eventually settle to non-zero steady-state values. Across all sessions, exponential functions fit these traces more accurately than linear functions. The average squared-error for exponential fits to traces of both MSEs and KLD were significantly smaller than for linear fits (Wilcoxon paired test, $p < 10^{-6}$).

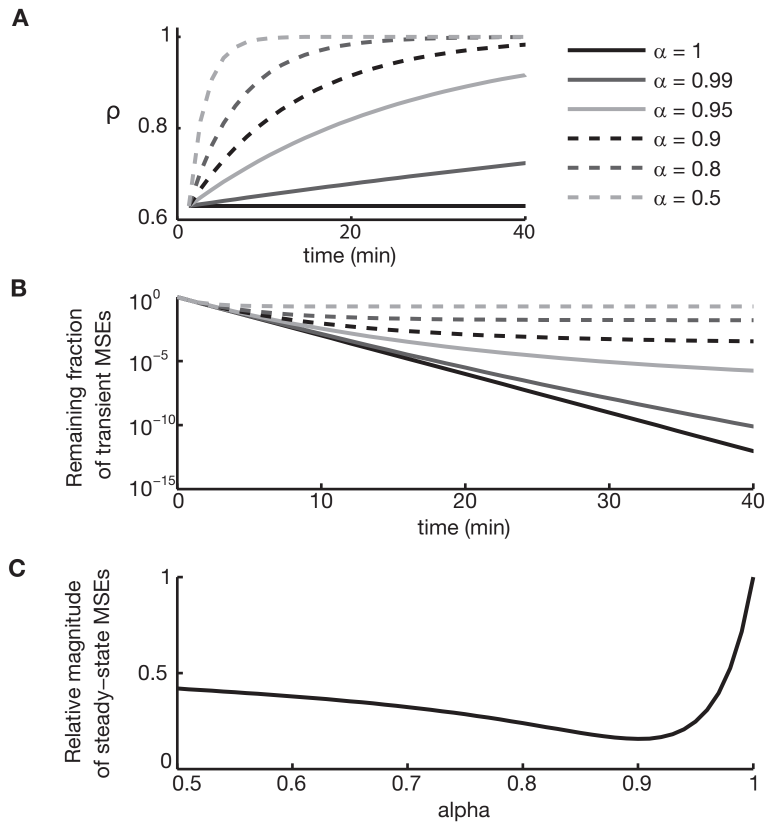


Figure 4.5: Experimental traces of MSEs (A and B) and KLD (C) for a session in which SmoothBatch was performed after VFB seeding.

Next, we analyzed the relationships between different settings of SmoothBatch's CLDA parameters and the corresponding empirically fit MSE decay rates and steady-state MSEs. Table 4.2 summarizes the predicted relationships for all of SmoothBatch's CLDA parameters. To simplify the results presented here, we focus only on the weighting parameter ρ. We restrict our analysis to sessions with 1) more than one data point for a given ρ and with 2) the same seeding condition (VFB), to isolate effects caused only ρ ($n = 29$). As predicted by convergence analysis, MSE decay rates are negatively correlated with ρ for both C ($r = -0.37$; $p < 0.05$) and Q ($r = -0.27$; $p = 0.15$) (Figure 4.6A–C). In addition, steady-state MSE values are negatively correlated with ρ for both C ($r = -0.50$; $p < 0.05$) and Q ($r = -0.33$; $p = 0.08$), further confirming the predictions derived from our convergence analysis (Figure 4.6D–F). Decay rates and steady-state values of KLDs showed similar correlations with ρ as MSEs.

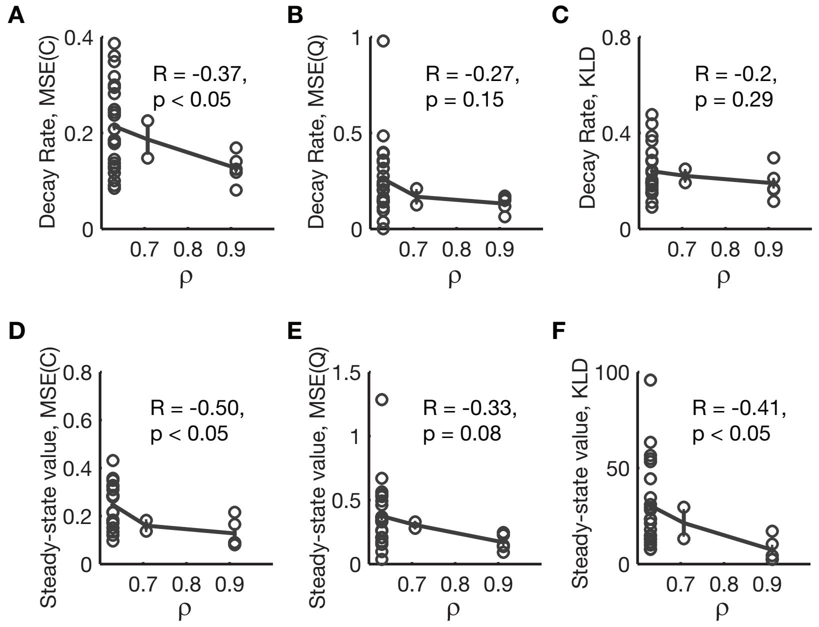


Figure 4.6: **Empirically-found relationships between SmoothBatch's ρ CLDA parameter and different aspects of convergence.** (A–C) Experimental decay rates of MSEs (A and B) and KLD (C), plotted vs. the weighting parameter ρ across all sessions with VFB seeding. (D–F) Experimental steady-state values of MSEs (D and E) and KLD (F), plotted vs. the weighting parameter ρ across all sessions with VFB seeding.

Finally, we tested the prediction that the decoder's initial seeding method should not affect either the MSE decay rate or the steady-state MSEs. Figure 4.7 shows the decay rates (top row) and steady-state values (bottom row) for MSE (C) (left), MSE (Q) (middle), and KLD (right), separated by seeding condition. We restrict our analysis to sessions with the same value of ρ to isolate effects of initialization (60 sessions total; *VFB*, $n = 22$; *Baseline*, $n = 17$; *Shuffled*, $n = 11$; *Ipsi*, $n = 8$; *Contra*, $n = 2$). Bars indicate the mean, error bars show standard error of the mean, and points show individual session data. A Kruskal-Wallis analysis of variance reveals some statistically significant differences in decay rates and steady-state values (differences indicated on the figure). However, no seeding conditions show consistent differences across any of the KF measures or convergence properties. Consistent with this result, previous work comparing behavioral improvements across seeding conditions revealed no significant differences across seeding conditions [28]. Specifically, all sessions reached similar final performance, and the time required to achieve maximum performance did not depend on the initialization.

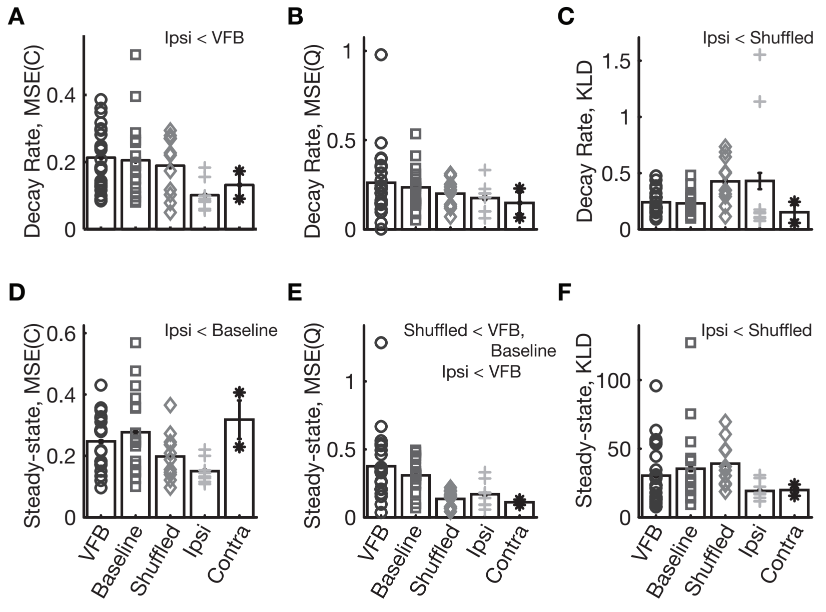


Figure 4.7: Experimental decay rates and steady-state values of MSEs (A and B) and KLD (C) across different decoder seedings.

4.5 Discussion

Closed-loop decoder adaptation is a powerful paradigm for rapidly improving online BMI performance. Here, we have established a general framework for the design and analysis of CLDA algorithms. We first identified a core set of important design considerations that frequently arise when designing CLDA algorithms. We explored the consequences of choosing different time-scales of adaptation, motivated the idea of selective parameter adaptation, illustrated the need for performing smooth parameter updates, and stressed the importance of having intuitive CLDA parameters. We then introduced mathematical convergence analysis as an effective design tool for predicting the convergence properties of a prototype CLDA algorithm before conducting closed-loop experiments. Using the SmoothBatch algorithm [28] as our case study, we demonstrated the ability to predict how CLDA parameters affect different aspects of convergence, and understand the resulting tradeoffs that are inherent in the choices of these parameters. We also characterized the effects of adjusting SmoothBatch's CLDA parameters, and found that the algorithm exhibits a fundamental tradeoff between the rate of convergence and the steady-state MSEs. By allowing for a time-varying weighting parameter, we found that we can achieve significantly lower MSEs with very little sacrificing of the rate of convergence. Although we demonstrated the utility of our convergence analysis using SmoothBatch as a particular example, our methods can be generally applied to a large class of other decoders and CLDA algorithms.

CLDA algorithms can operate on a variety of different decoders and can have different underlying goals, which may influence important aspects of their design. In clinical situations where motor deficits prevent patients from enacting the types of natural movements often used to seed the decoder, other methods of decoder initialization must be used (e.g., visual feedback seedings — see Section 4.2.2) that may result in low initial performance. The SmoothBatch algorithm has been demonstrated to achieve high-performance BMIs in these settings [28]. CLDA algorithms that aim to achieve different goals may differ from SmoothBatch in important aspects of their design. For instance, while SmoothBatch has been demonstrated to rapidly improve performance when operating on an intermediate time-scale of adaptation, a longer time-scale may be more appropriate for other algorithms with a different CLDA focus. As an example, Gilja et al. used batch maximum likelihood estimates of parameters to adapt a KF decoder. Using decoders seeded from contralateral arm movements and BMI performed during overt arm movements, subjects attained greater than 90% performance with the seed decoder in a center-out task. As a result, a single CLDA update formed from a 10-15 min batch of collected data was sufficient to achieve a significant improvement in reach kinematics [27]. However, when attempting to improve performance from unfavorable seedings, such a large batch period would be counterproductive — it would force the BMI user to persist for a long time with what might be a poorly performing decoder, and thus would likely reduce subject engagement in the task. Therefore, a single update of decoder parameters would likely not achieve significant performance improvements. Instead, high performance in this case could only be achieved either by performing multiple CLDA updates [28] or holding the decoder fixed long enough to facilitate the emergence of a stable motor memory [13].

Different types of CLDA training signals may also be more appropriate for other CLDA algorithms. Li et al. recently developed a CLDA algorithm for a KF decoder that aims to maintain long-term BMI control accuracy using an adaptive method of self-training updates [43]. Unlike SmoothBatch, Li et al.'s algorithm used overt arm movements to seed the decoder and was designed to sustain the accuracy of already high-performing decoders by adapting to neuronal plasticity and instability in neural recordings. As a result, Li et al.'s algorithm was able to directly use a Kalman smoothed version of decoder outputs as its training signal (it does not need to estimate the user's "intended kinematics"). In contrast, when SmoothBatch is used to improve performance from poor decoder seedings, the initial decoded cursor outputs are naturally a poor reflection of the user's intended kinematics, and therefore Kalman smoothing would not be an effective training signal[5].

Despite differences in the operation and goals of CLDA algorithms, some design elements are still likely to be shared in common by many CLDA algorithms. For example, despite its vastly different purpose, Li et al.'s algorithm maintains the property of selective decoder adaptation like SmoothBatch and does not adapt the KF's transition model parameters. Many algorithms are also designed in one way or another to have smooth decoder updates. Taylor et al. developed a CLDA algorithm for a PVA decoder in which the next set of decoder weights was determined from a corrected version of the current weights along with past weights that resulted in good performance [5]. Gage et al. estimated new KF parameters using a trial-by-trial sliding block of data, thus naturally allowing for smoothness across successive estimates [2]. Li et al.'s algorithm also achieved smooth updates, albeit in a characteristically different way, by endowing decoder parameters with probability distributions and updating them using Bayesian regression updates.

Closed-loop decoder adaptation should synergize well with previous results that have demonstrated the importance of neural plasticity in the BMI learning process [3, 5, 6, 11, 13, 47]. With respect to the learning and retention of neuroprosthetic skill, a study by Ganguly and Carmena demonstrated the importance of a stable neural map [13]. Some studies may have misinterpreted these results — for instance, contrary to Li et al.'s representation of these results, Ganguly and Carmena did not assert that a fixed decoder may be sufficient for long-term control accuracy [43]. Rather, they showed that a "stable circuit", consisting of a fixed decoder and stable neurons, can facilitate the development of a stable neural map of the decoder that enables performance improvements, can be readily recalled across days, and is robust to interference from a second learned map. However, even if CLDA is used to achieve initial rapid gains in performance, it is likely that subsequent fixing of decoder parameters could still allow the formation of such a stable map.

While our convergence predictions are consistent with our experimental results, we still find some variability in our results that is not accounted for by our analysis. For instance, calculating convergence measures from experimental data requires explicit assumptions that

[5]SmoothBatch instead uses Gilja et al.'s method for explicitly estimating intended cursor kinematics [27].

could introduce variability (see Appendix). As an example, the true underlying mapping between the observed neural firing rates y_t and the kinematic state x_t of the cursor is unlikely to be exactly linear with additive Gaussian noise. Existing work does suggest that incorporating nonlinearities into the KF could be beneficial, and that this decoding framework can be successfully used for CLDA [43]. Exploring system nonlinearities may, then, be a fruitful avenue for future research. However, we feel the design principles highlighted in this study of linear decoders will likely translate to nonlinear applications. Indeed, our findings regarding selective adaptation of KF matrices are consistent with the approach used by Li et al. using a non-linear KF.

Despite the aforementioned variability, all of our data show clear trends consistent with our convergence analysis predictions (e.g., the signs of all calculated correlations are consistent with our predictions). Indeed, the goal of our convergence analysis is not to predict the exact value of the MSEs of different decoder parameters. Instead, what we are more interested in is finding the general form of the MSEs, in order to understand both how they evolve over time and how they are affected by CLDA parameters (like SmoothBatch's ρ, T_b, and h). Furthermore, our methods should not only be able to evaluate a wide range of existing CLDA algorithms, but also provide insights into ways of improving these algorithms. Our convergence analysis techniques accomplish these goals, thus demonstrating that they can be an effective analytical tool for evaluating and informing CLDA algorithm design.

Chapter 5

BMI control using local field potentials

The work presented in this chapter was performed in collaboration with Kelvin So, Amy L. Orsborn, Michael C. Gastpar, and Jose M. Carmena, and was published in the Journal of Neural Engineering [48].

5.1 Introduction

Intracortical BMI studies have traditionally used single- and multi-unit activity as control signals, due to the fact that the discharge of motor cortical neurons correlates with different movement parameters, and that these cells can be volitionally modulated irrespective of physical movement using biofeedback [49]. On the other hand, certain features of local field potentials (LFPs), which are the extracellular potentials resulting from the synaptic activity in a neuronal population, have also been shown to exhibit similar tuning properties [50]. For instance, initiation of arm movement has been shown to reliably modulate power in the LFP beta band (~15–30 Hz) [51]. In addition, other studies have shown that muscle activity, eye movements, and reaching and grasping kinematics can all be decoded from LFP activity to varying degrees of accuracy [52–54]. This makes LFPs an attractive alternative to single- and multi-unit activity in BMIs because they are more robust to signal degradation over time [55] and contain more information than other less invasive field potentials such as electroencephalography (EEG) and electrocorticography (ECoG) signals.

Despite the promising properties of LFPs for BMI control, only a few studies to date have used them explicitly as an input signal in BMIs [56–58], with one study demonstrating 2-D continuous BMI control [58]. In that study, Flint et al. trained decoders offline biomimetically (using neural activity recorded during overt arm movements) and held them fixed thereafter. While proficient BMI control was achieved in this way, the biomimetic approach may constrain the subject to modulate neural activity during a closed-loop BMI task in a way similar to activity evoked during natural arm movements. For instance, a low frequency feature known as the local motor potential (LMP) has been found to be informative in decoding kinematic parameters of arm-related movements [59], and Flint et al. indeed reported that the LMP and the 0–4 Hz band were the most informative features for LFP-based BMI control [60]. However, little is known about the degree of flexibility of the brain to modulate LFP oscillations of higher frequencies, or whether a broader range of frequencies can be used for BMI control with LFPs.

Here, we apply closed-loop decoder adaptation (CLDA) [42] during LFP-based BMI control to adapt the decoder to subject-specific modulations of different LFP frequency bands. Using CLDA in combination with an assistive control paradigm, we trained two non-human primates to perform a center-out task using LFP activity in the 0–150 Hz range. We then analyzed the relative importance of the different frequency bands towards each monkey's cursor control. While both monkeys obtained proficient control of the cursor under identical task settings, the frequency bands most important to control were characteristically different between both monkeys. Next, we constrained the BMI control input to various frequency sub-ranges (0–40 Hz, 40–80 Hz, and 80–150 Hz) and found that our offline analyses accurately predicted the bands with which the monkeys achieved the best online performance. Interestingly, while each monkey performed better using certain frequency ranges, both monkeys were able to achieve BMI control well above chance level with all sub-ranges after decoder adaptation, suggesting that broad ranges of LFP frequencies can potentially be used for closed-loop BMI control. Finally, we found that the particular frequency bands most important for each subject's control influenced the number of channels they needed, due to differences in the correlations across channels at different frequencies. Overall, our results demonstrate proficient, continuous BMI control using LFPs and shed light on the range of frequencies that can potentially be used, with implications for channel and feature selection.

5.2 Materials & Methods

5.2.1 Electrophysiology and behavioral task

Two adult male rhesus macaques (*macaca mulatta*) were chronically implanted in the brain with bilateral microwire arrays of 128 teflon-coated tungsten electrodes (35 μm diameter, 500 μm wire spacing, 8×16 array configuration; Innovative Neurophysiology, Durham, NC), targeting the arm areas of primary motor cortex (M1) and dorsal premotor cortex (PMd). All procedures were conducted in compliance with the NIH Guide for Care and Use of Laboratory Animals and were approved by the University of California–Berkeley Institutional Animal Care and Use Committee.

LFP signals were sampled at 1 kHz using a 128-channel MAP recording system (Plexon Inc., Dallas, TX) and streamed to a dedicated computer running MATLAB (The Mathworks, Natick, MA) to implement feature extraction and closed-loop BMI control. Channels were referenced to ground and signal quality was visually inspected each day (channels with clear artifacts were removed). For each channel, we estimated the spectral power in 15 consecutive 10-Hz bands from 0–150 Hz using the multi-taper method [61]. Spectral estimation was

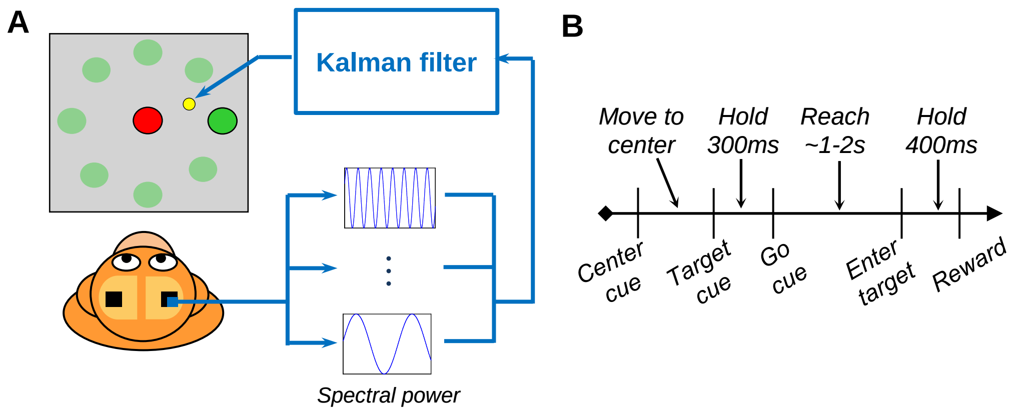


Figure 5.1: **Experimental setup.** (A) Illustration of the BMI task set-up. (B) Trial timeline for the BMI center-out task.

performed every 100 ms using a sliding window containing the most recent 200 ms of raw LFP activity. The estimates of log spectral power in different frequency bands, across multiple LFP channels, were used as neural features for closed-loop BMI control (Figure 5.1A).

The monkeys were head-restrained in a primate chair as they performed a self-paced 2-D center-out task. Monkeys were previously trained to perform this task by performing reaches using their right arm. During BMI control, the monkeys did not perform these arm reaches, as their arms were confined within the primate chair. Trials were initiated by moving the cursor under neural control to the center target and holding for 300 ms, after which the monkeys had to reach to one of eight peripheral targets uniformly spaced about a 13 cm diameter circle and hold for 400 ms to receive a liquid reward (target radii = 1.7 cm). The monkeys were then required to move the cursor back to the center target to initiate the next trial. If the monkeys failed to hold or reach the target within 10 s, the trial was restarted without reward (Figure 5.1B).

5.2.2 Decoding algorithm

A Kalman filter (KF) decoding algorithm ("decoder") was used to implement closed-loop BMI control. While there is no explicit delay between the kinematic state and neural activity in the KF model, each measurement y_t of extracted LFP features contained information from neural activity up to 200 ms in the past. C and Q were initially constructed by shuffling the rows and columns of a previous set of C and Q matrices that were trained with native arm reaches from the subject. These initial parameters were then updated during closed-loop BMI operation using the SmoothBatch CLDA algorithm (see 5.2.3). We typically adapted these parameters only during initial closed-loop control, and did not perform additional CLDA on the same decoder throughout the rest of the experiment.

During the first training session of each experiment, we utilized an assistive control paradigm [62] simultaneously with CLDA. After decoder initialization, the cursor was temporarily assisted towards the target. Specifically, the cursor trajectory was determined by:

$$\overrightarrow{v_{cursor}} = \alpha \cdot \overrightarrow{v_{assist}} + (1-\alpha) \cdot \overrightarrow{v_{user}}$$

where $\overrightarrow{v_{user}}$ is the decoded output from the Kalman filter, $\overrightarrow{v_{assist}}$ is a vector that points directly towards the current target, and $\overrightarrow{v_{cursor}}$ is a weighted average of the two that determines the final cursor output shown to the subject. The weighted average was set by the assist level $\alpha \in [0, 1]$, which was manually adjusted in real-time depending on the subject's performance (at $\alpha = 0$, the cursor was fully controlled by the subject). The assist speed was set to 0.8 cm/s to match the natural speed of the cursor when the subject performed the center-out task under manual control. All analyses were performed on data with the cursor under full volitional control (0% assist).

5.2.3 Closed-loop decoder adaptation

Closed-loop decoder adaptation (CLDA) is an emerging paradigm for improving or maintaining the online performance of brain-machine interfaces. By adapting the decoder's parameters during closed-loop BMI operation (i.e., while the subject is using the BMI), CLDA algorithms aim to match the decoder's output to the subject's particular pattern of neural activity [2, 5, 18, 27, 28, 30, 31, 37, 43, 46]. In our experiments, we used the SmoothBatch CLDA algorithm to adapt the KF decoder's parameters during initial closed-loop control. The SmoothBatch CLDA algorithm has been previously used in spike-based BMI experiments, where it was demonstrated to rapidly improve BMI performance independent of the decoder's initialization method [28]. Furthermore, SmoothBatch has also been shown to possess a variety of favorable algorithmic convergence properties [42].

To illustrate how the SmoothBatch CLDA algorithm operates, let $C^{(i-1)}$ and $Q^{(i-1)}$ denote the current observation model parameters of the KF decoder. SmoothBatch first collects a batch of neural activity and cursor kinematics as the subject operates the BMI cursor in closed-loop control using a decoder with these parameters. Next, the algorithm generates an estimate of the user's intended cursor movements. For example, in our experiments, we used Gilja et al.'s method for inferring intended cursor kinematics ("innovation 1" of the ReFIT-KF algorithm), which assumes that the subject always intends to move to directly towards the current target [27]. Other methods for estimating intended movements could also be used in conjunction with SmoothBatch, such as Shpigelman et al.'s supervised method [30] or Li et al.'s unsupervised method [43]. Using the estimate of intended cursor kinematics and the recorded neural activity, SmoothBatch then constructs batch maximum likelihood estimates, $\hat{C}$ and $\hat{Q}$, of the C and Q matrices using the following equations:

$$\hat{C} = YX^T \left(XX^T\right)^{-1} \tag{5.1}$$

$$\hat{Q} \;=\; \frac{1}{N}\left(Y-\hat{C}X\right)\left(Y-\hat{C}X\right)^{T} \tag{5.2}$$

where the Y and X matrices are formed by tiling N columns of recorded neural activity and intended kinematics, respectively. The size of the data batch is parametrized by the batch period $T_{\mathrm{b}} \triangleq N \cdot \mathrm{dt}$ (where dt is the time between decoder iterations). Finally, the algorithm updates the observation model parameters using a weighted average:

$$C^{(i)} \;=\; (1-\rho)\,\hat{C}^{(i)} + \rho C^{(i-1)} \tag{5.3}$$
$$Q^{(i)} \;=\; (1-\rho)\,\hat{Q}^{(i)} + \rho Q^{(i-1)} \tag{5.4}$$

where i indexes discrete batch periods and the weighting parameter $\rho \in [0,1]$ controls the influence of $\hat{C}$ and $\hat{Q}$ on the new parameter settings. Note that SmoothBatch does not update the KF transition model parameters A and W. We typically reparametrize ρ in terms of a half-life h (i.e., the time it takes for a previous maximum likelihood estimate's weight in the decoder to be reduced by a factor of 2):

$$\rho^{h/T_{\mathrm{b}}} = \frac{1}{2}.$$

where T_{b} is the size of the batch ("batch period") measured in units of time. In our experiments, batch periods and half-lives were typically chosen in the ranges $T_{\mathrm{b}} \in [1,6]$ mins and $h \in [0.5, 15]$ mins.

5.2.4 Performance evaluation

We quantified performance by measuring the target acquisition rate in trials/min and the percentage of initiated trials (those in which the subject was able to hold for 300 ms at the center) that ended in a success, both of which were computed using a 5 minute sliding window with 30 s steps. For comparison, we estimated the chance rate (measured to be 0.3 trials/min at a 0% assist level) by simulating 30 minutes of the task performed using a random walk trajectory (generated with the same transition model parameters as the Kalman filters used for BMI control). We also measured trajectory quality using reach time (RT), normalized path length (NPL), movement error (ME), and movement variability (MV); see [28] for details on each metric.

To perform approximate comparisons of BMI cursor control performance across studies (see Discussion), we calculated a summary statistic using the Fitts Law derived index of difficulty, which has been proposed as a standardized assessment method for neural prostheses [63]. The index of difficulty, measured in bits, is calculated based on the distance to target ("Distance") and target diameter ("Window") as:

$$\text{Index of difficulty} \;=\; \log_2\left(\frac{\text{Distance} + \text{Window}}{\text{Window}}\right)$$

From this value and the mean reach time, we calculate throughput in Fitts bits/sec as:

$$\text{Throughput} = \frac{\text{Index of difficulty}}{\text{Mean reach time}}$$

We refer the reader to [27] for further details on throughput calculation.

5.3 Results

During an initial set of experiments, we randomly chose 20 channels from the right M1/PMd implants and extracted the spectral power from fifteen 10-Hz bands, spanning 0–150 Hz, as input features to the decoder. Both animals attained full control of the 2-D cursor using CLDA with the assistive control paradigm. Assistive control started at 100% ($\alpha = 1$) and was reduced to zero after day 1 and CLDA was stopped after day 2. Both animals performed the task under full volitional control for two additional days. The target acquisition rate without assist improved from day 2 to 4 for both monkeys (Pearson's correlation coefficient, Monkey S: $R = 0.34$, $p < 0.001$; Monkey J: $R = 0.30$, $p < 0.001$). The average target acquisition rates on the 4th day for Monkey S and Monkey J were 10.6 ± 2.4 trials/min and 5.8 ± 2.9 trials/min, respectively. The average success percentage was $78\% \pm 6\%$ for Monkey S and $72\% \pm 8\%$ for Monkey J. Figure 5.2A shows typical reach trajectories, rate of target acquisition, and chance level for both monkeys on the 4th day of the series.

Trial-averaged spectrograms for each of the 8 target directions, along with tuning curves for each frequency band, are shown in Figure 5.2B–C for the two monkeys. The modulation depth, defined as the difference between the highest and lowest points of the tuning curve, is shown for each frequency in Figure 5.2D. Monkey S showed highest modulation in the 0–20 Hz frequency range, with another peak in the 50–80 Hz range, but decreasing modulation at higher frequencies. In contrast, Monkey J showed highest modulation depth in both 0–20 Hz and 80–150 Hz range. In addition, the LFP modulations during BMI control were also considerably different than those during natural arm control. Figure 5.3 shows the trial-averaged spectrograms, tuning curves and modulation depth plots in prior training sessions where Monkey S performed the same center-out task using natural arm reaches (no brain control). The characteristic decrease in beta band (15–40 Hz) power during movement was clearly evident during natural arm movement, but did not occur during BMI control.

Although tuning curves are one representation of the relationship between cursor movement and neural activity, they do not fully capture the exact mechanism by which the cursor position is generated from the neural activity at each iteration, since this is specified by the BMI decoder. To provide further insight into the LFP activity modulated during control, we

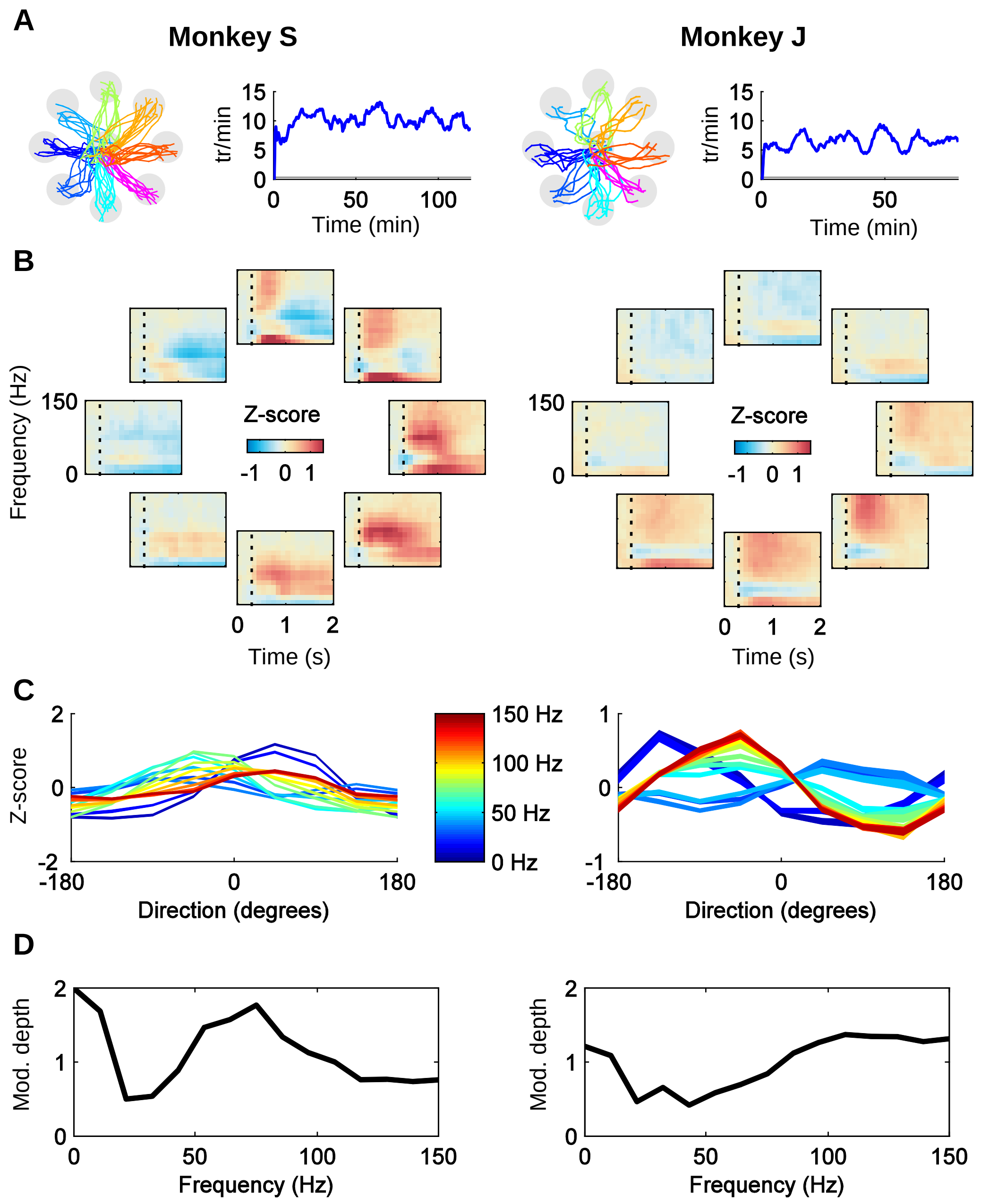


Figure 5.2: **Neural activity during closed-loop BMI control.** (A) *Left:* Typical reach trajectories to the eight center-out targets on the fourth day of the series for both monkeys. *Right:* Target acquisition rate on 4th day. Gray line indicates chance rate. (B) Trial-averaged spectrograms illustrating the z-scored log power on one channel during successful reaches for both subjects. (C) Tuning curves for each of the 15 spectral power features (on one particular channel) used for closed-loop control. Line thickness reflects standard error of the mean. (D) Modulation depth as a function of LFP frequency.

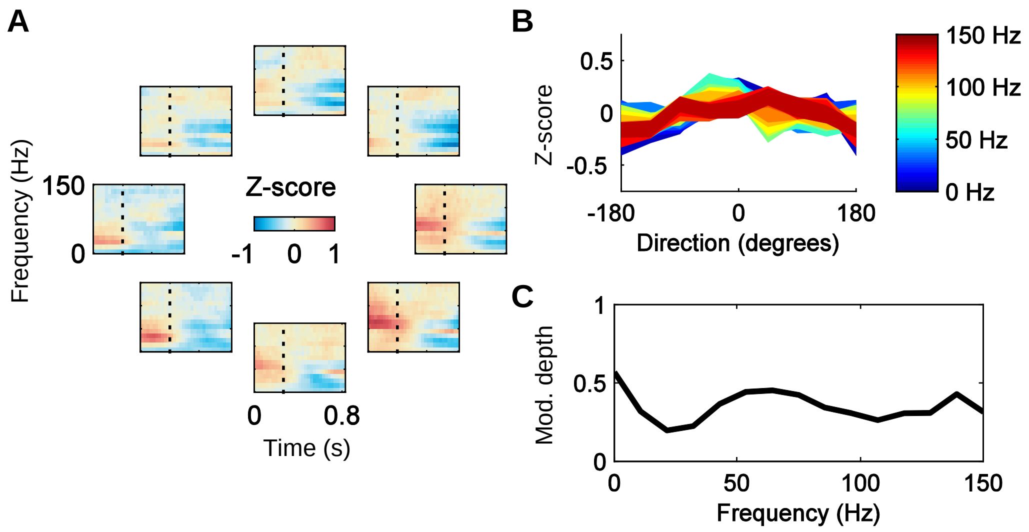


Figure 5.3: **Neural activity during natural arm movement.** (A) Trial-averaged spectrograms illustrating the z-scored log power on one channel during natural arm reaches in a center-out task for Monkey S. Dotted line indicates the go cue. (B) Tuning curves for each of the 15 spectral power features (on one particular channel). Line thickness reflects standard error of the mean. (C) Modulation depth as a function of LFP frequency.

examined the KF decoder's observation model parameters (C and Q), which capture the relationship between cursor kinematics and observed neural activity. Since our KF state vector models cursor position and velocity, the rows of C reflect the preferred position and velocity directions of different LFP features. For instance, Figure 5.4A illustrates the preferred velocity directions for each 10-Hz frequency band (averaged across channels) as assigned in C. In combination with the covariance structure modeled in Q, these preferred velocities ultimately determine how feature modulations of different features contribute to cursor movement at each iteration. For example, Figure 5.4B (top) shows the contributions to the cursor displacement from each LFP frequency band (averaged across channels) during an example trial. Each individual vector represents the net cursor displacement during the reach due to the modulations of a particular frequency band (together, the vectors add to produce the overall cursor displacement). The underlying modulations of neural activity (averaged over the course of the reach) that were generated to create this trajectory are shown in Figure 5.4B (bottom). Hence, to generate the trajectory shown, Monkey S increased the 0–20 Hz activity while reducing activity from 30–80 Hz.

In order to identify the LFP frequencies within the 0–150 Hz range that were most important for each monkey's overall BMI control, we calculated a movement contribution profile for each monkey by similarly decomposing the cursor movement across all successful trials. We considered three broad groups of frequencies — 0–40 (approximately the delta, theta, mu, and beta bands), 40–80 (low gamma), and 80–150 Hz (high gamma) — and determined the contribution of each frequency group, measured as a percentage of the total displacement. The resulting percentages directly measure which frequencies were responsible for moving the cursor on the screen (Figures 5.4C–D, pie charts). For Monkey S, the 0–40 Hz and 40–80 Hz bands together accounted for an average of 84% of the cursor movement per reach, while the higher frequencies (80–150 Hz) only accounted for 16% of the movement. Interestingly, Monkey J's movement contribution profile was markedly different than that of Monkey S. For Monkey J, the 80–150 Hz and 0–40 Hz bands were most responsible for cursor movement (46% and 34%, respectively).

As another way to measure the importance of different LFP frequencies for closed-loop control, we performed a leave-one-out offline sensitivity analysis by removing frequency groups from the Kalman filter decoder model and measuring the resulting increases in the steady-state Kalman error variances. These increases measure the impact on the decoder's confidence in its state estimates if the corresponding features were not used in closed-loop decoding (Figures 5.4C–D, bar graphs). For Monkey S, removing the 0–40 Hz and 40–80 Hz bands resulted in larger increases (34% and 17%) in the Kalman error variances than removing the 80–150 Hz frequencies (5%), suggesting that the 0–80 Hz frequencies were relatively more important. On the contrary, for Monkey J, removing the 80–150 Hz frequencies resulted in the largest increase (25%) in the error variance compared to removing the 0–40 Hz or 40–80 Hz bands (12% and 6% increase, respectively), suggesting that 80–150 Hz was more important than the other two frequency groups for Monkey J.

5.3.1 Testing different frequency bands

To test the predictions derived from our analyses, we conducted multiple experimental series in which both monkeys were restricted to only use features extracted from various sub-ranges (0–40 Hz, 40–80 Hz, and 80–150 Hz) of the full 0–150 Hz (from the same 20 channels) for closed-loop BMI control. Each series was composed of 3–4 days, with CLDA and assistive control performed on the first day starting from shuffled decoder parameters as before. On subsequent days, CLDA was used on rare occasions only if performance decreased drastically. Figure 5.5 depicts both monkeys' performance, measured using average success percentage, mean reach time and normalized path length, across these multiple frequency series.

Among the three frequency groups, Monkey S performed best in all metrics (success percentage, RT, NPL, MV, and ME) when using 40–80 Hz for closed-loop control. This result is consistent with his movement contribution profile (Figure 5.4C), which showed that the 40–80 Hz band accounted for an average of 41% of cursor movement during trials. However, the contribution profile showed that the 0–40 Hz band also accounted for a large percentage of movement (43%), and furthermore, our Kalman sensitivity analysis predicted 0–40 Hz to be the most important band for Monkey S. These analyses suggested that the combination of the two frequency bands (0–40 and 40–80 Hz) might allow Monkey S to achieve a higher level of performance. As such, we conducted an additional frequency series

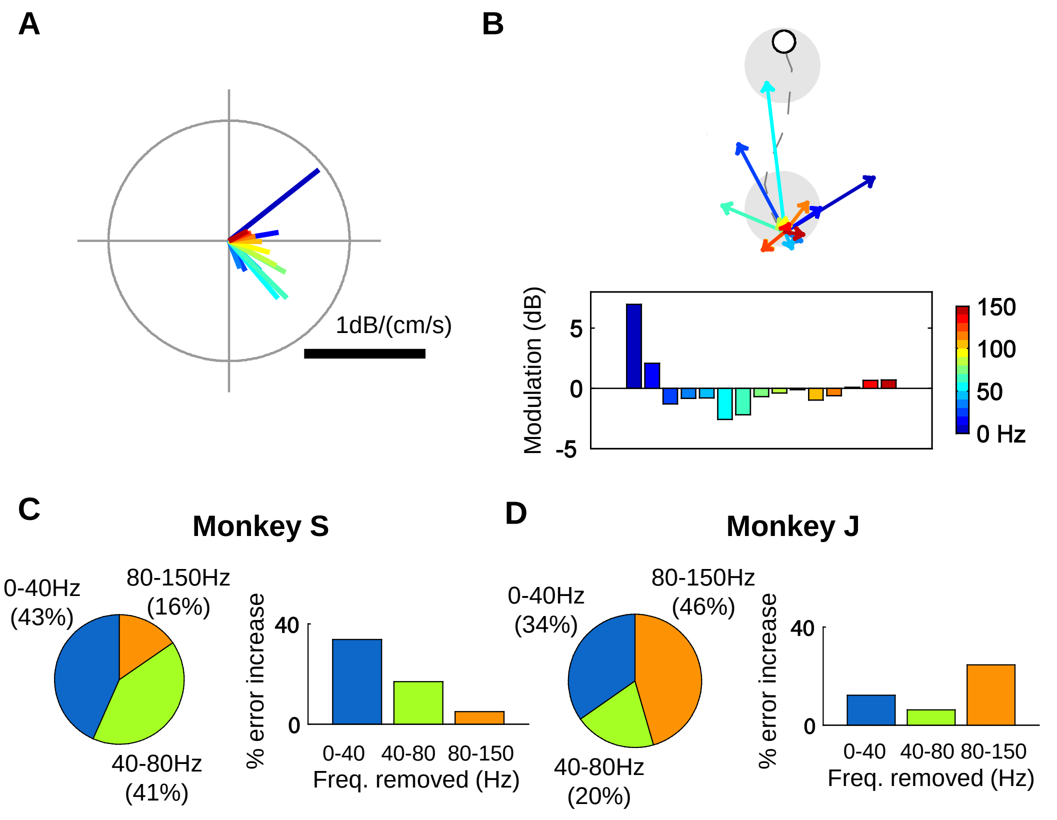


Figure 5.4: **Importance of different LFP features to closed-loop BMI control.** (A) Preferred velocity directions for each 10-Hz frequency band as assigned in the KF observation model matrix C for Monkey S. (B) *Top:* Cursor trajectory (dashed gray line) during an example reach to the top target. Each colored vector indicates the net cursor displacement during the reach resulting from a particular frequency band. The vectors from all frequency bands add to produce the overall cursor displacement. *Bottom:* The underlying modulations of neural activity generated during the sample reach. (C–D) Movement contribution profiles (pie charts, *left*) and sensitivity analyses (bar graphs, *right*) across three groups of frequencies — 0–40, 40–80, and 80–150 Hz, for both monkeys. Movement contribution profiles were calculated by decomposing the subjects' movements on successful trials into contributions from the three groups of frequencies, as in (B, *top*). Sensitivity analyses reflect increases in the Kalman error variances when certain groups of frequencies are removed from the KF decoder model.

in which Monkey S used 0–80 Hz for control. As expected, with respect to all metrics, Monkey S performed best across all frequency series when using 0–80 Hz. For instance, Monkey S's reach times were significantly lower when using 0–80 Hz than any other band ($p < 0.001$ for comparisons to each of the other four frequency series; Kruskal-Wallis ANOVA). Differences in mean normalized path length, movement error, and movement variability when using 0–80 Hz versus any other band were also all significant ($p < 0.001$ for ME, MV, and NPL; Kruskal-Wallis ANOVA).

Conversely, among all bands, Monkey J performed best using 80–150 Hz in all metrics ($p < 0.001$ for differences in RT, NPL, ME, and MV; Kruskal-Wallis ANOVA). Indeed, these findings confirm the results of Monkey J's movement contribution profile and Kalman sensitivity analysis, both of which predicted 80–150 Hz to be most important for his cursor control (Figure 5.4D). Moreover, just as those analyses had predicted 0–40 Hz to be second most important, Monkey J indeed performed second-best across all frequency series when using 0–40 Hz. Finally, Monkey J performed worst across all metrics when using 40–80 Hz, a result that is also consistent with predictions from the analysis.

Although we found that both monkeys performed better with some frequency sub-ranges than others, it is interesting to note that they were still able to achieve BMI control with all sub-ranges after initial decoder adaptation. Overall, these results suggest that there may be broad flexibility in the frequencies that could potentially be used for closed-loop LFP-based BMI control.

5.3.2 Testing different numbers of LFP channels

We also investigated the impact of reducing the number of channels on BMI control. Due to the close proximity of adjacent channels on the electrode array, the correlation between the raw LFPs from channels on the same array often exceeds 0.95. Hence, it is possible that multiple channels on the same array may not contribute additional information or degrees of freedom.

We varied the number of channels used for control in Monkey S from 50 channels to 1 channel (all from the right hemisphere, split between M1 and PMd) and tested each configuration for 3–4 days. We found that reach times decreased over time irrespective of the number of channels used (Figure 5.6A). We repeated the 50 channel configuration after the last series (1 channel series) and the reach times rapidly approached that of the previous series and kept decreasing. As mentioned above, one reason for the apparent low impact of the number of channels may be due to the high correlation between all LFP channels on a single electrode array. Comparing the power of each frequency band across 20 channels, we found high correlations between channels at frequencies below 80 Hz, but lower correlations at higher frequencies (Figure 5.6C). Since Monkey S relied primarily on modulating frequencies below 80 Hz for control, the redundancy across channels in the input

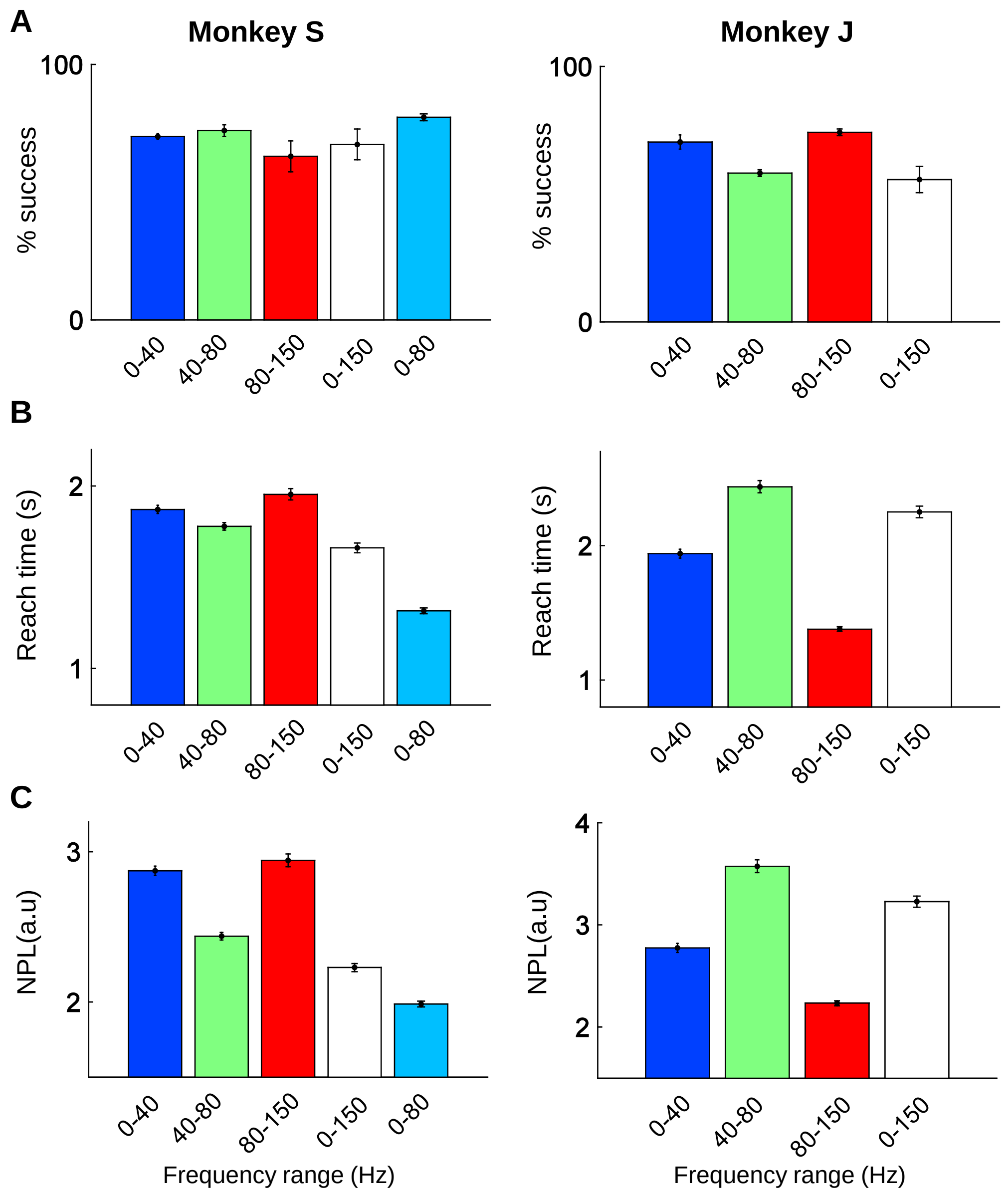


Figure 5.5: **Closed-loop BMI control with different sub-ranges of LFP frequencies.** Experimental performance, measured using (A) success percentage, (B) reach time (RT, seconds) and (C) normalized path length (NPL, arbitrary units), of both subjects across multiple experimental series in which various sub-ranges of the full 0–150 Hz range were used as neural features for closed-loop control. Bars indicate the mean across the entire series, and error bars indicate standard error of the mean.

was especially prominent. For Monkey S, our results suggested that the volitional modulation of different frequency bands, from even a single channel, was sufficient for 2-D cursor control.

We also tested BMI control in Monkey J with a 20 channel and 1 channel configuration (Figure 5.6B). In contrast to Monkey S, Monkey J's initial reach times with 1 channel input were dramatically higher than the 20 channel input series that was used earlier. While the reach times decreased with training, the high initial increase in reach times when switching from 20 channels to a single channel for Monkey J suggested that the removal of channels from the same hemisphere had a greater detrimental effect for Monkey J than for Monkey S. The disparity may result from the difference in frequency bands most important for cursor control between Monkey S and J, and the correlations across channels for those frequency bands. For Monkey J, the 80–150 Hz band accounted for most of the cursor movement, but was also less correlated among channels (Figure 5.6D). Hence, there was less redundancy across channels compared to Monkey S, resulting in a greater effect on performance when channels were removed. Overall, we found that the effect of the number of LFP channels used on BMI control can vary depending on the strategy of frequency modulations used by a particular subject.

5.4 Discussion

By using an adaptive approach (CLDA) to fit decoder parameters, we match the decoder output to the modulations of LFP evoked by the monkeys as they attempted to move the cursor towards the target. This paradigm allows the cursor to be controlled by the specific LFP modulations evoked by the subject during closed-loop BMI control. As a result, while both monkeys were trained on identical task settings, our movement contribution and sensitivity analyses revealed differences between monkeys in the contribution and relative importance of each LFP frequency band. We tested the analysis results empirically by requiring the monkeys to perform the same task using only sub-ranges of the entire 0–150 Hz band. For Monkey S, the 0–40 Hz (delta, theta, mu, and beta bands) and 40–80 Hz (low gamma), contributed significantly to the cursor movement (43% and 41%, respectively) based on the movement contribution profile. Subsequently, Monkey S performed best across all performance metrics when using frequencies 0–80 Hz and worst when these frequencies were removed. In contrast, for Monkey J, the 80–150 Hz (high gamma) range contributed the most out of the three groups to cursor movement in our analysis (46%), and indeed Monkey J performed significantly better across all metrics when using 80–150 Hz for closed-loop control. However, it is interesting to note that both monkeys were able to achieve above chance level performance using *all* of the frequency sub-ranges, even when they didn't include the most important frequency bands for each monkey.

A prior LFP-based BMI study by Flint et al. also extracted LFP spectral power as neural features, but sub-divided frequencies in a different way: 0–4, 7-20, 70–115, 130–200,

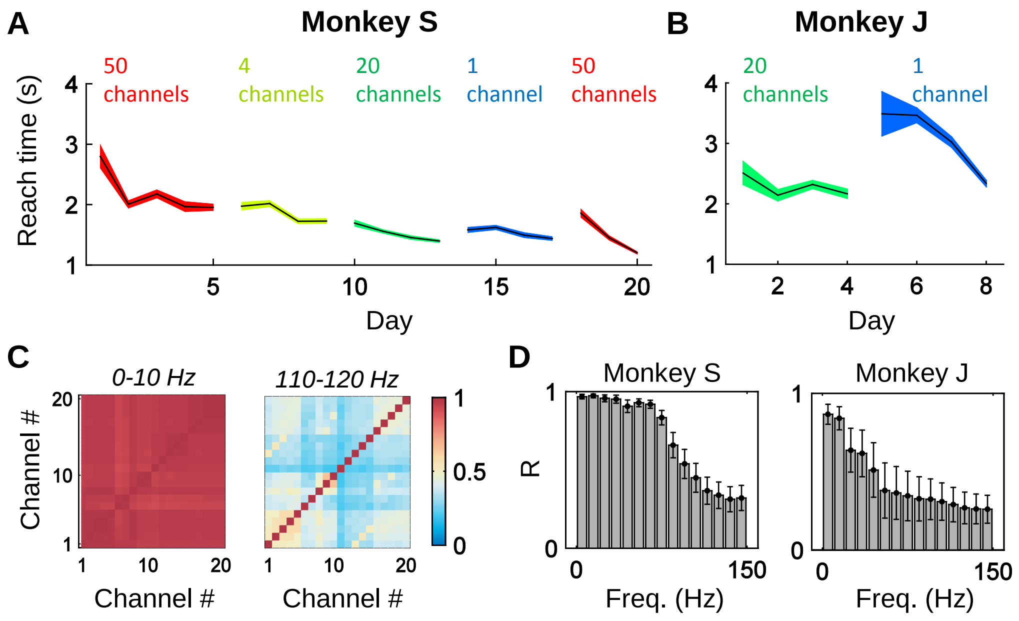


Figure 5.6: **Closed-loop BMI control with different numbers of LFP channels.** Experimental performance of (A) Monkey S and (B) Monkey J, measured using reach time, across multiple experimental series in which varying numbers of LFP channels were used in the decoder for closed-loop control. (C) Correlation matrices for Monkey S depicting the correlations across channels for the 0–10 Hz and 110–120 Hz frequency features (i.e., log power in different frequency bands). (D) Average correlation between features values on different channels for both subjects.

and 200–300 Hz [58]. In addition, Flint et al. utilized the local motor potential (LMP) — a very low-pass filtered version of the time-domain LFP signal — as a feature for closed-loop control. In a related initial study [60], Flint et al. reported that the LMP and the power in the 0–4 Hz band appeared to be among the most informative features for control for both subjects (together, they accounted for 80% and 43% of the features used in their two monkeys). A key difference compared to the current study was that Flint et al. used a fixed decoder trained from neural activity collected as the monkey performed natural reaches. Since the decoder parameters did not adapt, the subjects may have been constrained to modulate neural activity in ways similar to the activity used to fit the decoder; namely, those generated during natural reaches. Given that the LMP has been shown to be informative in decoding movement trajectories and reach direction [59], it is not surprising that the LMP was found to be the one of the most informative features for control when using a fixed decoder trained from natural reaches. In the current study, the decoder was initialized using a random shuffling method and fit with an adaptive procedure, using neural activity evoked during closed-loop BMI control. Hence, subjects were not constrained to modulate neural activity in a predefined way. The range of informative frequencies in our results indicate that a broader band of LFP frequencies can potentially be effective for closed-loop BMI control. This result is consistent with recent findings that LFP power in the 30–50 Hz range and ECoG power in the 75–105 Hz range can be volitionally modulated during closed-loop BMI in a operant conditioning paradigm [57, 64, 65].

Are there frequencies that are inherently more readily modulated than others in a closed-loop BMI setting? Hwang et al. [56] reported that LFP power in the 0–10 Hz and 20–40 Hz bands in the parietal reach region could be volitionally modulated to initiate trials. Engelhard et al. [57] found that the 30–50 Hz band could be robustly modulated when a constraint to use this range is directly imposed. When such constraints are relaxed, our results indicate that the range of frequencies that subjects modulate can vary from subject to subject. One potential reason for the variation may be due to the recording quality of high frequencies from the electrode array. From the time of initial array implantation to the time of the present study, the LFP power in frequencies above 80 Hz decreased by approximately 10 dB for Monkey S (~3 years post implant) but only 2 dB for Monkey J (~1.5 years post implant). Hence, the low contribution of the 80–150 Hz band to BMI control for Monkey S may be a result of the low signal-to-noise ratio in that frequency band. Nonetheless, Monkey S was able to obtain control well above the chance level using both these and other frequency bands (e.g., 0–80 Hz), indicating that broad ranges of LFP frequencies can potentially be used for a closed-loop BMI.

While the randomized decoder initialization may have some effect on the subjects' control strategy, we believe this effect to be negligible. The trained decoder that is achieved using the SmoothBatch CLDA algorithm ultimately depends on the patterns of spectral power modulations evoked by the subject, and the co-occurrence of those patterns with different directions of cursor movement. Therefore, if the subject uses roughly the same control strategy each time a new decoder is trained using CLDA, then the final trained decoder parameters will end up very similar, regardless of the initialization. Furthermore, due to the high level of assistive control during the initial CLDA, the cursor initially moves accurately regardless of the decoder's initialization, thereby significantly reducing any effect of the decoder's initialization on the subject's visual feedback and resulting control strategy.

An important implication that follows from the monkeys' choice of modulated frequencies is the effect on BMI control when the number of channels used are reduced, which relates to broader questions about the range of LFP signals. Past studies have suggested that LFPs can reflect neuronal processes 0.5–2 mm away [66]. Modulations of low frequencies (8–30 Hz) have been observed across wide areas of the motor cortex [67], while ECoG studies have reported spatially specific activity at higher frequencies ($>$40 Hz) [59]. The correlation between channels for the different frequency bands in the current study are consistent with these previous studies —namely, low frequencies were more correlated than high frequencies across the array. Moreover, the monkeys' task performance when using varying numbers of channels for control also confirms these results, as Monkey J performed worse than Monkey S when using fewer channels due to his greater dependence on higher LFP frequencies.

Comparison of BMI performance

Development of intracortical brain-machine interfaces to date have largely focused on spike activity as the only source of control input. The present study adds to a growing body of evidence that extracellular field potentials, measured through either intracortical arrays (i.e. LFP) or electrodes on the cortical surface (i.e. ECoG), are viable alternatives for BMIs. To compare the level of BMI performance reported here with that of other studies in the literature, we calculated the throughput in bits per second for various studies (see Methods). Throughput is a summary statistic that uses the mean reach time and Fitts Law derived index of difficulty [68, 69], which has been proposed as a standardized assessment method for neural prostheses [63]. Table 5.1 compares the BMI performance achieved in this study with a sampling of other notable closed-loop BMI cursor control studies using spikes, LFPs, ECoG, and EEG. For consistency, all studies listed in the table used a 4- or 8-target center-out task, except for Flint et al., 2013 [58] (we chose to include this study as well because it represents essentially the first demonstration of closed-loop continuous 2-D control with LFPs). To calculate throughput for Monkey S and J in the present study, we used the mean reach times from their 0–80 Hz and 80–150 Hz experimental series, respectively. Due to a wide range of varying task parameters in the literature (e.g., center-out vs. point-to-point reaching), it should be noted that the throughput metric is imperfect and does not allow for an exact comparison of cursor control performance across BMI studies. For instance, the calculation of the index of difficulty does not incorporate target hold time, an important aspect which undoubtedly affects the difficulty of the task (we have therefore listed the target hold time for each study as an additional indicator of task difficulty). However, despite its limitations, the throughput metric captures important aspects of target acquisition tasks such as target size, target distance, and reach time, and it therefore allows us to perform approximate comparisons between studies. Overall, with respect to the throughput metric, we found that our LFP-based BMI performance compares favorably with that of spike-based studies, and exceeds the performance of other studies using LFPs, ECoG, and EEG.

Practical considerations for neuroprosthetics

Local field potentials can potentially augment single-unit activity as control signals for neuroprosthetics to increase robustness and longevity. Our paradigm for achieving LFP-based BMI control is an important step forward towards clinically viable neuroprosthetics. We show that BMI control with LFPs can be achieved from initial arbitrary decoder parameters by using CLDA. In a similar study involving closed-loop LFP-based BMI control, Flint and colleagues recorded neural activity while the subject performed the cursor control task using overt arm movements to train a biomimetic decoder. However, this procedure may constrain the subject to modulate neural activity in ways similar to activity evoked during natural arm movements.

Table 5.1: Performance comparison to other BMI cursor reaching studies. Comparison of the BMI performance achieved in this study with a small sampling of recent closed-loop BMI cursor control studies using spikes, LFPs, ECoG, and EEG. Throughput in bits per second is computed by calculating an index of difficulty (measured in bits) based on task settings and dividing by the mean reach time (measured in seconds). The index of difficulty of a task is calculated based upon the distance to targets and the target size (see Methods).

Neural signal	Study	Target hold time (ms)	Index of difficulty (bits)	Reach time (s)	Throughput (bits/s)
Spikes	Taylor et al., 2002 [5]	N/A	0.80	1.50	0.53
	Kim et al., 2008 [17]	500	2.89	5.51	0.52
	Ganguly & Carmena, 2009 [13]	100	2.37	2.30	1.03
	Orsborn et al., 2012 [28]	400	1.36	1.23	1.10
	Gilja et al., 2012 [27]	500	1.07	0.59	1.81
LFP	Flint et al., 2013 [58]	100	N/A	N/A	0.73
	Present study (Monkey S)	400	1.27	1.31	0.97
	Present study (Monkey J)	400	1.27	1.35	0.94
ECoG	Schalk et al., 2008 [70]	0	1.71	2.13	0.80
EEG	Wolpaw et al., 2004 [71]	0	1.18	1.90	0.62

Our BMI paradigm allows the decoder's parameters to be uniquely optimized for the specific types of spectral power modulations evoked by the subject during closed-loop control. Using CLDA and an assistive control paradigm, both monkeys in this study achieved proficient performance with different ranges of LFP frequencies. Such flexibility in the neural features used for closed-loop BMI control could be important for patients with deficits in their ability to volitionally modulate certain LFP frequencies.

Chapter 6

Achieving rapid closed-loop decoder adaptation: Recursive Maximum Likelihood algorithm

The work presented in this chapter was performed in collaboration with Suraj Gowda, Helene G. Moorman, Amy L. Orsborn, Kelvin So, Maryam Shanechi, and Jose M. Carmena, and was published in Neural Computation [72].

6.1 Introduction

A CLDA algorithm that enables rapid performance acquisition could present a variety of advantages in certain situations. When initial BMI performance with a seed decoder is poor, subjects may lose motivation and become disengaged from the task if performance improves too slowly. However, by using a CLDA algorithm during an initial calibration or training session that provides them with improved BMI performance more quickly, subjects would be more likely to remain engaged. After initial decoder training, such an algorithm could also prove useful in helping to maintain a high level of performance. For example, in a scenario where the user must participate in a periodic (e.g., daily) decoder adjustment procedure, such an algorithm could make this process much less burdensome by shortening the required recalibration time. Furthermore, if sudden channel loss or electrode array shifts occur that disrupt BMI operation, it would be desirable to have an algorithm that could allow performance to recover rapidly.

One potential way to achieve rapid acquisition or recovery of performance is to design a CLDA algorithm that adapts decoder parameters on a fast time-scale by performing "continuous adaptation" — in other words, updating parameters at every decoder iteration. Here, we introduce a Recursive Maximum Likelihood (RML) algorithm that formulates the adaptation of KF parameters in terms of a weighted maximum likelihood estimation problem, which naturally allows more recently observed data to be more influential than past data in decoder updates. We demonstrate that RML possesses a variety of useful properties and practical algorithmic advantages. First, we show how unlike some continuously adaptive methods such as stochastic gradient descent that can produce noisy individual updates, RML leverages the accuracy of updates based on a batch of data while still adapting parameters on every iteration. Second, we illustrate how the algorithm's rate of adaptation is parametrized by an easily interpretable "half-life" parameter that, unlike typical batch-based algorithms, can be adjusted in real-time. Third, we show how even if the number of neural features is very large, RML's memory efficient recursive update rules can be reformulated to avoid costly matrix inversions so that continuous adaptation remains feasible. Finally, although RML is designed for continuous adaptation, we illustrate how RML's update rules are naturally modified to perform batch updates within the same algorithmic framework. We then test the RML algorithm in closed-loop experiments with three macaque monkeys trained to perform a 2-D center-out cursor control task using either spiking activity or local field potentials. In comparison to SmoothBatch [28], a previous CLDA algorithm that adapts parameters on a more intermediate time-scale, we show that RML achieves higher levels of performance following a short period of adaptation, with an average across subjects of 16% lower movement error, 18% lower movement variability, 17% lower reach times, and 13% lower normalized path length. Overall, our results demonstrate that RML is an effective CLDA algorithm for achieving rapid performance acquisition using continuous adaptation.

6.2 Neurophysiological Methods

6.2.1 Electrophysiology

Spiking activity (monkeys C and J) and local field potentials (monkey S) were used for BMI control in this study. All three monkey subjects were adult male rhesus macaques that were chronically implanted with bilateral microwire arrays of 128 Teflon-coated tungsten electrodes (35 μm diameter, 500 μm wire spacing, 8×16 array configuration; Innovative Neurophysiology, Durham, NC), targeting the arm areas of primary motor cortex (M1) and dorsal premotor cortex (PMd). Monkey C was also implanted bilaterally with 64-channel arrays (similar to those previously mentioned) in ventral premotor cortex (PMv), although units from these arrays were not used for BMI control. All procedures were conducted in compliance with the NIH Guide for Care and Use of Laboratory Animals and were approved by the University of California–Berkeley Institutional Animal Care and Use Committee.

Neural activity was recorded using an OmniPlex system (Plexon Inc., Dallas, TX) for monkey C and a MAP system (Plexon Inc.) for monkeys J and S. For monkey C, spiking activity was sorted using the PlexControl online sorting application (Plexon, Inc.), and only neural activity with well-identified waveforms (high signal-to-noise ratio and low variability in waveform shape) were used for BMI control. For monkey J, channel-level activity [73] was defined by setting thresholds for each channel at 5.5 standard deviations from the mean while the subject sat quietly for 2 minutes at the start of each session. The SortClient sorting application (Plexon, Inc.) was then used to define unit templates, which typically captured all threshold crossings, in order to reject non-neuronal artifacts during BMI sessions. For monkey S, local field potential (LFP) signals were sampled at 1 kHz with channels referenced to ground, and signal quality was visually inspected each day (channels with clear artifacts were removed).

6.2.2 Behavioral task

The monkeys were head-restrained in a primate chair and had their arms confined within the chair while performing a self-paced 2-D center-out task. Monkeys were previously trained to perform this task using their right arm. Figure 6.1 depicts the task structure and trial timeline. Trials were initiated by moving the cursor under neural control to the center target and holding for 300 ms, after which the monkeys had to move the cursor to one of eight peripheral targets and hold for 300 ms to receive a liquid reward (target radii = 1.2 cm). If the monkeys entered a peripheral target but left before completing the required hold duration, a target hold error was assessed and no reward was given. The distance from the center to a peripheral target was 10 cm for monkey C and 6.5 cm for monkeys J and S. After every trial, the monkeys were required to move the cursor back to the center target to initiate the next trial. If the monkeys failed to hold or reach the target within 10 s, the trial was restarted without reward.

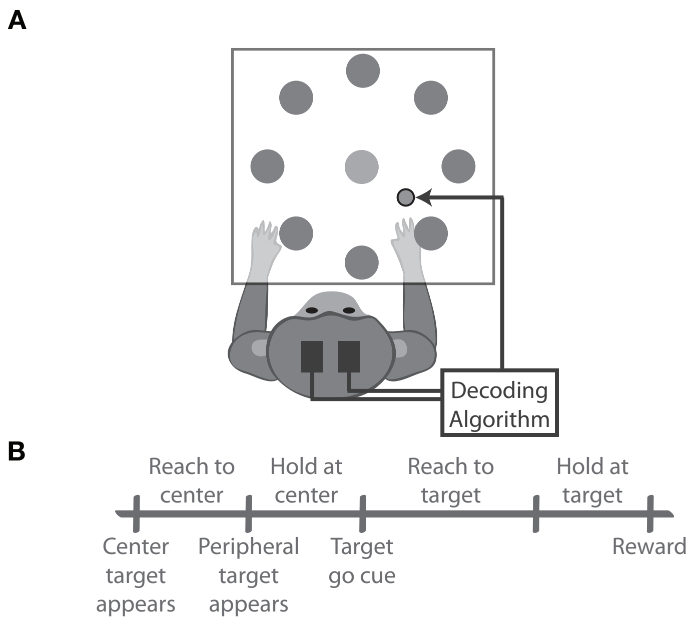


Figure 6.1: **Experimental setup.** (A) Illustration of the BMI task set-up. (B) Trial timeline for the BMI center-out task.

6.2.3 Feature extraction

Neural data was streamed to a dedicated computer running MATLAB (The Mathworks, Natick, MA) or Python to implement feature extraction and closed-loop BMI control. For monkeys C and J, spike counts binned at 100 ms were passed into the decoder as neural features for closed-loop BMI control. For monkey S, the LFP spectral power in consecutive 10-Hz bands from 0–80 Hz was estimated for each channel (in a subset of 20 channels) using the multi-taper method. Spectral estimation was performed every 100 ms using a sliding window containing the most recent 200 ms of raw LFP activity. These log spectral power estimates, across multiple frequency bands and LFP channels, were then used as neural features for closed-loop BMI control. See [48] for more details on LFP-based BMI methodology.

6.2.4 Performance evaluation

We used the following metrics to assess the quality of successful reach trajectories:

1. Movement Error (ME; cm). The average deviation of movement perpendicular to the reach direction; measures the straightness of the reach trajectory.

2. Movement Variability (MV; cm). The standard deviation of movement errors perpendicular to the movement direction; measures the consistency of the reach trajectory.

3. Reach Time (RT; secs). The time between the go cue and entering the peripheral target; measures the speed of the reach movement.

4. Normalized Path Length (NPL; arbitrary units). The distance traveled between leaving the center and entering the target, divided by the straight-line distance; measures the "extra" distance traversed relative to a straight-line trajectory.

6.3 Computational Methods

6.3.1 Decoder model

We used a Kalman filter (KF) decoding algorithm ("decoder") to implement closed-loop BMI control (see Chapter 2 for more details). Here, we focus on BMI cursor control, although our methods apply more generally to other types of BMI systems. In our experiments, the vector of neural observations y_t was composed of either spike counts binned at 100 ms (monkeys C and J) or log spectral power estimates in consecutive 10-Hz bands from 0–80 Hz (monkey S). See Section 6.2.3 for more details on neural feature extraction.

The KF model is parametrized by the matrices $\{A, W, C, Q\}$. The transition model parameters A and W were trained using a data set of arm movements collected while the subject performed the center-out task in manual control, and these parameter estimates were used across all sessions. We constrained the structures of A and W so that these state transition model parameters modeled position as the integral of velocity plus noise [27]. In contrast, the observation model parameters C and Q were initialized ("seeded") differently during each session using one of the following two methods: 1) Visual Feedback, or 2) Shuffled (see Section 2.3 for more details).

When performing decoder adaptation during the initial training phase of a session, only the C and Q parameters of the KF were updated by the RML algorithm, since these parameters model the mapping between intended cursor movements and observed neural activity. Since A and W model cursor dynamics, which were not expected to vary from session to session, these parameters were held fixed during all sessions [42].

In conjunction with CLDA, we simultaneously used an assistive control paradigm [62] during the initial decoder training only to temporarily assist the cursor towards the target. In this phase, the cursor trajectory was determined by:

$$\overrightarrow{v_{cursor}} = \alpha \cdot \overrightarrow{v_{assist}} + (1-\alpha) \cdot \overrightarrow{v_{user}}$$

where $\overrightarrow{v_{user}}$ is the decoded output from the Kalman filter, $\overrightarrow{v_{assist}}$ is a vector that points directly towards the current target, and $\overrightarrow{v_{cursor}}$ is a weighted average of the two that determines the final cursor output shown to the subject. The magnitude of $\overrightarrow{v_{assist}}$ was set to correspond to a speed of 2 cm/s if the cursor was not currently inside a target, and 0 cm/s otherwise. The weighted average was set by the assist level $\alpha \in [0, 1]$. The starting assist level (typically 0.6) was linearly decreased concurrently with CLDA, reaching 0 at the same time that CLDA was ceased. All performance analyses were conducted on data with no assist and with the cursor under full volitional control by the subjects.

6.3.2 Estimating intended movements

In the context of BMI cursor control, a Kalman filter decoder outputs a sequence $\{\hat{x}_t\}$ of decoded cursor kinematics over time. If the decoder's parameters are not yet optimized or the user has not learned to use the decoder for accurate cursor control, the user will likely make movement errors when attempting to control the cursor. In order to perform closed-loop decoder adaptation during a calibration phase, one could estimate the user's intended cursor kinematics over time, which we will denote as $\{\tilde{x}_t\}$. Multiple methods for estimating intended kinematics have been presented in the literature [27, 30, 43, 74]. In the present study, we chose to use Gilja et al.'s method ("innovation 1" of the ReFIT-KF algorithm), which assumes that the user always intends to move the cursor directly towards the next target [27]. Given estimates $\{\tilde{x}_t\}$ of the user's intended cursor kinematics and simultaneously recorded neural features $\{y_t\}$, a CLDA algorithm such as RML can then update the KF decoder's parameters in order to make future decoded outputs more accurately reflect the user's underlying intention.

6.3.3 Recursive Maximum Likelihood (RML) CLDA algorithm

Here, we introduce the Recursive Maximum Likelihood (RML) algorithm for adapting the parameters of a KF decoder during closed-loop control. Let us assume that we have a method for generating estimates of the user's intended cursor kinematics at each time-step. Let $\tilde{x}_{1:n}$ and $y_{1:n}$ represent the collection of these intended kinematics and observed neural activity, respectively, up to iteration n:

$$\begin{aligned}\tilde{x}_{1:n} &\triangleq \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n\} \\ y_{1:n} &\triangleq \{y_1, y_2, \ldots, y_n\}\end{aligned}$$

Let $\theta^{(n)} = \left\{C^{(n)}, Q^{(n)}\right\}$ represent the current KF observation model parameters that are used as part of the BMI decoder on iteration n. In order to fully specify the RML algorithm, we must specify update rules that dictate how to produce $\theta^{(n+1)}$. To derive these update rules, let us consider the log likelihood function $l\left(\theta^{(n+1)}; y_{1:n}, \tilde{x}_{1:n}\right)$, which is defined as the log probability of observing the neural activity $y_{1:n}$ if this data originated from a model parametrized by $\theta^{(n+1)}$ (and $\tilde{x}_{1:n}$ was the corresponding set of intended cursor kinematics):

$$\begin{aligned}l\left(\theta^{(n+1)}; y_{1:n}, \tilde{x}_{1:n}\right) &= \log p\left(y_{1:n} | \theta^{(n+1)}, \tilde{x}_{1:n}\right) \\ &= \sum_{t=1}^{n} \log p\left(y_t | \theta^{(n+1)}, \tilde{x}_t\right)\end{aligned}$$

Here, one option (known as maximum likelihood estimation) would be to update the KF parameters by setting $\theta^{(n+1)}$ to be the parameters that maximize the log likelihood:

$$\theta^{(n+1)} = \arg\max_{\theta} \sum_{t=1}^{n} \log p\left(y_t | \theta, \tilde{x}_t\right)$$

In this approach, all data points from $t = 1$ to n contribute equally to the objective. However, since more recently observed data is often more relevant for accurate parameter estimation than past data, it is often more desirable to assign greater weight or importance to more recent data points. By doing so, one can naturally allow more recently observed data to be more influential in decoder updates. Therefore, let us instead consider a weighted log likelihood objective function $l_w(\cdot)$ that contains a monotonically increasing weighting function $\beta(t)$:

$$l_w\left(\theta^{(n+1)}; y_{1:n}, \tilde{x}_{1:n}\right) = \sum_{t=1}^{n} \beta(t) \log p\left(y_t | \theta^{(n+1)}, \tilde{x}_t\right)$$

For the RML algorithm, we choose an exponentially decaying weighting function:

$$\beta(t) = \lambda^{n-t} \tag{6.1}$$

where $\lambda \in (0,1]$. RML's update rules for C and Q are then derived by solving the following weighted maximum likelihood problem:

$$\theta^{(n+1)} = \arg\max_{\theta} \sum_{t=1}^{n} \lambda^{n-t} \log p\left(y_t | \theta, \tilde{x}_t\right) \tag{6.2}$$

Since $y_t|\theta, \tilde{x}_t \sim \mathcal{N}\left(C\tilde{x}_t, Q\right)$, we have that:

$$\log p\left(y_t|\theta, \tilde{x}_t\right) = \text{const.} - \frac{n}{2}\log|Q| - \frac{1}{2}\sum_{t=1}^{n}\lambda^{n-t}\left(y_t - Cx_t\right)^T Q^{-1}\left(y_t - Cx_t\right)$$

We can then calculate the derivatives of the weighted log likelihood objective with respect to C and Q:

$$\begin{aligned}
\frac{\partial}{\partial C}\sum_{t=1}^{n}\lambda^{n-t}\log p\left(y_t|\theta,\tilde{x}_t\right) &= Q^{-1}\sum_{t=1}^{n}\lambda^{n-t}\left(y_t - C\tilde{x}_t\right)\tilde{x}_t^T \\
\frac{\partial}{\partial Q}\sum_{t=1}^{n}\lambda^{n-t}\log p\left(y_t|\theta,\tilde{x}_t\right) &= -\frac{n}{2}Q^{-1} + \frac{1}{2}Q^{-1}\left(\sum_{t=1}^{n}\lambda^{n-t}\left(y_t - C\tilde{x}_t\right)\left(y_t - C\tilde{x}_t\right)^T\right)Q^{-1}
\end{aligned}$$

Setting these derivatives equal to 0 and solving, we therefore have the following basic rules for updating C and Q after the n^{th} iteration of the Kalman filter:

$$C^{(n+1)} = \left(\sum_{t=1}^{n}\lambda^{n-t}y_t\tilde{x}_t^T\right)\left(\sum_{t=1}^{n}\lambda^{n-t}\tilde{x}_t\tilde{x}_t^T\right)^{-1} \tag{6.3}$$

$$Q^{(n+1)} = \frac{1-\lambda}{1-\lambda^n}\sum_{t=1}^{n}\lambda^{n-t}\left(y_t - C^{(n+1)}\tilde{x}_t\right)\left(y_t - C^{(n+1)}\tilde{x}_t\right)^T \tag{6.4}$$

6.3.4 Recursive update rules on sufficient statistics

One of the main advantages of choosing the exponentially decaying weighting function $\beta(t)$ in equation 6.1 is that the resulting update rules (equations 6.3 and 6.4) can be re-written in a simpler, recursive form so that they do not require storing histories $\tilde{x}_{1:n}$ and $y_{1:n}$. Let us define new parameters R, S, T, and EBS ("effective batch size") as:

$$\begin{aligned}
R^{(n+1)} &\triangleq \sum_{t=1}^{n}\lambda^{n-t}\tilde{x}_t\tilde{x}_t^T \\
S^{(n+1)} &\triangleq \sum_{t=1}^{n}\lambda^{n-t}y_t\tilde{x}_t^T \\
T^{(n+1)} &\triangleq \sum_{t=1}^{n}\lambda^{n-t}y_t y_t^T \\
\text{EBS}^{(n+1)} &\triangleq \sum_{t=1}^{n}\lambda^{n-t}
\end{aligned}$$

Then, by substituting these definitions into equations 6.3 and 6.4, it is straightforward to show that we can express $C^{(n+1)}$ and $Q^{(n+1)}$ exclusively in terms of these new parameters as:

$$\begin{aligned} C^{(n+1)} &= S^{(n+1)} \left(R^{(n+1)}\right)^{-1} && (6.5) \\ Q^{(n+1)} &= \frac{1}{\mathrm{EBS}^{(n+1)}} \left(T^{(n+1)} - S^{(n+1)} R^{(n+1)^{-1}} S^{(n+1)^T}\right) && (6.6) \end{aligned}$$

From these equations, we see that once we have determined the updated values $R^{(n+1)}$, $S^{(n+1)}$, $T^{(n+1)}$, and $\mathrm{EBS}^{(n+1)}$, then the data $\tilde{x}_{1:n}$ and $y_{1:n}$ are no longer needed to determine $C^{(n+1)}$ and $Q^{(n+1)}$. Therefore, an intuitive interpretation of R, S, T, and EBS is that they are sufficient statistics for the weighted maximum likelihood estimation of C and Q.

The advantage of introducing these sufficient statistics — rather than adapting C and Q directly as in equations 6.3 and 6.4 — is that they are easily updated at every iteration in a recursive fashion. For example, for $S^{(n+1)}$ we have:

$$\begin{aligned} S^{(n+1)} &= \sum_{t=1}^{n} \lambda^{n-t} y_t \tilde{x}_t^T \\ &= \sum_{t=1}^{n-1} \lambda^{n-t} y_t \tilde{x}_t^T + y_n \tilde{x}_n^T \\ &= \lambda \left(\sum_{t=1}^{n-1} \lambda^{n-1-t} y_t \tilde{x}_t^T \right) + y_n \tilde{x}_n^T \\ &= \lambda S^{(n)} + y_n \tilde{x}_n^T \end{aligned}$$

and likewise for the others. Therefore, in order to determine the updated values $C^{(n+1)}$ and $Q^{(n+1)}$, we first simply update $R^{(n)}$, $S^{(n)}$, $T^{(n)}$, and $\mathrm{EBS}^{(n)}$ recursively as follows:

$$\begin{aligned} R^{(n+1)} &= \lambda R^{(n)} + \tilde{x}_n \tilde{x}_n^T && (6.7) \\ S^{(n+1)} &= \lambda S^{(n)} + y_n \tilde{x}_n^T && (6.8) \\ T^{(n+1)} &= \lambda T^{(n)} + y_n y_n^T && (6.9) \\ \mathrm{EBS}^{(n+1)} &= \lambda \mathrm{EBS}^{(n)} + 1 && (6.10) \end{aligned}$$

and then update C and Q as in equations 6.5 and 6.6. This parameter update procedure is depicted in Figure 6.2.

In some continuously adaptive methods such as stochastic gradient descent, parameters are updated at each iteration based on a single data point. Importantly, however, such methods do not necessarily guarantee the accuracy of individual updates, but rather rely on the successive combination of multiple updates to produce accurate adjustments to parameters. While RML also performs continuous adaptation and uses a single data point at each iteration in the update equations 6.7–6.10, those equations are used to update the RML sufficient statistics, not the decoder parameters. Indeed, since the updated sufficient statistics are then subsequently used update the decoder, each overall decoder update still represents the solution to a weighted batch maximum likelihood estimation problem. In this way, RML is able to leverage the accuracy of updates based on a batch of data, while still adapting parameters on every iteration.

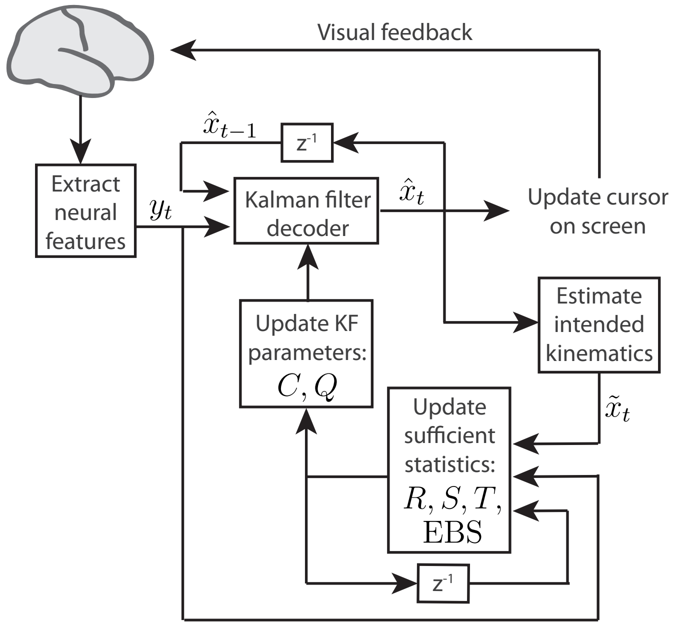


Figure 6.2: **RML algorithm block diagram.** Block diagram illustrating how the RML algorithm is used to adapt the parameters of a Kalman filter decoder. z^{-1} blocks represent delay elements. Unlike other CLDA algorithms for KF decoders, RML first updates a set of sufficient statistics (R, S, T, and EBS) before updating the actual KF C and Q matrices directly.

6.3.5 Half-life reparametrization

Another advantage of the recursive form of RML's update rules for the sufficient statistics is that λ can be reparametrized more conveniently in terms of an intuitive half-life parameter h. For instance, starting from equation 6.8 and using repeated substitution, one can show that for any number of iterations $k \geq 0$, we have that:

$$S^{(n+k)} = \sum_{t=0}^{k-1} \lambda^{k-1-t} y_{n+t} \tilde{x}_{n+t}^T + \lambda^k S^{(n)}$$

In other words, we see that influence of the value $S^{(n)}$ in the future version of the parameter $S^{(n+k)}$ decreases exponentially in k. By calculating when the factor λ^k is equal to one-half, we can define a half-life h as:

$$\lambda^{\frac{h}{\mathrm{dt}}} = \frac{1}{2}. \tag{6.11}$$

where dt is the iteration period at which the Kalman filter decoder is being run (e.g., 100 ms). One of the advantages of a continuously adaptive CLDA algorithm like RML is the ability to instantaneously adjust adaptation rates, such as the half-life h. In our closed-loop experiments testing the RML algorithm (see Section 6.4), we start with a relatively low half-life or 30 s, which we continuously increase over the course of CLDA to a final value of 300 s. In this way, we begin CLDA by making large initial parameter updates that bring the parameters into the right "ballpark", while smoothly transitioning towards more conservative, fine-tuned adaptation as decoder parameters begin to converge. While using a time-varying half-life for SmoothBatch has also been proposed in previous work [28, 42], it has not yet been tested in closed-loop spike-based experiments. Therefore, in our BMI experiments, we compare RML against both the standard form of SmoothBatch with a non-changing half-life (as originally tested in [28]) and SmoothBatch with a time-varying (albeit not continuously) half-life.

6.3.6 Avoiding costly matrix inversions

The actual Kalman filter equations that perform state estimation require the computation of a Kalman gain matrix on every iteration of filter. For instance, on iteration $n + 1$, the Kalman gain is computed as:

$$K_{n+1} = P_{n+1|n} C^{(n+1)^T} \left(C^{(n+1)} P_{n+1|n} C^{(n+1)^T} + Q^{(n+1)} \right)^{-1} \tag{6.12}$$

where $P_{n+1|n}$ is the a priori estimation error covariance matrix [26]. If C and Q are being adapted by a CLDA algorithm that does not update parameters on every iteration, then

$P_{n+1|n}$ converges quickly to a constant matrix P after each update and repeated computation of the inverse in equation 6.12 can be avoided until the next update, since the inverse does not change. However, for a CLDA algorithm that performs continuous adaptation of C and Q, this inverse is constantly changing and needs to be calculated on every iteration. When the number of neural features (which corresponds to the number of rows/columns of the matrix that needs to be inverted) is large, then the resulting matrix inversion calculation may be too computationally expensive to be completed within a typical BMI decoder loop time (e.g., 50–100 ms). For example, in a LFP-based BMI system where 20 features are being extracted on each of 50 LFP channels, a 1000×1000 matrix would need to be inverted.

To significantly reduce the computation involved in the Kalman gain calculation and make a continuously adaptive CLDA algorithm like RML more computationally feasible when the number of neural features is large, we can apply a formula known as the Woodbury matrix identity [75]. By applying this identity to the matrix inversion in the Kalman gain, we can compute this inverse in an alternate way that requires inverting a matrix of the same size as $P_{n+1|n}$, which is typically very small (only 5×5 in position-velocity KF decoders):

$$\begin{aligned} K_{n+1} &= P_{n+1|n} C^{(n+1)^T} Q^{(n+1)^{-1}} \times \Big[I - C^{(n+1)} \cdot \\ &\qquad \left(P_{n+1|n} + C^{(n+1)^T} Q^{(n+1)^{-1}} C^{(n+1)} \right)^{-1} C^{(n+1)^T} Q^{(n+1)^{-1}} \Big] \end{aligned} \tag{6.13}$$

Nonetheless, this alternate formula still requires the computation of Q^{-1} on every iteration, which would mean that a continuous adaptation approach would still be infeasible if Q is a large matrix. However, two further applications of the Woodbury matrix identity can make RML a viable CLDA algorithm. First, by taking inverses on both sides of RML's update rule for Q and then applying the Woodbury identity, we have that:

$$\begin{aligned} Q^{(n+1)^{-1}} &= \mathrm{EBS}^{(n+1)} \cdot \Big[T^{(n+1)^{-1}} - T^{(n+1)^{-1}} S^{(n+1)} \cdot \\ &= \quad \left(S^{(n+1)^T} T^{(n+1)^{-1}} S^{(n+1)} - R^{(n+1)} \right)^{-1} S^{(n+1)^T} T^{(n+1)^{-1}} \Big] \end{aligned} \tag{6.14}$$

Second, if we apply the same identity to the update rule for T, we have that

$$\begin{aligned} T^{(n+1)^{-1}} &= \left(\lambda T^{(n)} + y_n y_n^T \right)^{-1} \\ &= \frac{T^{(n)^{-1}}}{\lambda} - \frac{T^{(n)^{-1}} y_n y_n^T T^{(n)^{-1}}}{\lambda \left(\lambda + y_n^T T^{(n)^{-1}} y_n \right)} \end{aligned}$$

Therefore, rather than updating Q at every iteration and then having to calculate its inverse to run the next iteration of KF equations, we can instead efficiently update T^{-1} directly, which allows us to update Q^{-1} efficiently and use this value directly in equation 6.13. Therefore, the above alternate update rules for Q^{-1} and T^{-1} can be used to achieve more efficient decoder updates when computational resources are scarce.

6.3.7 Generalization to batch-based parameter updates

Although so far we have introduced RML exclusively as a CLDA algorithm for continuous adaptation, RML's update rules can be easily modified to perform batch-based updates within the same algorithmic framework. Indeed, if we modify the weighted maximum likelihood estimation problem in equation 6.2 so that consecutive groups of $N \geq 1$ data points are weighted equally, then it is straightforward to show that we arrive at the following analogous update rules for our sufficient statistics:

$$\begin{aligned}
R^{(n+N)} &= \lambda R^{(n)} + \sum_{t=0}^{N-1} \tilde{x}_{n+t}\tilde{x}_{n+t}^T && (6.15)\\
S^{(n+N)} &= \lambda S^{(n)} + \sum_{t=0}^{N-1} y_{n+t}\tilde{x}_{n+t}^T && (6.16)\\
T^{(n+N)} &= \lambda T^{(n)} + \sum_{t=0}^{N-1} y_{n+t}y_{n+t}^T && (6.17)\\
\text{EBS}^{(n+N)} &= \lambda \text{EBS}^{(n)} + N && (6.18)
\end{aligned}$$

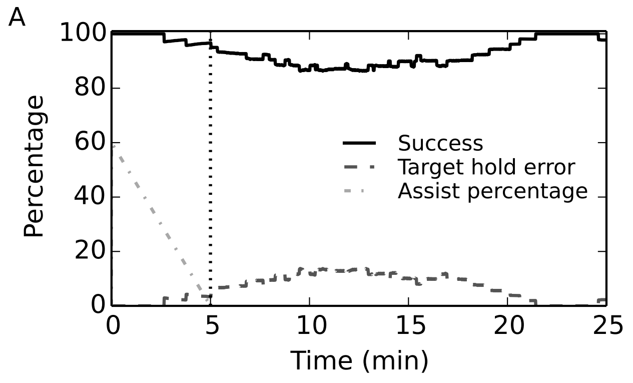
As before, C and Q are still updated according to equations 6.5 and 6.6 whenever these sufficient statistics are updated. Intuitively, when $N = 1$, these update rules are equivalent to those presented before in equations 6.7–6.10. When $N > 1$, the λ weighting factor is reparametrized into a half-life h according to the following modified equation:

$$\lambda^{\frac{h}{N \cdot \mathrm{dt}}} = \frac{1}{2}. \qquad (6.19)$$

6.4 Results

6.4.1 RML decoder adaptation

Across all three subjects, the RML CLDA algorithm rapidly and reliably lead to acquisition of high performance after CLDA ceased and decoder parameters were held fixed. Figure 6.3 illustrates monkey J's performance during a representative session where the decoder was seeded using the Shuffled procedure (see Section 6.3.1), resulting in an initial decoder with which the monkey could not perform successful trials. A short period (5 minutes) of RML adaptation was then performed. An assistive control paradigm was used concurrently with CLDA, which accounts for the subject's apparent high performance from the start of the session. Upon turning off assist and ceasing decoder adaptation (dotted vertical line), the subject was able to maintain performance under full volitional control. Figure 6.3C depicts the subject's typical reach trajectories to the eight center-out targets using the fixed (i.e., unchanging) decoder.



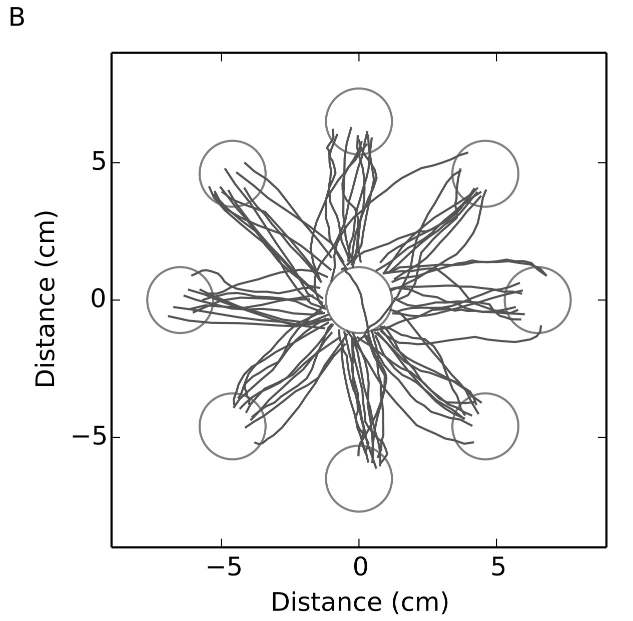


Figure 6.3: **Closed-loop BMI performance with RML.** Example application of the RML algorithm (representative session from monkey J), starting from a KF decoder (seeded using the Shuffled method) with which the subject could not perform successful trials. The subject's apparent high performance from the start is due to the the assistive paradigm that was used concurrently with CLDA. Performance is plotted in terms of (A) percentage and (B) rate of events/min. The dotted vertical line indicates the time at which assistance was turned off, CLDA ceased, and decoder parameters were held fixed. (C) Representative reach trajectories in the center-out task after RML adaptation was performed.

The evolution of decoder parameters during this example session is shown in Figure 6.4. Since our KF state vector models cursor velocity, the rows of the KF C matrix can be used to determine the preferred velocity directions of the different neural features being used for decoding (binned spike counts for monkeys C and J, and LFP spectral power for monkey S). By plotting the preferred velocity vectors for each neural feature as modeled in C, we can visualize the direction tuning of different units in the decoder and observe how the decoder changes over time as CLDA is performed. Figure 6.4A shows the preferred velocity directions of the Shuffled seed decoder, which the subject is unable to perform successful trials. As RML decoder adaptation is performed, the C matrix is rapidly updated to more closely reflect the true velocity tuning (Figures 6.4B–D). By the end of adaptation (Figure 6.4E–F), the decoder's parameters begin to converge, leading to a final KF decoder that the subject could use to skillfully control the cursor in the center-out task even after CLDA and assist were ceased.

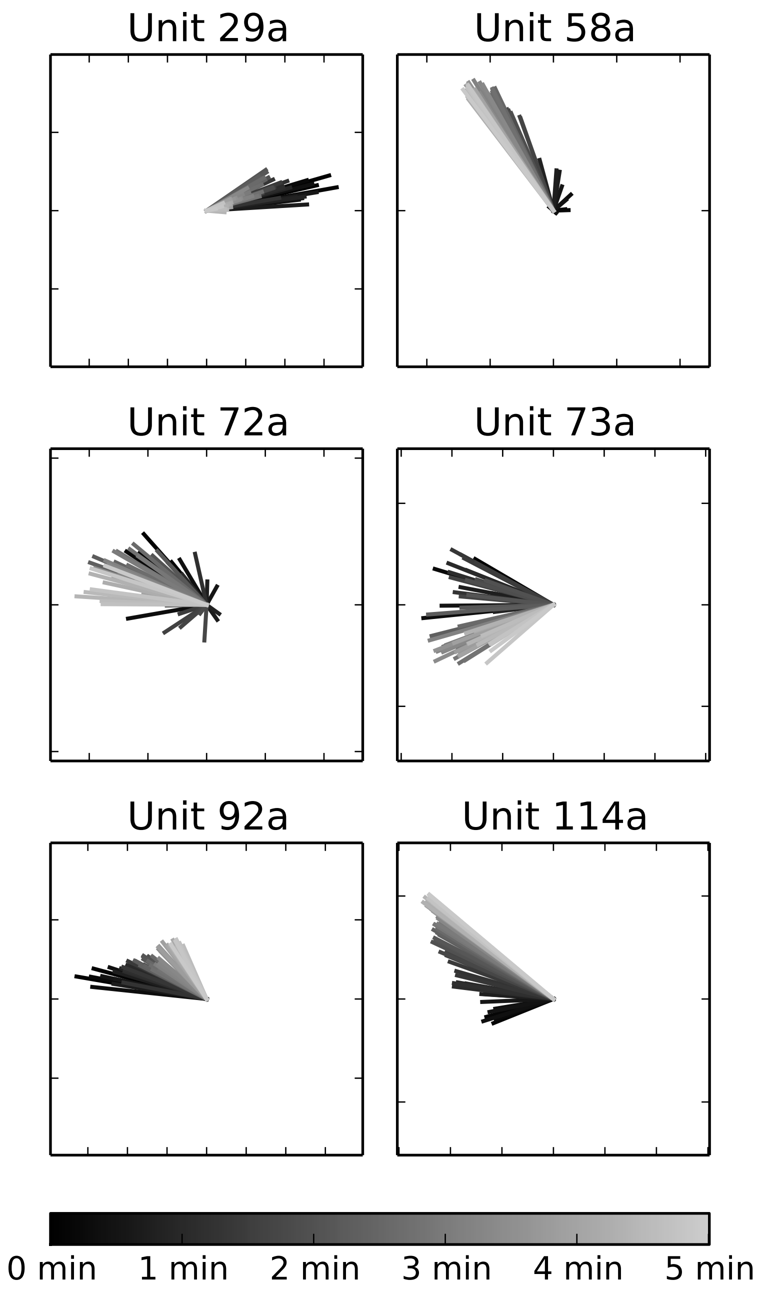


Figure 6.4: **Decoder evolution during RML adaptation.** (A–F) Tuning plots representing the evolution of the decoder's model of preferred velocity vectors of the different units being used for BMI control, as CLDA is performed. This sequence of tuning evolution corresponds to the same BMI session plotted in Figure 6.3.

6.4.2 Performance comparison to SmoothBatch

In order to measure the performance of the RML CLDA algorithm, we compared it against SmoothBatch, a previously developed algorithm that has been demonstrated to rapidly improve BMI performance independent of the decoder's initialization method [28]. In each session, starting from a Shuffled decoder seeding, a KF decoder was adapted for a short amount of time (5 minutes; "short adaptation"), once each using either the RML or SmoothBatch algorithm (using a random ordering). During CLDA, we simultaneously used an assistive control paradigm (see Section 6.3.1) in which assist started at 60% and decreased linearly to 0% by the end of adaptation. Since RML is a continuously adaptive algorithm, the half-life can be changed at every KF decoding iteration. Therefore, for RML adaptation, a time-varying half-life was used that started at 30 s and increased linearly to 300 s over the course of adaptation. For SmoothBatch, standard CLDA parameters of a 60 s batch size and a 120 s half-life were used as originally described in [28]. (In additional experiments, a time-varying half-life was also used with SmoothBatch — see below.) When decoder adaptation and assist ceased after short adaptation, subjects then used the fixed decoder to perform the 2-D center-out task. Sessions in which the monkeys did not perform successful trials following decoder adaptation are reported below but were not included in statistical significance testing.

The shaded gray bars in Figure 6.5 illustrate the subjects' performance with a fixed decoder after short adaptation with either RML or SmoothBatch (SB) CLDA, averaged over multiple sessions. The corresponding individual session data is plotted in Figure 6.6. For all subjects, performance after RML was better than after SmoothBatch with respect to movement error (ME), movement variability (MV), reach time (RT), and normalized path length (NPL). Monkey C achieved 17% lower ME, 19% lower MV, 24% lower RT, and 16% lower NPL after RML adaptation than after SB CLDA. Performance differences for monkeys J and S were similar (ME: 13% and 18%, MV: 15% and 21%, RT: 15% and 12%, and NPL: 10% and 13%, respectively for J and S). Statistically, these performance differences were significant ($p < 0.05$, one-sided, paired Wilcoxon signed-rank test) for all subjects across all four performance metrics. Furthermore, RML adaptation proved to be more robust than SmoothBatch, as all three subjects were able to successfully perform trials more often after RML adaptation versus after SB adaptation. Out of a total of $n = 12$ sessions, monkey C was unable or unwilling to successfully acquire targets in only 1 session after RML, but this was the case in 6 of the sessions for SB. Monkey J ($n = 16$) and monkey S ($n = 15$) performed the task in all sessions after RML adaptation, but each was unwilling or unable to successfully acquire targets after SB adaptation in 4 of their respective sessions.

We also evaluated the subjects' performance after a longer period of adaptation (10+ minutes; "extended adaptation") with both algorithms (see white bars in Figure 6.5). For this condition, decoders were seeded using the VFB (visual feedback) method. With more extended decoder adaptation and a potentially more favorable decoder seeding, most differences in performance were insignificant (only normalized path length for monkey C was significantly lower after RML than after SB, $p < 0.05$). However, performance after RML adaptation was still better than after SB adaptation in most cases, except for the reach times for monkey C and movement variability for monkey S. Moreover, all three monkeys performed successful trials after RML adaptation in all sessions ($n = 7$, 11, and 9 for monkeys C, J, and S, respectively). However, both monkeys S and J were either unable or unwilling to perform trials in one of their respective sessions after SB adaptation.

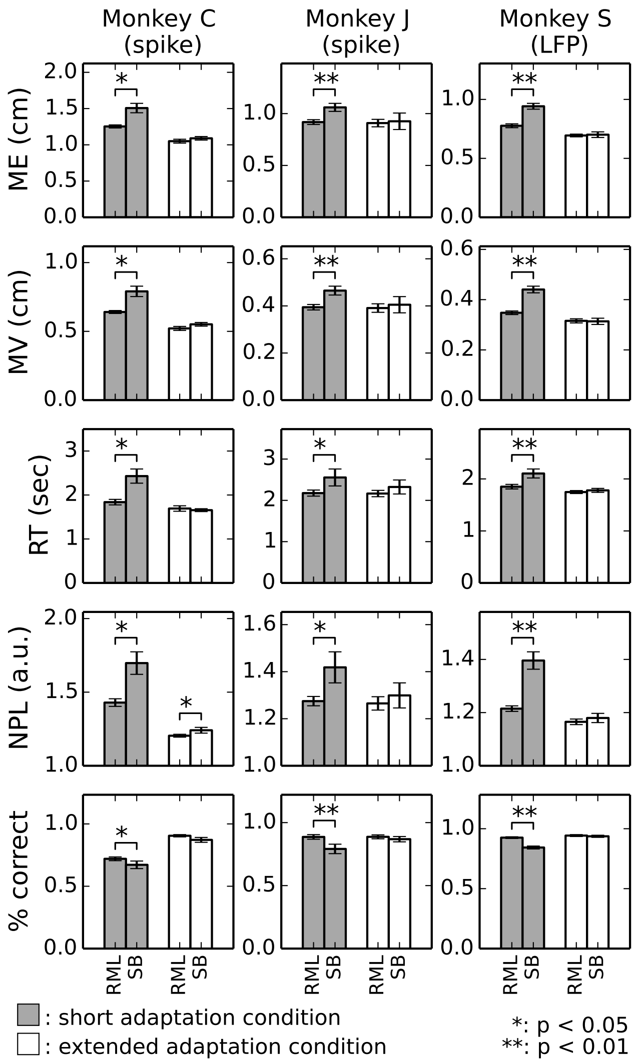


Figure 6.5: **Performance comparison between the RML and SmoothBatch CLDA algorithms (averaged data).** For all three subjects, both RML and SmoothBatch (SB) adaptation were performed starting from identical decoder seedings, across multiple sessions. Shaded gray bars depict performance (averaged across sessions) with a fixed decoder after a short adaptation period (5 minutes), starting from a Shuffled KF seeding. White bars depict performance (averaged across sessions) with a fixed decoder after a more extended adaptation period (10+ minutes), starting from a VFB seeding. Error bars denote standard error of the mean, and asterisks indicate statistical significance ($*$: $p < 0.05$, $**$: $p < 0.01$). Performance was quantified by evaluating four different measures of cursor trajectory quality on successful trials. Smaller values represent better performance for all metrics.

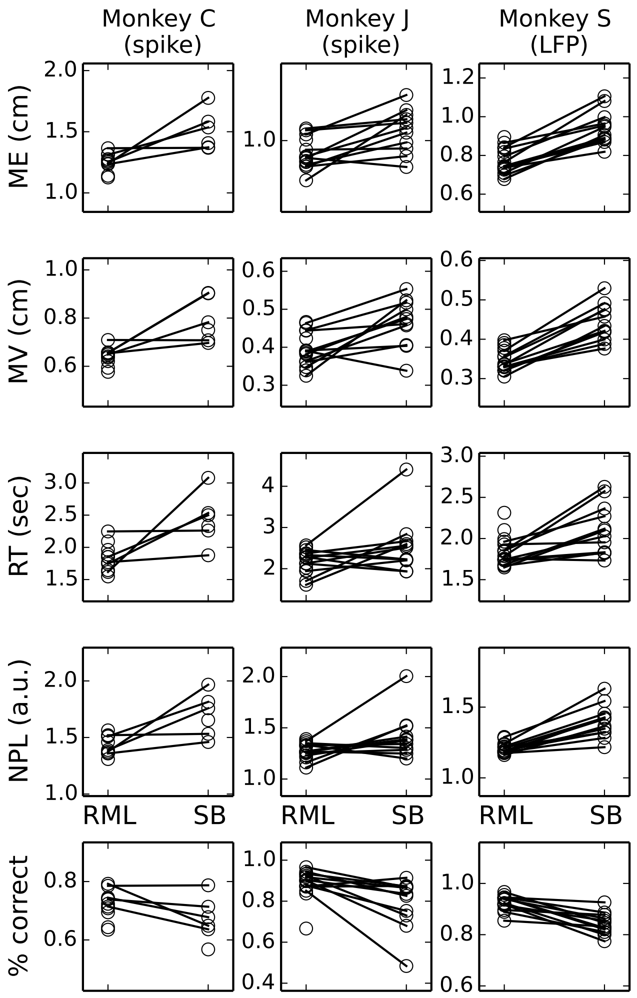


Figure 6.6: **Performance comparison between the RML and SmoothBatch CLDA algorithms (individual session data).** Individual session data used to plot the averaged data in Figure 6.5. The circle markers connected by each line represent the performance after a short period (5 minutes) of RML and SmoothBatch CLDA, starting from a common Shuffled decoder seeding. A circle marker for one CLDA algorithm with no connected line indicates that the monkey did not perform successful trials following CLDA with the other algorithm. Performance is quantified by evaluating four different measures of cursor trajectory quality on successful trials. Smaller values represent better performance for all metrics.

To control for the possibility that the difference in half-life settings between both algorithms might be responsible for the observed performance gap after a short amount of adaptation, we conducted an additional set of experiments in which a time-varying half-life was also used for SmoothBatch adaptation. Although the half-life could not be increased instantaneously (at every KF iteration) as with RML, the half-life schedule for SmoothBatch was set such that the first decoder update occurred with a half-life of 30 s and the last decoder update occurred with a half-life of 300 s, to match the RML condition as closely as possible. Figure 6.7 illustrates the subjects' average performance with a fixed decoder after short adaptation with either RML or SB, both with a time-varying half-life. For monkeys C and S, performance after RML was still better than after SmoothBatch across all four performance metrics, and these differences were significant ($p < 0.05$, one-sided, paired Wilcoxon signed-rank test) for almost all metrics (only the reach times for monkey S after RML were not significantly larger than after SB). For monkey J, reach times and normalized path length were lower after RML than after SB (difference in reach times was significant, $p < 0.05$), and movement error and normalized path length were not significantly different ($p > 0.05$). Overall, RML adaptation still proved to be more robust than SmoothBatch. Notably, while all three subjects always performed successful trials after RML adaptation in every session, they did not perform successful trials after SB adaptation in some sessions (they were either unable or unwilling to do so). In particular, this occurred in 3 out of $n = 7$ sessions for monkey C, 1 out of $n = 8$ sessions for monkey J, and 3 out of $n = 12$ sessions for monkey S.

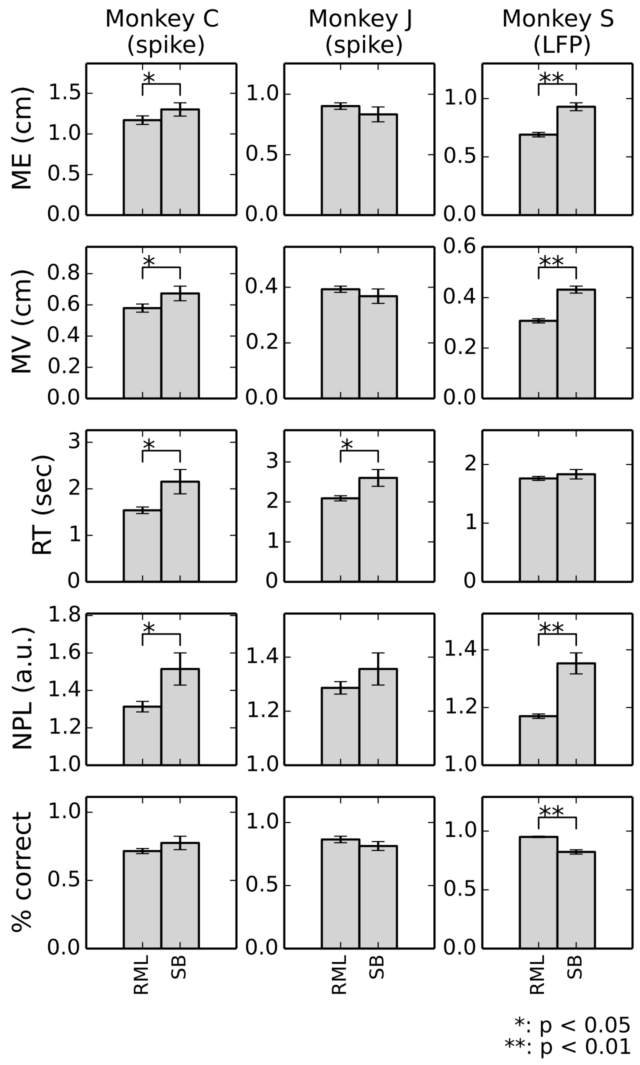


Figure 6.7: **Control comparison between the RML and SmoothBatch CLDA algorithms (averaged data).** Performance (averaged across sessions) with a fixed decoder after a short adaptation period (5 minutes), starting from a Shuffled KF seeding. In contrast to the data plotted in Figure 6.5, where a time-varying half-life was used for RML only, a time-varying half-life was used here for both RML and SmoothBatch. Error bars denote standard error of the mean, and asterisks indicate statistical significance ($*$: $p < 0.05$, $**$: $p < 0.01$). Performance was quantified by evaluating four different measures of cursor trajectory quality on successful trials. Smaller values represent better performance for all metrics.

6.5 Discussion

By developing a continuously adaptive CLDA algorithm that updates parameters on every decoder iteration, we have demonstrated the ability to achieve fast acquisition of BMI performance when starting from potentially adverse seeding conditions. Since we used an assistive control paradigm concurrently with CLDA, the subject appeared to have high performance from the start, making it was difficult to measure how performance improved during the course of decoder adaptation. Therefore, we chose to measure performance by stopping CLDA (and assist) after a fixed amount of time and evaluating the subjects' cursor control with a fixed decoder. When comparing our RML algorithm to a previously developed CLDA algorithm (SmoothBatch) that is not continuously adaptive, we found that 3 macaque monkey subjects using spiking activity or local field potentials achieved higher levels of performance after a relatively short of period of RML adaptation compared to SmoothBatch adaptation. When more extended adaptation was performed, both algorithms reached comparable levels of performance. One of the advantages of a continuously adaptive CLDA algorithm is the ability to instantaneously adjust adaptation rates (such as the half-life h), which we have leveraged in our experiments with RML. By starting with a relatively low half-life (30 s) and gradually increasing this value over the course of CLDA, we aim to make large initial parameter updates that bring the parameters into the right "ballpark", while smoothly transitioning towards more conservative, fine-tuned adaptation as decoder parameters begin to converge. However, our control experiments testing SmoothBatch with a increasing half-life showed that a time-varying half-life by itself does not account for the performance gap after RML versus SmoothBatch CLDA, suggesting that continuous adaptation itself appears to be important for rapid acquisition of performance.

Previous work in brain-machine interfaces and related fields has investigated comparisons between batch-based and continuously adaptive algorithms for improving performance [32, 76, 77]. In brain-machine interfaces, continuously adaptive algorithms have been developed and tested in simulation for different decoder types and neural signal sources [37, 71, 76, 78–80], but not all algorithms fully translated to experimental settings when compared to batch-based methods. For example, in [32], closed-loop experiments demonstrated that CLDA with a continuously adaptive method led to performance improvements, but those improvements occurred relatively slowly and were not fully maintained upon fixing the decoder. In the realm of human-machine interfaces, Danziger et al. conducted experiments in which high-dimensional signals from a wearable glove were transformed to control the joint angles of a simulated two-link robot arm [77]. Two learning algorithms — Moore-Penrose (MP) pseudoinverse (batch) and LMS gradient descent (continuously adaptive) — were applied to adapt the glove-to-robot transformation based on errors measured in past performance. Although the LMS group outperformed a control group while the MP group did not, the LMS subjects failed to achieve better generalization than the control subjects. Generalization was also found to be a problem in [32], where closer inspection of the adapted parameter trajectories revealed that with a continuously adaptive stochastic gradient method, some parameters did not appear to converge, perhaps due to the method's tendency to over-fit on short time-scales. Indeed, in algorithms based on stochastic gradient methods, each individual update to decoder parameters is based only on a single data point and therefore not necessarily guaranteed to be accurate. In contrast, the SmoothBatch algorithm [28] avoids this problem by making updates based on small (1–2 minutes) batches of data. However, it too is partially limited by the fact that its data batches cannot be too small, or else the corresponding batch parameter estimates used as part of the weighted averages in its update rules will become too inaccurate. On the other hand, if the data batches are made to be larger, then the batch parameter estimates will likely become more accurate, but then the user must potentially wait for a long time for the next decoder update to occur. Since the RML algorithm updates its sufficient statistics at each iteration before updating the KF parameters themselves, each RML update is effectively still based on a (weighted) batch of data. As a result, RML is able to perform continuous adaptation while still ensuring that each decoder update results in a more potentially accurate adjustment of parameters than would be achieved if each update was only based on a single data point.

Importantly, the RML algorithm is computationally fast and therefore feasible as a continuously adaptive CLDA algorithm for a wide variety of neural signal sources including spikes, LFP, electrocorticography (ECoG), and electroencephalography (EEG). By first updating a set of sufficient statistics that are then used to adapt the actual decoder parameters, the RML algorithm is expressed in terms of recursive update rules that are computationally simple and memory efficient. Moreover, when the number of neural features being passed into the decoder is large, RML's update rules can be reformulated using the matrix inversion lemma to avoid costly matrix inversions in the KF equations that would otherwise make continuous adaptation infeasible.

Overall, our results with RML indicate that continuous adaptation is indeed a feasible and successful CLDA paradigm for rapid performance acquisition in BMIs. For these types of continuously adaptive algorithms, it is important to determine whether CLDA will synergize well with previous results that have demonstrated the importance of neural plasticity for BMI learning [3, 5, 6, 11, 13, 47]. For instance, Ganguly and Carmena demonstrated that by fixing the parameters of the BMI decoder and keeping neurons stable, a neural map of the decoder forms that is stable across time, can be readily recalled, and is robust to interference from a second learned map [13]. Therefore, even if continuously adaptive CLDA algorithms are used to rapidly improve initial performance, subsequent practice with a fixed decoder could potentially still lead to the formation of a stable map in order to achieve long-term retention of neuroprosthetic skill.

Chapter 7

Conclusion

7.1 Thesis Contributions

This thesis makes a number of important contributions to the brain-machine interface field:

- We developed a CLDA algorithm called SmoothBatch that can rapidly and robustly improved BMI performance regardless of initial closed-loop performance. In particular, we demonstrated the algorithm's ability to improve closed-loop BMI performance independent of whether the initial decoder was seeded using 1) visual observation of cursor movement, 2) ipsilateral arm movement, 3) neural activity during quiet sitting, or 4) arbitrary/shuffled weights. Such an algorithm could be paramount in situations where initial closed-loop performance may be severely limited, such as in clinical BMI applications for patients that cannot enact natural movement because of spinal cord injury or other neurological disorders [28].

- While conducting closed-loop BMI experiments is ultimately the only conclusive way to evaluate a CLDA algorithm's convergence properties, such experiments are lengthy and costly. We introduced common design considerations for CLDA algorithms and demonstrated how mathematical convergence analysis, using measures such as mean-square error (MSE) or KL divergence, can be a useful precursor to closed-loop experiments that can further inform CLDA design and constrain the necessary experimental testing. We applied our analysis to SmoothBatch as an example, and guided by the convergence predictions that followed, we proposed a specific method to improve the SmoothBatch algorithm by allowing for time-varying CLDA parameters [42].

- We demonstrated 2-D closed-loop BMI control using local field potential signals by using the power of closed-loop decoder adaptation to adapt the decoder to the patterns of neural activity elicited during BMI operation. Our work added to the growing body of evidence that field potentials (LFP or ECoG) can be volitionally modulated well enough to accurately control a 2-D cursor. Our post-experimental analysis showed substantial differences in the patterns of LFP modulations between the two subjects

— when allowed a broad range of spectral power features to control the BMI, one subject primarily utilized lower frequencies of the LFP for BMI control, while the other preferred to use higher frequencies. The use of adaptive methods such as CLDA can therefore be a useful tool that can help account for these inter-subject differences [48].

- We developed a CLDA algorithm that enables rapid performance acquisition, which can present a variety of advantages in certain situations. Our algorithm, recursive maximum likelihood (RML), leverages the accuracy of updates based on a batch of data while still both weighting recently observed data more heavily and efficiently adapting parameters on every iteration. RML can help rapidly improve BMI performance when the seed decoder is poor, and can also prove useful in helping to maintain a high level of performance by shortening the required recalibration time during daily BMI operation [72].

7.2 Pitfalls

In previous work, we developed some CLDA algorithms based on gradient descent. The first algorithm, named the Adaptive Kalman Filter (AKF), used gradient-based update rules for the A and C matrices of the Kalman filter, with the rules based on separate mean-squared error objective functions [37]. For the W and Q covariance matrices, the AKF used update rules of heuristic form. In later work, we extended this idea to develop another algorithm called Likelihood Gradient Descent (LGA). In contrast to the AKF, LGA's update rules are based and are derived directly from a single, unified log likelihood objective function. When we compared the performance of the LGA algorithm to the AKF in a 2-D center-out cursor control task using a closed-loop simulator [36], the LGA outperformed the AKF. However, in various closed-loop experiments, e.g., [32], neither of these algorithms performed as well as other algorithms, such as SmoothBatch [28] or RML [72]. One common shortcoming of both LGA and AKF that may explain their observed performance is the highly stochastic nature of their gradient-based updates, as single time-point estimates of the gradient can be quite noisy and inaccurate. Unlike these algorithms that produce noisy individual updates, algorithms like RML that are not gradient-based may likely better leverage the accuracy of updates based on a batch of data, while still being able to adapt parameters on every time step like the AKF and LGA algorithms.

One downside of the RML algorithm is that the same CLDA framework may not extend directly to other decoding models. For instance, when performing maximum likelihood estimation for the parameters of a point-process filter (PPF), not all of the sufficient statistics are in the form of sums like they are for the Kalman filter model. Therefore, the same RML-style recursive update rules on sufficient statistics may not directly apply for the PPF model.

7.3 Future Work

While we have demonstrated high-level 2-D BMI control using only local field potential signals, future work should explore the possibility developing BMI systems that use both spike and LFP signals for prosthetic control. In addition, while we used 10-Hz frequency bands as the basis for our extracted neural LFP features, a different arrangement of frequency bands (e.g., smaller or larger bands, or bands with different sizes) may be more optimal. Furthermore, other types of features entirely (e.g., time-domain features) could lead to higher performance LFP-based control.

Overall, our results with the SmoothBatch and RML CLDA algorithms indicate that CLDA is indeed a feasible and successful paradigm for rapid performance acquisition in BMIs. In the experiments presented in this dissertation, we primarily developed CLDA algorithms for a Kalman filter decoder operating with 100 ms between iterations. While the Kalman filter is currently still a state-of-the-art decoding algorithm for BMI applications, CLDA algorithms for other decoder architectures, should also be explored. Alternative decoding architectures that operate on faster time-scales have been proposed, such as the point-process filter (PPF) [78]. Since the PPF models the occurrence of individual spikes, a PPF decoder would likely need to operate with <5 ms between iterations. Allthough the RML algorithm may not be directly applicable to the PPF, it should be explored whether CLDA on an even faster time-scale within a PPF framework [74] could enable more rapid and accurate acquisition of BMI performance. Furthermore, with the recent re-emergence of artificial neural networks within the machine learning field as a powerful tool for multi-dimensional classification and regression problems, it should be explored whether CLDA algorithms can be developed for neural network decoders to help leverage the power of deep learning for BMI decoding applications.

Bibliography

[1] John K. Chapin, Karen A. Moxon, Ronald S. Markowitz, and Miguel A. L. Nicolelis. Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nature Neuroscience*, 2(7):664–670, July 1999.

[2] Gregory J Gage, Kip A Ludwig, Kevin J Otto, Edward L Ionides, and Daryl R Kipke. Naïve coadaptive cortical control. *Journal of Neural Engineering*, 2005.

[3] Aaron C. Koralek, Xin Jin, John D. Long Ii, Rui M. Costa, and Jose M. Carmena. Corticostriatal plasticity is necessary for learning intentional neuroprosthetic skills. *Nature*, 483(7389):331–335, March 2012.

[4] Mijail D Serruya, Nicholas G Hatsopoulos, Liam Paninski, Matthew R Fellows, and John P Donoghue. Instant neural control of a movement signal. *Nature*, 416(6877):141–142, March 2002. PMID: 11894084.

[5] Dawn M. Taylor, Stephen I. Helms Tillery, and Andrew B. Schwartz. Direct cortical control of 3D neuroprosthetic devices. *Science*, 296(5574):1829 –1832, June 2002.

[6] Jose M Carmena, Mikhail A Lebedev, Roy E Crist, Joseph E O'Doherty, David M Santucci, Dragan F Dimitrov, Parag G Patil, Craig S Henriquez, and Miguel A. L Nicolelis. Learning to control a Brain–Machine interface for reaching and grasping by primates. *PLoS Biology*, 1(2):e42, October 2003.

[7] S. Musallam, B. D. Corneil, B. Greger, H. Scherberger, and R. A. Andersen. Cognitive control signals for neural prosthetics. *Science*, 305(5681):258 –262, July 2004.

[8] Gopal Santhanam, Stephen I. Ryu, Byron M. Yu, Afsheen Afshar, and Krishna V. Shenoy. A high-performance brain–computer interface. *Nature*, 442(7099):195–198, July 2006.

[9] Meel Velliste, Sagi Perel, M. Chance Spalding, Andrew S. Whitford, and Andrew B. Schwartz. Cortical control of a prosthetic arm for self-feeding. *Nature*, 453(7198):1098–1101, June 2008.

[10] Chet T. Moritz, Steve I. Perlmutter, and Eberhard E. Fetz. Direct control of paralysed muscles by cortical neurons. *Nature*, 456(7222):639–642, December 2008.

[11] Beata Jarosiewicz, Steven M Chase, George W Fraser, Meel Velliste, Robert E Kass, and Andrew B Schwartz. Functional network reorganization during learning in a brain-computer interface paradigm. *Proceedings of the National Academy of Sciences of the United States of America*, 105(49):19486–19491, December 2008. PMID: 19047633.

[12] Joseph E. O'Doherty, Mikhail A. Lebedev, Timothy L. Hanson, Nathan A. Fitzsimmons, and Miguel A. L. Nicolelis. A brain-machine interface instructed by direct intracortical microstimulation. *Frontiers in Integrative Neuroscience*, 3, September 2009. PMID: 19750199 PMCID: 2741294.

[13] Karunesh Ganguly and Jose M. Carmena. Emergence of a stable cortical map for neuroprosthetic control. *PLoS Biology*, 7(7):e1000153, July 2009.

[14] Aaron J. Suminski, Dennis C. Tkach, Andrew H. Fagg, and Nicholas G. Hatsopoulos. Incorporating feedback from multiple sensory modalities enhances Brain–Machine interface control. *The Journal of Neuroscience*, 30(50):16777 –16787, December 2010.

[15] C. Ethier, E. R. Oby, M. J. Bauman, and L. E. Miller. Restoration of grasp following paralysis through brain-controlled stimulation of muscles. *Nature*, 485(7398):368–371, May 2012.

[16] Leigh R. Hochberg, Mijail D. Serruya, Gerhard M. Friehs, Jon A. Mukand, Maryam Saleh, Abraham H. Caplan, Almut Branner, David Chen, Richard D. Penn, and John P. Donoghue. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442(7099):164–171, July 2006.

[17] Sung-Phil Kim, John D Simeral, Leigh R Hochberg, John P Donoghue, and Michael J Black. Neural control of computer cursor velocity by decoding motor cortical spiking activity in humans with tetraplegia. *Journal of Neural Engineering*, 5(4):455–476, December 2008.

[18] Leigh R Hochberg, Daniel Bacher, Beata Jarosiewicz, Nicolas Y Masse, John D Simeral, Joern Vogel, Sami Haddadin, Jie Liu, Sydney S Cash, Patrick van der Smagt, and John P Donoghue. Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, 485(7398):372–375, May 2012. PMID: 22596161.

[19] Jennifer L Collinger, Brian Wodlinger, John E Downey, Wei Wang, Elizabeth C Tyler-Kabara, Douglas J Weber, Angus J C McMorland, Meel Velliste, Michael L Boninger, and Andrew B Schwartz. High-performance neuroprosthetic control by an individual with tetraplegia. *Lancet*, 381(9866):557–564, February 2013. PMID: 23253623.

[20] Jose Millan and Jose Carmena. Invasive or noninvasive: Understanding brain-machine interface technology [Conversations in BME. *IEEE Engineering in Medicine and Biology Magazine*, 29(1):16–22, January 2010.

[21] Vikash Gilja, Cindy A Chestek, Ilka Diester, Jaimie M Henderson, Karl Deisseroth, and Krishna V Shenoy. Challenges and opportunities for next-generation intracortically based neural prostheses. *IEEE Transactions on Bio-Medical Engineering*, 58(7):1891–1899, July 2011. PMID: 21257365.

[22] Remy Wahnoun, Jiping He, and Stephen I Helms Tillery. Selection and parameterization of cortical neurons for neuroprosthetic control. *Journal of Neural Engineering*, 3(2):162–171, June 2006. PMID: 16705272.

[23] Karunesh Ganguly and Jose M Carmena. Neural correlates of skill acquisition with a cortical brain-machine interface. *Journal of Motor Behavior*, 42(6):355–360, November 2010. PMID: 21184353.

[24] Shinsuke Koyama, Steven M Chase, Andrew S Whitford, Meel Velliste, Andrew B Schwartz, and Robert E Kass. Comparison of brain-computer interface decoding algorithms in open-loop and closed-loop control. *Journal of Computational Neuroscience*, 29(1-2):73–87, August 2010. PMID: 19904595.

[25] Wei Wu, Michael J Black, Yun Gao, Elie Bienenstock, Mijail Serruya, Ali Shaikhouni, and John P Donoghue. Neural decoding of cursor motion using a kalman filter. *Advances in Neural Information Processing Systems*, 15, 2003.

[26] Wei Wu, Yun Gao, Elie Bienenstock, John P Donoghue, and Michael J Black. Bayesian population decoding of motor cortical activity using a kalman filter. *Neural Computation*, 18(1):80–118, January 2006. PMID: 16354382.

[27] Vikash Gilja, Paul Nuyujukian, Cindy A. Chestek, John P. Cunningham, Byron M. Yu, Joline M. Fan, Mark M. Churchland, Matthew T. Kaufman, Jonathan C. Kao, Stephen I. Ryu, and Krishna V. Shenoy. A high-performance neural prosthesis enabled by control algorithm design. *Nature Neuroscience*, 15(12):1752–1757, 2012.

[28] 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*, PP(99):1, 2012.

[29] Karunesh Ganguly, Lavi Secundo, Gireeja Ranade, Amy Orsborn, Edward F. Chang, Dragan F. Dimitrov, Jonathan D. Wallis, Nicholas M. Barbaro, Robert T. Knight, and Jose M. Carmena. Cortical representation of ipsilateral arm movements in monkey and man. *The Journal of Neuroscience*, 29(41):12948 –12956, October 2009.

[30] Lavi Shpigelman, Hagai Lalazar, and Eilon Vaadia. Kernel-ARMA for hand tracking and brain-machine interfacing during 3D motor control. *Advances in Neural Information Processing Systems*, 21, 2008.

[31] Babak Mahmoudi and Justin C. Sanchez. A symbiotic brain-machine interface through value-based decision making. *PLoS ONE*, 6(3):e14760, March 2011.

[32] Amy L Orsborn, Siddharth Dangi, Helene G Moorman, and Jose M Carmena. Exploring time-scales of closed-loop decoder adaptation in brain-machine interfaces. *Proceedings of the Engineering in Medicine and Biology Society Annual International Conference of the IEEE*, 2011:5436–5439, 2011. PMID: 22255567.

[33] George Paxinos, Xu-Feng Huang, and Arthur W. Toga. *The Rhesus Monkey Brain in Stereotaxic Coordinates*. Academic Press, 1st edition, November 1999.

[34] V. Gilja, P. Nuyujukian, C.A. Chestek, J.P. Cunningham, B.M. Yu, S.I. Ryu, and K.V. Shenoy. High-performance continuous neural cursor control enabled by feedback control perspective. Computational and Systems Neuroscience (COSYNE 2010), 2010.

[35] V. Gilja, P. Nuyujukian, C.A. Chestek, J.P. Cunningham, J.M. Fan, B.M. Yu, S.I. Ryu, and Shenoy K.V. A high-performance continuous cortically-controlled prosthesis enabled by feedback control design. Society for Neuroscience (SFN 2010), 2010.

[36] John P Cunningham, Paul Nuyujukian, Vikash Gilja, Cindy A Chestek, Stephen I Ryu, and Krishna V Shenoy. A closed-loop human simulator for investigating the role of feedback control in brain-machine interfaces. *Journal of Neurophysiology*, 105(4):1932–1949, April 2011. PMID: 20943945.

[37] S. Dangi, S. Gowda, R. Heliot, and J. M Carmena. Adaptive kalman filtering for closed-loop brain-machine interface systems. In *2011 5th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 609–612. IEEE, May 2011.

[38] Dennis Tkach, Jacob Reimer, and Nicholas G. Hatsopoulos. Congruent activity during action and action observation in motor cortex. *The Journal of Neuroscience*, 27(48):13241 –13250, November 2007.

[39] Wilson Truccolo, Gerhard M. Friehs, John P. Donoghue, and Leigh R. Hochberg. Primary motor cortex tuning to intended movement kinematics in humans with tetraplegia. *The Journal of Neuroscience*, 28(5):1163 –1178, January 2008.

[40] Aaron J Suminski, Dennis C Tkach, and Nicholas G Hatsopoulos. Exploiting multiple sensory modalities in brain-machine interfaces. *Neural Networks: The Official Journal of the International Neural Network Society*, 22(9):1224–1234, November 2009. PMID: 19525091.

[41] Benjamin Blankertz, Claudia Sannelli, Sebastian Halder, Eva M Hammer, Andrea Kübler, Klaus-Robert Müller, Gabriel Curio, and Thorsten Dickhaus. Neurophysiological predictor of SMR-based BCI performance. *NeuroImage*, 51(4):1303–1309, July 2010. PMID: 20303409.

[42] Siddharth Dangi, Amy L Orsborn, Helene G Moorman, and Jose M Carmena. Design and analysis of closed-loop decoder adaptation algorithms for brain-machine interfaces. *Neural computation*, 25(7):1693–1731, July 2013. PMID: 23607558.

[43] Zheng Li, Joseph E. O'Doherty, Mikhail A. Lebedev, and Miguel A. L. Nicolelis. Adaptive decoding for brain-machine interfaces through bayesian parameter updates. *Neural Computation*, 23(12):3162–3204, 2011.

[44] Sen Cheng and Philip N Sabes. Modeling sensorimotor learning with linear dynamical systems. *Neural Computation*, 18(4):760–793, April 2006. PMID: 16494690.

[45] Rodolphe Héliot, Karunesh Ganguly, Jessica Jimenez, and Jose M Carmena. Learning in closed-loop brain-machine interfaces: modeling and experimental validation. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics: A Publication of the IEEE Systems, Man, and Cybernetics Society*, 40(5):1387–1397, October 2010. PMID: 20007050.

[46] Rodolphe Heliot, Subramaniam Venkatraman, and Jose M Carmena. Decoder remapping to counteract neuron loss in brain-machine interfaces. *Proceedings of the Engineering in Medicine and Biology Society Annual International Conference of the IEEE*, 2010:1670–1673, 2010. PMID: 21096393.

[47] Karunesh Ganguly, Dragan F. Dimitrov, Jonathan D. Wallis, and Jose M. Carmena. Reversible large-scale modification of cortical networks during neuroprosthetic control. *Nature Neuroscience*, 14(5):662–667, 2011.

[48] Kelvin So, Siddharth Dangi, Amy L Orsborn, Michael C Gastpar, and Jose M Carmena. Subject-specific modulation of local field potential spectral power during brain-machine interface control in primates. *Journal of Neural Engineering*, 2014.

[49] Eberhard E Fetz. Volitional control of neural activity: implications for brain-computer interfaces. *Journal of Physiology*, 579(3):571–579, March 2007. PMID: 17234689.

[50] Jörn Rickert, Simone Cardoso de Oliveira, Eilon Vaadia, Ad Aertsen, Stefan Rotter, and Carsten Mehring. Encoding of movement direction in different frequency ranges of motor cortical local field potentials. *The Journal of Neuroscience*, 25(39):8815 –8824, 2005.

[51] J N Sanes and J P Donoghue. Oscillations in local field potentials of the primate motor cortex during voluntary movement. *Proceedings of the National Academy of Sciences of the United States of America*, 90(10):4470–4474, May 1993. PMID: 8506287 PMCID: PMC46533.

[52] Arjun K Bansal, Carlos E Vargas-Irwin, Wilson Truccolo, and John P Donoghue. Relationships among low-frequency local field potentials, spiking activity, and three-dimensional reach and grasp kinematics in primary motor and ventral premotor cortices. *Journal of Neurophysiology*, 105(4):1603–1619, April 2011. PMID: 21273313.

[53] David A. Markowitz, Yan T. Wong, Charles M. Gray, and Bijan Pesaran. Optimizing the decoding of movement goals from local field potentials in macaque cortex. *Journal of Neuroscience*, 31(50):18412–18422, December 2011.

[54] Robert D Flint, Christian Ethier, Emily R Oby, Lee E Miller, and Marc W Slutzky. Local field potentials allow accurate decoding of muscle activity. *Journal of Neurophysiology*, 108(1):18–24, July 2012. PMID: 22496527.

[55] Robert D Flint, Eric W Lindberg, Luke R Jordan, Lee E Miller, and Marc W Slutzky. Accurate decoding of reaching movements from field potentials in the absence of spikes. *Journal of Neural Engineering*, 9(4):046006, August 2012. PMID: 22733013.

[56] Eun Jung Hwang and Richard A. Andersen. Brain control of movement execution onset using local field potentials in posterior parietal cortex. *Journal of Neuroscience*, 29(45):14363 –14370, November 2009.

[57] Ben Engelhard, Nofar Ozeri, Zvi Israel, Hagai Bergman, and Eilon Vaadia. Inducing gamma oscillations and precise spike synchrony by operant conditioning via brain-machine interface. *Neuron*, 77(2):361–375, January 2013.

[58] Robert D. Flint, Zachary A. Wright, Michael R. Scheid, and Marc W. Slutzky. Long term, stable brain machine interface performance using local field potentials and multiunit spikes. *Journal of Neural Engineering*, 10(5):056005, October 2013.

[59] G. Schalk, J. Kubánek, K. J. Miller, N. R. Anderson, E. C. Leuthardt, J. G. Ojemann, D. Limbrick, D. Moran, L. A. Gerhardt, and J. R. Wolpaw. Decoding two-dimensional movement trajectories using electrocorticographic signals in humans. *Journal of Neural Engineering*, 4(3):264, September 2007.

[60] R.D. Flint, Z.A. Wright, and M.W. Slutzky. Control of a biomimetic brain machine interface with local field potentials: Performance and stability of a static decoder over 200 days. In *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6719 –6722, September 2012.

[61] Simon S Haykin. *Adaptive filter theory.* Prentice Hall, 4th edition, 2002.

[62] Wei Wang, Jennifer L Collinger, Alan D Degenhart, Elizabeth C Tyler-Kabara, Andrew B Schwartz, Daniel W Moran, Douglas J Weber, Brian Wodlinger, Ramana K Vinjamuri, Robin C Ashmore, John W Kelly, and Michael L Boninger. An electrocorticographic brain interface in an individual with tetraplegia. *PloS one*, 8(2):e55344, 2013. PMID: 23405137 PMCID: PMC3566209.

[63] J. P. Donoghue, J. D. Simeral, S.-P. Kim, G. M. Friehs, L. R. Hochberg, and M. J. Black. Toward standardized assessment of pointing devices for brain-computer interfaces. Society for Neuroscience (SfN 2007), 2007.

[64] Adam G. Rouse, Jordan J. Williams, Jesse J. Wheeler, and Daniel W. Moran. Cortical adaptation to a chronic micro-electrocorticographic brain computer interface. *Journal of Neuroscience*, 33(4):1326–1330, January 2013.

[65] Jeremiah D Wander, Timothy Blakely, Kai J Miller, Kurt E Weaver, Lise A Johnson, Jared D Olson, Eberhard E Fetz, Rajesh P N Rao, and Jeffrey G Ojemann. Distributed cortical adaptation during learning of a brain-computer interface task. *Proceedings of the National Academy of Sciences of the United States of America*, 110(26):10818–10823, June 2013. PMID: 23754426.

[66] Yoshinao Kajikawa and Charles E. Schroeder. How local is the local field potential? *Neuron*, 72(5):847–858, December 2011. PMID: 22153379 PMCID: PMC3240862.

[67] N. E. Crone, D. L. Miglioretti, B. Gordon, J. M. Sieracki, M. T. Wilson, S. Uematsu, and R. P. Lesser. Functional mapping of human sensorimotor cortex with electrocorticographic spectral analysis. i. alpha and beta event-related desynchronization. *Brain*, 121(12):2271–2299, December 1998.

[68] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954.

[69] I S MacKenzie. A note on the information-theoretic basis of fitts' law. *Journal of Motor Behavior*, 21(3):323–330, September 1989.

[70] G Schalk, K J Miller, N R Anderson, J A Wilson, M D Smyth, J G Ojemann, D W Moran, J R Wolpaw, and E C Leuthardt. Two-dimensional movement control using electrocorticographic signals in humans. *Journal of Neural Engineering*, 5(1):75–84, March 2008.

[71] Jonathan R. Wolpaw and Dennis J. McFarland. Control of a two-dimensional movement signal by a noninvasive brain-computer interface in humans. *Proceedings of the National Academy of Sciences of the United States of America*, 101(51):17849–17854, December 2004. PMID: 15585584.

[72] Siddharth Dangi, Suraj Gowda, Helene G Moorman, Amy L Orsborn, Kelvin So, Maryam Shanechi, and Jose M Carmena. Continuous closed-loop decoder adaptation with a recursive maximum likelihood algorithm allows for rapid performance acquisition in brain-machine interfaces. *Neural computation*, 26(9):1811–1839, August 2014.

[73] Cynthia A. Chestek, Vikash Gilja, Paul Nuyujukian, Justin D. Foster, Joline M. Fan, Matthew T. Kaufman, Mark M. Churchland, Zuley Rivera-Alvidrez, John P. Cunningham, Stephen I. Ryu, and Krishna V. Shenoy. Long-term stability of neural prosthetic

control signals from silicon cortical arrays in rhesus macaque motor cortex. *Journal of Neural Engineering*, 8(4):045005, August 2011.

[74] Maryam Shanechi and Jose M Carmena. Optimal feedback-controlled point process decoder for adaptation and assisted training in brain-machine interfaces. *2013 6th International IEEE/EMBS Conference on Neural Engineering (NER)*, 2013, 2013.

[75] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing. Second Edition.* 1992.

[76] S-P Kim, J C Sanchez, Y N Rao, D Erdogmus, J M Carmena, M A Lebedev, M A L Nicolelis, and J C Principe. A comparison of optimal MIMO linear and nonlinear models for brain-machine interfaces. *Journal of neural engineering*, 3(2):145–161, June 2006. PMID: 16705271.

[77] Zachary Danziger, Alon Fishbach, and Ferdinando A Mussa-Ivaldi. Learning algorithms for human-machine interfaces. *IEEE transactions on bio-medical engineering*, 56(5):1502–1511, May 2009. PMID: 19203886 PMCID: PMC3286659.

[78] Uri T. Eden, Loren M. Frank, Riccardo Barbieri, Victor Solo, and Emery N. Brown. Dynamic analysis of neural encoding by point process adaptive filtering. *Neural Computation*, 16(5):971–998, May 2004.

[79] Yiwen Wang and Jose C Principe. Tracking the non-stationary neuron tuning by dual kalman filter for brain machine interfaces decoding. *Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 2008:1720–1723, 2008. PMID: 19163011.

[80] Siddharth Dangi, Kelvin So, Amy L Orsborn, Michael C Gastpar, and Jose M Carmena. Brain-machine interface control using broadband spectral power from local field potentials. *Proceedings of the Engineering in Medicine and Biology Society Annual International Conference of the IEEE*, 2013, 2013.