Non-Linear Adaptive Bayesian Filtering for Brain Machine Interfaces

by

Zheng Li

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Craig S. Henriquez, Advisor

_____
Mikhail A. Lebedev

_____
Ronald Parr

_____
Carlo Tomasi

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2010

Abstract

(Computer Science)

Non-Linear Adaptive Bayesian Filtering for Brain Machine Interfaces

by

Zheng Li

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Craig S. Henriquez, Advisor

_____
Mikhail A. Lebedev

_____
Ronald Parr

_____
Carlo Tomasi

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2010

Abstract

Brain-machine interfaces (BMI) are systems which connect brains directly to machines or computers for communication. BMI-controlled prosthetic devices use algorithms to decode neuronal recordings into movement commands. These algorithms operate using models of how recorded neuronal signals relate to desired movements, called models of tuning. Models of tuning have typically been linear in prior work, due to the simplicity and speed of the algorithms used with them. Neuronal tuning has been shown to slowly change over time, but most prior work do not adapt tuning models to these changes. Furthermore, extracellular electrical recordings of neurons' action potentials slowly change over time, impairing the preprocessing step of spike-sorting, during which the neurons responsible for recorded action potentials are identified.

This dissertation presents a non-linear adaptive Bayesian filter and an adaptive spike-sorting method for BMI decoding. The adaptive filter consists of the n-th order unscented Kalman filter and Bayesian regression self-training updates. The unscented Kalman filter estimates desired prosthetic movements using a non-linear model of tuning as its observation model. The model is quadratic with terms for position, velocity, distance from center of workspace, and velocity magnitude. The tuning model relates neuronal activity to movements at multiple time offsets simultaneously, and the movement model of the filter is an order n autoregressive model.

To adapt the tuning model parameters to changes in the brain, Bayesian regression self-training updates are performed periodically. Tuning model parameters are stored

as probability distributions instead of point estimates. Bayesian regression uses the previous model parameters as priors and calculates the posteriors of the regression between filter outputs, which are assumed to be the desired movements, and neuronal recordings. Before each update, filter outputs are smoothed using a Kalman smoother, and tuning model parameters are passed through a transition model describing how parameters change over time. Two variants of Bayesian regression are presented: one uses a joint distribution for the model parameters which allows analytical inference, and the other uses a more flexible factorized distribution that requires approximate inference using variational Bayes.

To adapt spike-sorting parameters to changes in spike waveforms, variational Bayesian Gaussian mixture clustering updates are used to update the waveform clustering used to calculate these parameters. This Bayesian extension of expectation-maximization clustering uses the previous clustering parameters as priors and computes the new parameters as posteriors. The use of priors allows tracking of clustering parameters over time and facilitates fast convergence.

To evaluate the proposed methods, experiments were performed with 3 Rhesus monkeys implanted with micro-wire electrode arrays in arm-related areas of the cortex. Off-line reconstructions and on-line, closed-loop experiments with brain-control show that the n-th order unscented Kalman filter is more accurate than previous linear methods. Closed-loop experiments over 29 days show that Bayesian regression self-training helps maintain control accuracy. Experiments on synthetic data show that Bayesian regression self-training can be applied to other tracking problems with changing observation models. Bayesian clustering updates on synthetic and neuronal data demonstrate tracking of cluster and waveform changes. These results indicate the proposed methods improve the accuracy and robustness of BMIs for prosthetic devices, bringing BMI-controlled prosthetics closer to clinical use.

For my grandparents

Contents

List of Tables

List of Figures

List of Abbreviations and Symbols

Abbreviations

BMI	Brain-machine interface
BR	Bayesian regression
CC	Correlation coefficient (Pearson's r)
EM	Expectation-maximization (algorithm)
GMM	Gaussian mixture model
HMM	Hidden Markov model
KF	Kalman filter
MCMC	Markov Chain Monte Carlo (methods)
MSE	Mean squared-error
OLS	Ordinary least squares
PV	Population vector (method)
RR	Ridge regression
SNR	Signal-to-noise ratio
UKF	Unscented Kalman filter
VB	Variational Bayes
VBR	Variational Bayesian regression
VBGM	Variational Bayesian Gaussian mixture (clustering)
WF	Wiener filter

Symbols

Generally, vectors and matrices are denoted with bold letters and scalars are denoted with italicized letters. A superscript T indicates a transpose. Subscripts in parenthesis index the rows or elements of matrices or indicate the sizes of matrices when $\times$ is present. Symbols used multiple times are listed below:

$\leftarrow$	Assignment operator
$\otimes$	Kronecker product
$\sim$	As a unary operator, indicates the set excluding the following element
$\langle . \rangle$	Expectation of object inside angle brackets
$\lvert . \rvert$	Determinant of the matrix inside the bars
$\mathbf{0}$	Zero matrix
α_k	Hyper-parameter for mixing coefficient of cluster k
β_k	Hyper-parameter for mean of cluster k
$\mathcal{D}$	Dirichlet distribution
D	Number of features in observation (tuning) model
d	Number of dimensions in state or clustering features
$\mathbf{F}$	Transition (movement) model matrix
$\Gamma()$	Gamma function
$\Gamma_N()$	Multivariate Gamma function
$\mathbf{g}()$	Waveform feature generation function
$\mathbf{H}$	Observation (tuning) model coefficient matrix
$\mathbf{h}()$	Observation (tuning) model function
$\mathbf{I}$	Identity matrix
κ	Parameter for UKF which specifies number of copies of prior mean included in sigma points

$\mathbf{K}_t$	Kalman gain at time t
K	Number of clusters or neurons in spike-sorting
KL()	Kullback-Leibler divergence
$\mathcal{L}$	Lower bound in variational Bayes approximation
$\mathbf{\Lambda}, \mathbf{\Lambda}_i, \mathbf{\Lambda}_k$	Hyper-parameter for all tuning coefficients, precision hyper-parameter for tuning coefficients of neuron i, and precision parameter of cluster k, respectively
λ	Ridge regression regularization parameter
ln	Natural logarithm
μ, μ_i	Mean hyper-parameter for all tuning coefficients and mean hyper-parameter for tuning coefficients of neuron i, respectively
$\mathbf{M}_k$	Mean parameter for cluster k
m	Degrees of freedom hyper-parameter for tuning noise covariance matrix
$\mathcal{N}$	(Multivariate) normal distribution
N	Number of neurons
n	Number of taps of kinematics in state of n-th order UKF
$\mathbf{\Psi}$	Inverse scale matrix hyper-parameter for tuning noise covariance matrix
$\psi()$	Digamma function
π_k	Mixing coefficient for cluster k
$\mathbf{P}_t$	Covariance matrix of state (kinematic variables) at time t estimated from all observations up to and including time t.
$\mathbf{P}_{t'}$	Covariance matrix of state (kinematic variables) at time t estimated from all observations up to and including time $t-1$.
$\mathbf{P}_{zz,t}$	Covariance matrix of observations at time t
$\mathbf{P}_{xz,t}$	State-observation cross-covariance matrix at time t
$pos_x(t)$	X-axis position of cursor or hand at time t
$pos_y(t)$	Y-axis position of cursor or hand at time t

$\mathbf{Q}$	Transition (movement) model noise covariance matrix
$\mathbf{R}$	Observation (tuning) model noise covariance matrix
θ_k	Mean hyper-parameter for mean of cluster k
T	Number of data points (time instants)
t	Data or time index
tr()	Trace operator
v_k	Degrees of freedom hyper-parameter for precision of cluster k
vec()	Operator which converts a matrix into a column vector by vertically concatenating the columns
$vel_x(t)$	X-axis velocity of cursor or hand at time t
$vel_y(t)$	Y-axis velocity of cursor or hand at time t
$\mathcal{W}$	Wishart distribution
$\mathcal{W}^{-1}$	Inverse-Wishart distribution
$\mathbf{W}_k$	Scale matrix hyper-parameter for precision of cluster k
w_i	Weight for sigma point i
$\mathcal{X}_i$	Sigma point i calculated from predicted state
$\mathbf{X}$	Matrix of kinematic variables (features) for a block of data
$\hat{\mathbf{x}}_t$	Mean vector of state (kinematic variables) at time t estimated from all observations up to and including time t.
$\hat{\mathbf{x}}_{t'}$	Mean vector of state (kinematic variables) at time t estimated from all observations up to and including time $t-1$.
$\mathbf{Y}$	Matrix of binned spike counts for a block of data
$\mathbf{y}_t$	Vector of observations (binned spike counts) at time t
$\mathcal{Z}_i$	Sigma point i of predicted observation
$\mathbf{z}_t$	Vector of predicted observations at time t

Acknowledgements

I thank my advisor, Craig Henriquez, and committee members, Mikhail Lebedev, Ron Parr, and Carlo Tomasi, for their intellectual support, encouragement, and advice. I also express my deep gratitude to Miguel Nicolelis for the opportunity to conduct research in the Nicolelis Lab, without which none of this would be possible. I am also indebted to the many professors and teachers who have shaped me into the person I am.

I thank my colleagues and fellow students for their assistance during my graduate studies: Joey O'Doherty, Tim Hanson, Nathan Fitzsimmons, Ben Grant, Chris La Pilla, Josh Robinson, Qi An, and Christopher Painter-Wakefield. I also thank the students and staff of the Nicolelis Lab for their assistance.

My work was funded by the Duke University Graduate School, the Defense Advanced Research Projects Agency, and the Telemedicine & Advanced Technology Research Center of the US Army, and I thank them for their generous financial support.

Lastly, I thank my family for their love and support, and I thank my wonderful wife, Chengting, for her love, companionship, and encouragement.

Chapter 1

Introduction

This dissertation aims to provide better algorithms for the computational problem of interpreting recordings from the brain for brain-control of machines. This process, called *decoding*, is a core component of brain-machine interfaces — systems that link brains to machines directly. This dissertation presents algorithms which enable use of more accurate models of brain activity and facilitate adaptation to changes in the brain over time. These features enable increased accuracy and robustness for brain-machine interfaces.

This chapter presents background on brain-machine interfaces, brain-controlled prosthetics, and the computational problems involved in their implementation. Then the contributions of this dissertation are summarized. Finally, a description of the remaining chapters is given.

1.1 Brain-Machine Interfaces

Brain-machine interfaces (BMI), also called brain-computer interfaces or brain-controlled interfaces (BCI), are systems which enable the brain to communicate directly with machines, computers, sensors, and actuators. This direct communication may be facilitated through many technologies. For the brain-to-machine direction

of communication, current and speculative methods include single and population extracellular electrode recordings, local-field potential recordings, electrocorticography, electroencephalography, functional magnetic resonance imaging, magnetoencephalography, near infra-red spectroscopy, and optical imaging. For the machine-to-brain direction of communication, current and speculative methods include extracellular electrical micro-stimulation, transcranial magnetic stimulation, and channel-rhodopsin optical stimulation. Each technological paradigm offers advantages and disadvantages in areas such as spatial resolution, temporal resolution, invasiveness, robustness, portability, and cost.

Extracellular Recordings

This dissertation focuses on the paradigm of extracellular recordings of populations of cortical neurons, though the algorithms and models described may be applicable to other paradigms with some modifications. In this paradigm, arrays of microelectrodes are implanted just below the surface of the neocortex. The electrodes typically penetrate a few millimeters deep so that their tips are close to output-layer neurons. Voltage recordings made at the tips, and possibly other recording sites, are amplified and filtered. Then, signal processing methods are used to detect *action potentials*, the basic units of communication between neurons. Action potentials are often called *spikes* due to their shape in a voltage versus time graph. The frequency and timing of spikes characterize the communication between neurons. To make sense of the activity in the brain, computational models are designed which relate spikes from recorded neurons to behavior or desired actions.

Spike-Sorting

An important preprocessing step for extracellular recordings is spike-sorting, the identification of neurons responsible for action potentials recorded by an electrode.

This process is necessary because the electrical activity of more than one neuron is usually recorded on an electrode. Spikes of different neurons are distinguishable because different neurons tend to have different *waveforms*, the voltage versus time curves of extracellular recordings of action potentials. Spike-sorting is typically performed by matching the shapes of spike waveforms with waveform templates for each neuron. Templates are set at the beginning of each experiment by human experts or through unsupervised clustering of waveform shapes, using the assumption that waveforms in each cluster belong to the same neuron. Besides template matching, other categorization algorithms may be used, but the clustering process or labeling by expert is still necessary to label training data.

1.1.1 BMI-Controlled Prosthetics

Brain-machine interfaces, once mature and wide-spread, enable a myriad of potential applications, as they can improve every interaction between humans and their tools. One important short-term application for which BMI is well-suited is control of prosthetic devices for people with paralysis from upper spinal-cord injury. For such injuries, the movement control signals sent by the brain are interrupted in the injured spinal-cord. Thus, less invasive prosthetics controlled by peripheral nerves are not applicable.

Example Devices

An example prosthetic device is a robotic arm controlled using neuronal recordings from the arm- and hand-related areas of the cortex. Such a prosthetic would offer increased independence to people with paralysis. The BMI may also be used to control a robotic wheelchair or direct a cursor on a computer, possibly to control other devices. This dissertation will focus on the control of an arm prosthetic or computer cursor from cortical neuronal recordings. The methods described in this dissertation are

also applicable to controlling other devices which take continuous-valued control commands.

1.2 Decoding Movement Commands

Estimating desired movement commands from neuronal recordings is often called decoding since the activity of certain neurons in the brain are said to *encode* the desired movements of limbs in their firing rate or timing. Decoding can output either discrete-valued commands or continuous-valued commands, and the two paradigms typically involve different models and algorithms. This dissertation will focus on estimating continuous-valued commands.

Commands

Movement commands for prosthetic arms may be in the form of joint angles or end-effector positions and their time derivatives. Alternatively, commands may be in terms of forces or other parameters and used with a physical model of the prosthetic arm [57, 105]. This dissertation will focus on decoding movement commands in terms of desired end-effector positions. The methods presented should also be applicable to decoding of angles, forces, or other parameters with little modification.

Time and Neuronal Activity Features

The decoding of movement commands occurs sequentially in time. Typical approaches discretize time into intervals ranging from 100 ms to 1 ms. Implants record neuronal activity, which is preprocessed and passed as features to the decoder at each time step. The decoder presented in this dissertation will use spike counts for each neuron during the time step as an estimate of the instantaneous firing rate. This procedure is called *binning*. Other BMI paradigms use other features, such as the spectral power of local field potential recordings in different frequency bands or the presence of spikes

in 1 to 5 ms bins.

1.2.1 Notation

Neuronal recordings at time t make up a vector $\mathbf{y}(t)$. The vector encompasses the entire population of neurons with one (or more) numbers per neuron. The recordings are used to infer values of interest at time t, which for this dissertation will be the x- and y-axis coordinates of a cursor or robotic arm end-effector. The values of interest are represented by a vector $\mathbf{x}(t)$. The decoder is a function which maps $\mathbf{y}(t)$, or the pair $\{\mathbf{y}(t), \mathbf{x}(t-1)\}$, to $\mathbf{x}(t)$. The former is used by direct methods, such as the population vector method [37] and the Wiener filter [11, 42, 44, 104, 139]. The latter is used by decoders which have temporal models of dynamics, such as Bayesian filters.

Parameter Fitting

Decoders have parameters which are fitted to the recorded neurons, since the relationship between $\mathbf{y}$ and $\mathbf{x}$ is not known a priori. This relationship is called neuronal *tuning* or *modulation* to movement. Typically, some form of regression is used to fit the parameters of a tuning model. Some decoders, such as Bayesian filters, are parameterized by two models: one describes the tuning and a second describes the relationship between $\mathbf{x}(t-1)$ and $\mathbf{x}(t)$. These models are discussed in more detail in Chapter 2.

1.3 Adapting to Changes in Waveforms and Tuning

Neuron Identity

Early demonstrations of brain-control of cursors and robotic arms used decoder parameters which were fit at the beginning of each experimental session and held fixed for the remainder of the session [11, 104, 139]. These parameters were discarded

after the experiment, and new parameters were calculated in subsequent experiments. The use of this paradigm is due to concern that the neurons recorded during one session may not be the same as the neurons recorded in the next. Micro-movement of the recording electrodes with respect to neurons, immune system responses, and electrical noise may change the waveforms of neurons or cause them to appear or disappear from recordings.

These changes impair spike-sorting and cause neuron loss, discovery, and mistakes in inferred identity. Thus, in most prior work, the instability of neuronal recordings forced daily adjustment of spike-sorting templates, which make the decoder parameters inapplicable to subsequent sessions. Researchers understood that a BMI for a clinical prosthesis must overcome these problems, as daily adjustment of spike-sorting templates and re-fitting of parameters is impractical. Efforts have been made to identify and characterize stable neurons over longer durations [12, 20, 31, 76]. However, discarding neurons with unstable waveforms is sub-optimal, and methods which update spike-sorting templates automatically are needed.

Adaptive Decoders

Even with reliable neuron identification, neurons' tuning characteristics slowly change over time due to many factors [12, 31, 63, 68, 87, 96, 120, 125]. Thus, decoders must be able to adapt to these changes to operate for long durations without requiring re-fitting of parameters.

1.4 Contributions

This dissertation presents a non-linear adaptive Bayesian filter which improves upon existing decoding methods in accuracy and robustness to tuning change. This filter is composed of two components: the n-th order unscented Kalman filter [73] and the Bayesian regression self-training update method. This dissertation also presents a

method for updating spike-sorting parameters using variational Bayesian Gaussian mixture clustering.

Non-linear Decoding

The n-th order unscented Kalman filter allows non-linear models of neuronal tuning to be used in a computationally light manner. A novel non-linear model, the quadratic model of tuning, was developed to take advantage of the filter. The filter has the advantage of modeling the tuning of neurons to movement at multiple time offsets simultaneously, and more complex patterns of movements can be captured using the autoregressive order n movement model of the filter.

The improved performance of the n-th order unscented Kalman filter is demonstrated in off-line reconstructions and closed-loop brain-control experiments with 2 Rhesus monkeys.

Adaptive Decoding

The novel Bayesian regression self-training update method allows the tuning model of the n-th order unscented Kalman filter to adapt to changes in neural tuning. The method uses a Bayesian linear regression to periodically update neural tuning model parameters using the outputs of the filter, which are assumed to be the desired movements. Uncertainty in the model parameters are tracked over time so that proper trade-offs between prior knowledge of the parameters and new information from updates are calculated automatically. Two Bayesian regression variants are presented. The first is a fast, analytical method which uses a joint probability distribution to represent the tuning model parameters. The second uses a factorized distribution to represent the parameters, which allows subsets of neurons to be updated and new neurons to be added. However, computing the Bayesian regression solution on this factorized representation requires an iterative approximation method

called variational Bayes.

The ability of the Bayesian regression self-training method to help maintain control accuracy is demonstrated in off-line reconstructions of data from 3 monkeys and closed-loop brain-control with 1 monkey over a span of 29 days. Experiments on synthetic data show the method can be applied to other tracking problems where observation models change over time.

Updating Spike-Sorting

Variational Bayesian Gaussian mixture clustering allows the clustering of spike waveforms to be updated in a Bayesian and computationally-efficient manner. During updates, Bayesian clustering of spike waveforms is performed using priors on cluster locations, shapes, and mixing probabilities. These priors are the previous clustering results, which allows tracking of clusters over time and speeds up convergence of the procedure.

Clustering updates are tested on synthetic data, and spike-sorting updates are demonstrated on neuronal data recorded from a monkey.

Together, these methods make up a non-linear, adaptive brain-machine interface for prosthetic control which offers improved decoding accuracy and robustness to changes in neuronal tuning and recording, bringing closer the clinical use of brain-controlled prosthetic devices.

1.5 Summary of Remaining Chapters

Chapter 2 first reviews prior work in decoding methods and models of neuronal tuning. Then, the Kalman filter for BMI decoding is reviewed. Next, the quadratic model of tuning is presented. This is followed by a review of the unscented transform and a description of the n-th order unscented Kalman filter. Lastly, model fitting

procedures are described.

Chapter 3 first reviews prior studies of changes in neuronal tuning over time and prior attempts at building adaptive decoders. Also, related work on adaptive filtering from the signal-processing literature is briefly reviewed. Then, the issues of identifiability, divergence, and alertness are discussed. This is followed by a description of the Bayesian regression self-training method and the two Bayesian regression variants. Lastly, model fitting using Bayesian regression and the transition model for tuning parameters are described.

Chapter 4 presents two sets of experiments. The first set demonstrates the improved accuracy of the n-th order unscented Kalman filter compared to previous linear methods. The second set demonstrates the effectiveness of the Bayesian regression self-training method at maintaining the accuracy of the n-th order unscented Kalman filter. Results from both off-line and closed-loop experiments are presented in each set, and additional experiments testing Bayesian regression self-training on synthetic and modified neuronal data are presented. Also, Chapter 4 describes primate experimental methods, including descriptions of implants, behavioral tasks, neuronal data preprocessing, and analysis procedures.

Chapter 5 first reviews prior work in updating waveform clustering for spike-sorting. Then the variational Bayesian Gaussian mixture clustering update method is presented. Lastly, the method is tested to synthetic and neuronal data.

Chapter 6 presents discussion of the methods proposed in this dissertation. Extensions and future work are then proposed. Finally, a summary is given.

Chapter 2

Non-Linear Decoding

Linear filtering methods are commonly used for BMI decoding in prior work. However, the relationship between neuronal activity and movement can be better described using a non-linear relationship (see Figure 4.4 on page 91 for an illustration). Filtering with a non-linear model requires more sophisticated methods than filtering with a linear model. This chapter proposes a computationally fast non-linear filter, the n-th order unscented Kalman filter (UKF), and a quadratic model of tuning for BMI decoding.

In this chapter, prior work on models of neuronal tuning to arm movements is first reviewed. Next, prior work on decoding algorithms is reviewed. This is followed by a description of the Kalman filter for decoding. Then, the quadratic model of tuning is presented, and a description of the n-th order unscented Kalman filter is given. Finally, procedures for model fitting are described.

2.1 Prior Work

Since a discussion of decoding algorithms cannot be completely separate from a discussion of the models of neuronal tuning, there will be overlap and repeated mention of studies in the following two subsections.

2.1.1 Models of Neuronal Tuning

Cosine Tuning

In their seminal work, Georgopoulos et al. [36] proposed a cosine model of tuning to explain neuronal modulations during center-out reaching movements made by monkeys. This model described the normalized instantaneous firing rate of a modulated neuron as a cosine function of the difference in angle between the hand movement direction, $d(t)$, and the neuron's preferred direction:

$$y(t) = \cos(d(t) - \text{PD}) \tag{2.1}$$

The preferred direction, PD, is a property of each modulated neuron which can be found by regression.

The cosine tuning model can be generalized from modeling direction to modeling the velocity vector of movement by replacing the cosine term with linear coefficients for the Cartesian coordinates of the velocity:

$$y(t) = b_1 vel_x(t) + b_2 vel_y(t) \tag{2.2}$$

$vel_x(t)$ and $vel_y(t)$ are the x- and y-axis velocities of the hand at time t, and b_1 and b_2 are parameters for the neuron. This linear model will be called the *linear velocity model* in the sequel. The linear velocity model is widely used due to its simplicity [11, 44, 104, 120, 147], despite ignoring neuronal tuning to other kinematic variables.

Tuning to Speed

A few studies have investigated neuronal modulation to the speed of the movement or the magnitude of the velocity vector. Moran and Schwartz [79] proposed a model similar to the linear velocity model with the addition of a coefficient for the velocity magnitude. This velocity magnitude term makes the model quadratic in Cartesian

coordinates. Brockwell et al. [9, 10] used a generalized linear model (GLM) with a logarithmic link function and the Moran and Schwartz model as the linear portion.

Truccolo et al. [125] investigated the fit of GLMs with logarithmic link functions to cortical data collected from humans. They investigated GLMs with 3 different linear portions: the cosine tuning model, the linear velocity model, and a model with a velocity magnitude term only. Their analysis showed that the linear velocity model had the lowest deviance among the 3 models.

Tuning to Distance

A few studies have investigated tuning to the distance of an arm reaching motion. Fu et al. [29] investigated the time-course of tuning during reaches and used a model which included terms for the direction of the reach, distance of the reach, distance squared, and position offset of the reach. Fu et al. found that tuning to distance of the reach occurred last in the time course of the reaching movement.

Turner and Anderson [127] investigated cosine tuning and tuning to distance of reaches in globus pallidus neurons. They found significant tuning to the distance of reaches in the majority of neurons recorded. Cowan and Taylor [15] reported that about 10 to 20% of neurons were tuned to distance to target in a 3D center-out task, while up to 85% were tuned to movement direction and movement vector.

Other Models of Tuning

Many studies have investigated more sophisticated models alongside more sophisticated decoding algorithms. Lin et al. [75] used a self-organizing feature map to construct a non-parametric model. Isaacs et al. [47] used a k-nearest-neighbors non-parametric model on neuronal data with dimensionality reduced through principal components analysis. Several studies have investigated modeling with artificial neural networks [60, 99, 139], which can approximate non-linear functions. Shpigelman et

al. [108–110] proposed a kernelized model with a custom-designed kernel function called the *Spikernel*. Gao et al. [33] modeled tuning with generalized linear models and generalized additive models. Their generalized linear models used logarithmic link functions and linear position, velocity, and acceleration terms; their generalized additive models used 4th degree spline features. Wu et al. [151] decoded with a mixture of linear models and a switching Kalman filter. Navot et al. [83] used a non-parametric k-nearest-neighbors decoder and a novel feature selection method. Shoham et al. [106] proposed a grid-based model with a novel *normalized-Gaussian* error distribution. Wood et al. [146] modeled population spiking activity using Gibbs distributions. Truccolo and Donoghue [123] used a regression-tree based non-parametric model fitted with stochastic gradient boosting regression. Fraser et al. [28] modeled tuning with splines and decoded with a particle filter. They showed their method could decode without spike-sorting. Truccolo et al. [126] proposed a GLM with terms for the population's recent spiking history and showed that this model could predict single spikes with high accuracy.

2.1.2 Algorithms

Much work has been done in search of accurate and efficient decoders for brain-machine interfaces. Bashashati et al. [5] provides a comprehensive survey of work prior to 2007, and Koyama et al. [65] presents a systematic comparison of several common approaches.

The first decoder was the population vector method developed by Georgopoulos et al. [37] based on the cosine tuning model. This method calculates movement velocity as the sum of preferred direction vectors scaled by the neurons' normalized firing rates [37]. Many BMI studies have used this algorithm, or variants of it, to decode movement [55, 56, 120, 134, 135].

Wiener Filter

The Wiener filter [43], a well-known linear filter, improves upon the population vector approach. The Wiener filter is an optimal linear method and takes into account temporal offsets and autocorrelations in tuning using its time tap structure. The Wiener filter has been used in many studies [11, 42, 44, 104, 139] and is popular in BMI research because of its relative simplicity and efficacy. It remains the standard by which other algorithms are compared.

Kalman Filter

Research on BMI decoding later turned towards Bayesian state-space methods, particularly, the Kalman filter [8, 72, 147–149, 151]. The Bayesian state-space or hidden Markov model (HMM) approach underlying the Kalman filter explicitly separates the models of how neuronal activity relates to movements and how these movements evolve over time. The Kalman filter also provides confidence estimates of its output, though for a stationary model the confidence estimates are determined entirely by the parameter fit and time since initialization, and do not reflect on-going activity.

Non-linear Filters

Non-linear models of neuronal tuning provide better descriptions of neuronal modulations, but are more computationally demanding to use and require greater vigilance against over-fitting. The switching Kalman filter, in which several Kalman filters operate in parallel using different linear models, was a non-linear method applied to decoding by Wu et al. [148].

Another non-linear Bayesian filtering approach, the particle filter (also called sequential Monte Carlo or condensation) has been used for decoding in several studies. As the particle filter is based on sampling, given enough samples or particles, it

can estimate using any non-linear model. However, due to the heavy computational burden of simulating many particles, demonstrations of real-time brain-control using particle filters have been rare. Examples of off-line application of the particle filter can be found in the studies by Gao et al. [32, 33], Brockwell et al. [9, 10], Shoham et al. [106], and Wang et al. [137, 138]. Fraser et al. [28] tested in real-time a particle filter which used a spline tuning function.

Point-Process Filters

Another class of decoding methods estimate movement directly from individual spike arrivals instead of binned spike counts. In this approach, spike trains are modeled as discrete events or point-processes and decoding can operate at millisecond time scales. The point process analog of the Kalman filter, called the stochastic-state point-process filter (SSPPF), was proposed by Eden et al. [24, 25]. The SSPPF uses a Gaussian representation for uncertainty in state estimates and an inhomogeneous Poisson model of spiking. Barbieri et al. [4] estimated the location of a foraging rat using recordings from CA1 hippocampal neurons and the SSPPF. Truccolo et al. [124, 125] analyzed and compared the ability of the SSPPF to estimate several behavioral variables in simulations, data collected from monkeys, and data collected from humans. Brockwell et al. [9,10] and Wang et al. [137,138] proposed point-process particle filters.

Including Targets in Inference

To improve decoding of simple reaching movements, tuning to the goal coordinates of reach trajectories has been used to supplement tuning to movement. Kemere et al. [54] included both movement tuning and target position tuning in a maximum likelihood filter. Srinivasan et al. [113, 114] incorporated estimated target position in both the Kalman and point-process filter frameworks. Later, Srinivasan et al. [112]

combined tuning to target position, point-process inputs, and continuous-value inputs to allow spikes and other neural measurements to be used in a single Bayesian filter. Mulliken et al. [80] included the target location in the state of a Kalman filter for prediction from posterior parietal cortex.

Other Methods

Other techniques have been investigated for decoding of continuous movements. Isaacs et al. [47] used principal components analysis and the k-nearest-neighbor algorithm. Kim et al. [60] proposed a competitive mixture of linear filters. Darmanjian et al. [17] first classified whether the user desired movement or no movement with a hidden Markov model and then predicted movement using a moving average model. Wessberg et al. [139], Sanchez et al. [98–100], Hatsopoulos et al. [42], and Choi et al. [13] used various artificial neural network decoders. Shpigelman et al. [108–110] decoded with support vector regression and a custom-built kernel called the Spikernel. Fisher and Black [27] used an autoregressive moving average (ARMA) approach, and Shpigelman et al. [107] demonstrated kernel autoregressive moving average (KARMA) decoding in closed-loop BMI. Shakhnarovich et al. [105] developed a spring-based model for a prosthetic arm and decoded the spring coefficients that determined motion using a Wiener filter and support vector regression. Kim et al. [57] decoded muscle activation with a Wiener filter and an artificial neural-network and used a musculoskeletal model of the arm to produce movements. Ventura [131] proposed a decoding paradigm which avoids spike-sorting by replacing sorted spike counts with conditional expectations of spike counts given unsorted spikes.

Other Work Related to Decoding

A variety of techniques have been employed for decoding discretized action choices [21, 42, 45, 46, 62, 81, 86, 101, 102].

Studies have applied various machine learning techniques to improve parameter fitting for decoders. Kim et al. [58] applied least absolute shrinkage and selection operator (LASSO) regression and the wavelet transform to fit a Wiener filter. Kim et al. [59] used nonnegative matrix factorization to fit a mixture of Wiener filters. Ting et al. [121, 122] developed the variational Bayesian least squares method for fitting a Wiener filter with a sparse set of coefficients. Santhanam et al. [102] presented a Poisson noise model factor analysis method for fitting a classification decoder.

2.2 Review of the Kalman Filter

Bayesian Approach

The Bayesian approach to decoding, as used in e.g. the Kalman filter [8, 72, 147–149, 151], stochastic state point-process filter [4, 24, 25, 112, 124, 125], and particle filter [9, 10, 28, 32, 33, 106, 137, 138], has several advantages. The principled handling of uncertainty provides theoretical assurances of optimality if the modeling assumptions are valid. Explicitly keeping track of uncertainty allows failure and abnormality detection and diagnosis. The common language of probability allows modularity of components (as seen in [112]). Among Bayesian approaches to decoding, the Kalman filter offers particular characteristics which make it well-suited for BMIs.

Kalman Filter Background

The Kalman filter, also called the Stratonovich-Kalman-Bucy filter [52, 53, 115, 116] is a well-known optimal sequential Bayesian estimator. It is optimal for inference on continuous-variable hidden Markov models (HMMs) that are linear and have normally-distributed noise, called linear dynamical systems. In HMMs, the variables in the system are separated into hidden variables, called *states*, and observable variables, called *observations*. Time is often discretized into fixed-duration steps, though the Kalman-Bucy variant of the filter operates on continuous time. Two models describe

the HMM: the *transition model* and the *observation model*. The transition model describes how states evolve over time, and the observation model describes how states relate to observations. In linear dynamical systems, these models are linear and Gaussian, i.e. they can be described by linear functions corrupted with noise which is drawn i.i.d. from a zero-mean normal distribution. Due to these assumptions, the uncertainty in the state estimate can be exactly described using a normal distribution, and the state of the system can be estimated in closed form with sequential updates.

If the linear and Gaussian assumptions are valid for the system of interest, the estimates produced by the Kalman filter are optimal. Furthermore, given only the linear assumption, the Kalman filter is still optimal in terms of the quadratic loss function [52]. The Kalman filter is often sufficiently accurate for many applications. It is also simple to implement and computationally fast. These qualities make the Kalman filter very practical, and it is widely used in many fields.

Comparison to Other Filters

When compared to the particle filter, the Kalman filter is far faster in computation, at the cost of the linear and Gaussian restrictions. The linear restriction can be eased using an approximation method called the unscented transform, as discussed in the next section.

Point-process filters [4,9,10,24,25,112,124,125,137,138] model probability of spiking using a conditional intensity function, which is a generalization of the inhomogeneous Poisson rate function [24]. While spiking activity of neurons is generally thought of as inhomogeneous Poisson in distribution when spikes are counted in bins, the refractory period of neurons makes this an approximation. The normal noise assumption of the Kalman filtering approach, while inaccurate at small bin sizes, is reasonable for larger bin sizes, when the larger mean makes the Poisson distribution of spike counts closer to a normal distribution. Furthermore, the multivariate normal distribution for the

tuning model noise allows correlated noise to be easily handled by the Kalman filter, while extensions to the point-process filters are required to handle correlated noise. Even with the normal noise assumption invalid, the Kalman filter remains optimal in terms of quadratic loss if the model is linear.

State and Observations for BMI Decoding

For the BMI decoding application, one way to model the system is the following: the hidden state is the desired movement (e.g., the hand or cursor position and velocity) and the observations are measurements of neuronal activity (e.g. the binned spike count). An illustration of this HMM is shown in panel A of Figure 2.1. This approach to modeling has been shown to work well in several studies [8, 72, 147–149, 151] and is used in this work.

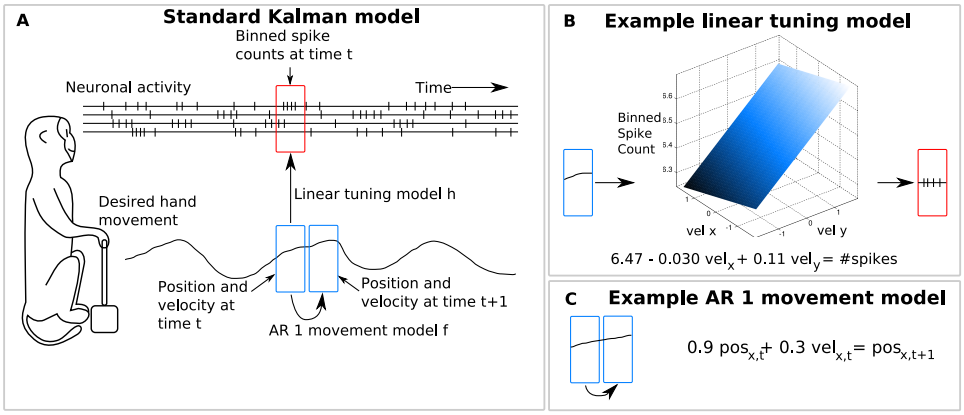

Figure 2.1: HMM for Kalman filter decoding. A: AR 1 movement model predicts future position and velocity from current position and velocity. Linear tuning model relates current position and velocity with binned spike counts. B: Example simplified linear tuning model with only velocity terms. C: Example AR 1 movement model for x-axis position.

An alternative approach to modeling the system is to let neuronal activity be control input, let movements be observations, and leave the state as an unknown latent variable to be inferred during model fitting with expectation-maximization [1, 129].

Using the hidden state as desired movements approach, the kinematic variables describing movement at time t are the state variables: $\mathbf{x}(t)$:

$$\mathbf{x}(t) = \begin{bmatrix} pos_x(t) \\ pos_y(t) \\ vel_x(t) \\ vel_y(t) \end{bmatrix} \tag{2.3}$$

$pos_x(t)$ and $pos_y(t)$ are the x- and y-axis coordinates of the hand or cursor at time t, and $vel_x(t)$ and $vel_y(t)$ are the x- and y-axis velocities at time t. Neuronal firing rates at time t are represented by a column vector $\mathbf{y}(t)$. In this work, the firing rates are estimated by binning spikes with non-overlapping, 100 ms wide bins.

Transition Model

The linear transition model $\mathbf{f}$ predicts the state at the current time step given the state at the previous time step:

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t-1)) + \mathbf{w}(t-1) \tag{2.4}$$

$\mathbf{x}(t)$ and $\mathbf{x}(t-1)$ are the state random variables at time steps t and $t-1$, respectively. $\mathbf{w}(t-1)$ is zero-mean, normally-distributed *transition model noise*. This noise describes the uncertainty arising from approximations made in the transition model and intrinsic randomness in the process. In the BMI decoder application, the transition model describes the movement of the hand or cursor and thus is called the *movement model*. An example portion of a movement model is shown in panel C of Figure 2.1 on the preceding page. The transition model for the state variables described in Equation 2.3 is an autoregressive order 1 model, because the current state only depends on one previous state.

Observation Model

The linear observation model $\mathbf{h}$ relates the observations (i.e., binned spike counts), $\mathbf{y}(t)$, to the state, $\mathbf{x}(t)$:

$$\mathbf{y}(t) = \mathbf{h}(\mathbf{x}(t)) + \mathbf{v}(t) \tag{2.5}$$

$\mathbf{y}(t)$ is the column vector containing the observations (binned firing rates) at time t. $\mathbf{v}(t)$ is zero-mean, normally-distributed noise, called the *observation model noise*. It describes the uncertainty in the observation model arising from inherent randomness, sensor noise, or approximations taken in the model. In the BMI decoder application, the neuronal tuning model is the observation model. The following linear tuning model is used:

$$y(t) = b_1 pos_x(t) + b_2 pos_y(t) + b_3 vel_x(t) + b_4 vel_y(t) \tag{2.6}$$

$y(t)$ is the mean-subtracted single-neuron firing rate at time t, $pos_x(t)$ and $pos_y(t)$ are the x- and y-axis coordinates of the cursor or hand at time t , $vel_x(t)$ and $vel_y(t)$ are the x- and y-axis velocities of the cursor at time t, and $b_1, b_2, ..., b_4$ are scalar parameters, 4 per neuron. An illustration of a simplified version of this model with only velocity coefficients is shown in panel B of Figure 2.1 on page 19.

State Uncertainty Representation

The hidden state $\mathbf{x}$ is a random vector with uncertainty. This uncertainty is modeled probabilistically using a multivariate normal distribution. The normal distribution representation is exact because of the linear and Gaussian modeling assumptions. The distribution is parameterized by the estimated mean vector $\hat{\mathbf{x}}$ and the estimated covariance matrix $\mathbf{P}$. The values are estimated, since the hidden state is not directly observable during filtering.

Filter Iterations

The Kalman filter is implemented in a series of equations which operate on the estimated state mean vector and covariance matrix repeatedly. Each iteration corresponds to a step in time. In each iteration, the transition model *predicts* the state at the current time step using the state at the previous time step. This is *dead-reckoning* based solely on the previous state. Then the observations *update* this prediction through the observation model.

Predict Step

The predict step consists of two operations. The first predicts the state mean vector at time t:

$$\hat{\mathbf{x}}_{t'} = \mathbf{F}\hat{\mathbf{x}}_{t-1} \tag{2.7}$$

$\hat{\mathbf{x}}_{t'}$ is the predicted state mean vector at time t, and $\hat{\mathbf{x}}_{t-1}$ is the state mean vector at time $t-1$. The matrix $\mathbf{F}$ implements the linear transition model $\mathbf{f}$. $\mathbf{F}$ is a square d by d matrix, where d is the number of state variables. For the state described in Equation 2.3, $d = 4$.

In the Kalman filtering literature, time indexing is often denoted with conditional bars, such as $\hat{\mathbf{x}}_{t|t}$ to mean the estimated state at time t given all observations up to and including time t and $\hat{\mathbf{x}}_{t|t-1}$ to mean the estimated state at time t given all observations up to and including time $t-1$. This notation makes explicit the observations included in the state estimates during each point of filter operation. For notational simplicity, the exposition here uses $\hat{\mathbf{x}}_t$ to denote $\hat{\mathbf{x}}_{t|t}$ and $\hat{\mathbf{x}}_{t'}$ to denote $\hat{\mathbf{x}}_{t|t-1}$, and likewise for $\mathbf{P}$.

The second operation in the predict step predicts the state covariance at time t:

$$\mathbf{P}_{t'} = \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q} \tag{2.8}$$

$\mathbf{P}_{t-1}$ is the state covariance at time $t-1$, and $\mathbf{P}_{t'}$ is the predicted covariance at time t. $\mathbf{Q}$ is the covariance matrix of $\mathbf{w}(t)$, the zero-mean, normally-distributed

transition model noise. Pre- and post-multiplying the state covariance $\mathbf{P}_{t-1}$ by the matrix $\mathbf{F}$ modifies the uncertainty in the state estimate according to the transition model. Adding the d by d matrix $\mathbf{Q}$ increases the state covariance to account for the uncertainty introduced by the dead-reckoning.

Update Step

In the update step, the model of neuronal tuning is applied to the predicted state (hand or cursor position and velocity) to generate predicted observations (binned spike counts for each neuron):

$$\mathbf{z}_t = \mathbf{H}\hat{\mathbf{x}}_{t'} \tag{2.9}$$

$\mathbf{z}_t$ is the vector of predicted observations, $\hat{\mathbf{x}}_{t'}$ is obtained from Equation 2.7, and $\mathbf{H}$ is a matrix that implements the linear observation (neuronal tuning) model $\mathbf{h}$. $\mathbf{H}$ is a N by d matrix, where N is the number of observations (neurons).

The difference between the predicted observations $\mathbf{z}_t$ and the actual observations $\mathbf{y}_t$ (the same as $\mathbf{y}(t)$ in the model specification) is used to update the predicted state $\hat{\mathbf{x}}_{t'}$ to better correspond to the observations. To calculate the size of this update, the matrix $\mathbf{S}_t$, an N by N covariance matrix of the predicted observations, is first calculated:

$$\mathbf{S}_t = \mathbf{H}\mathbf{P}_{t'}\mathbf{H}^T + \mathbf{R} \tag{2.10}$$

Matrix $\mathbf{R}$ is the N by N covariance matrix for $\mathbf{v}(t)$, the zero-mean, normally-distributed observation model noise.

The magnitude of the update depends on the uncertainty of the current state estimate and the uncertainty in the predicted observations. For example, if the state estimate has low uncertainty (small $\mathbf{P}_{t'}$) and the predicted observations have high uncertainty (large $\mathbf{S}_t$), the correction should be small. The optimal amount of

correction given the linear and Gaussian assumptions is the Kalman gain $\mathbf{K}_t$:

$$\mathbf{K}_t = \mathbf{P}_{t'}\mathbf{H}^T\mathbf{S}_t^{-1} \tag{2.11}$$

Once calculated, the Kalman gain is used to update the state mean and covariance:

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t'} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{z}_t) \tag{2.12}$$

$$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_{t'} \tag{2.13}$$

Matrix $\mathbf{I}$ is the d by d identity matrix.

Equations 2.7 through 2.13 make up one iteration of the Kalman filter.

Kalman Gain

Equation 2.12 demonstrates why the Kalman filter is a "filter": $\mathbf{K}_t$ acts as a gain and allows only a portion of the difference between the predicted and actual observations to update the state. This gain "filters" the noise in the observations. Equation 2.13 decreases the state covariance because the observations decrease the uncertainty of the state estimate.

Pre-Computing Gains

Computationally, the slowest step is the matrix inverse in Equation 2.11. However, the sequence of Kalman gains for a system with fixed models depends only on the initial state covariance. This can be seen by computing the sequence of $\mathbf{P}_{1\ldots T}$, which alone determine the Kalman gains, and noting that observations are not involved. Thus the Kalman gains can be pre-computed and the matrix inversion can be avoided in real-time computation. Note that, for fixed models, the state covariance and Kalman gain converge to a steady-state which depends only on the model parameters.

Initialization

If the true values are available, the state mean vector can be initialized to the true values and the state covariance set to a small value. Otherwise, a standard approach is to initialize the state mean vector and covariance matrix to the mean and covariance of the state values in the data used for model fitting, respectively. If these are not available, a diffuse initialization with a large covariance can be tried.

2.3 Quadratic Model of Tuning

Inspired by the work of Moran and Schwartz [79], analysis on tuning to vector magnitudes of position and velocity was conducted. By visualizing velocity tuning in two dimensions, tuning to velocity magnitude was confirmed (see Figure 4.4 on page 91). Adding the position vector and position vector magnitude allowed the model to describe tuning even better (see Section 4.2 on page 89). From these insights, a novel quadratic model of tuning, combining position, position magnitude, velocity, and velocity magnitude, was developed:

$$\begin{aligned} y(t) =& b_1 pos_x(t) + b_2 pos_y(t) + b_3\sqrt{pos_x(t)^2 + pos_y(t)^2} \\ &+ b_4 vel_x(t) + b_5 vel_y(t) + b_6\sqrt{vel_x(t)^2 + vel_y(t)^2} \end{aligned} \tag{2.14}$$

$y(t)$ is the mean-subtracted single-neuron firing rate at time t, $pos_x(t)$ and $pos_y(t)$ are the x- and y-axis coordinates of the cursor or hand at time t , $vel_x(t)$ and $vel_y(t)$ are the x- and y-axis velocities of the cursor at time t, and $b_1, b_2, ..., b_6$ are scalar parameters, 6 per neuron. This model will be called the *quadratic model of tuning* in the sequel. Note that this equation describes the quadratic model of tuning relating firing rates to movement at one time instant. This model can be extended to relate firing rates with movement at multiple time instants, or *taps*, by adding terms for the other time offsets of kinematics. For example, modeling the firing rate relationship to

movement at two time instants, t and $t+1$, can be done by duplicating the six terms with time $t+1$. In general, the quadratic model has $6n$ scalar parameters per neuron, where n is the number of time taps. The n tap quadratic model is specified below:

$$\begin{aligned} y(t) = \sum_{i=0}^{n-1} & b_{1,i} pos_x(t-i+k) + b_{2,i} pos_y(t-i+k) \\ & + b_{3,i} \sqrt{pos_x(t-i+k)^2 + pos_y(t-i+k)^2} \\ & + b_{4,i} vel_x(t-i+k) + b_{5,i} vel_y(t-i+k) \\ & + b_{6,i} \sqrt{vel_x(t-i+k)^2 + vel_y(t-i+k)^2} \end{aligned} \tag{2.15}$$

Here, k is the number *future taps*, the number of taps of movement after the time instant of the neuronal activity that are included in the model (see Section 2.4.2 on page 30 for more details).

Higher derivative terms, such as acceleration and jerk, were not included because they did not contribute substantially to decoding accuracy. The model can be changed to use firing rates which are not mean-subtracted by adding a bias term, increasing the number of parameters by one. As this model is linear in its features (though non-linear in Cartesian coordinates), linear regression is used to perform parameter fitting. As often done in regression analysis, all variables are preprocessed to have unit standard deviation.

2.4 Unscented Kalman Filter

Extended Kalman Filter

As the quadratic model of tuning is non-linear in the kinematics variables of the state, the Kalman filter is unsuitable for decoding with this model. One solution to this problem is to use the extended Kalman filter (EKF), a variant of the Kalman filter designed to handle non-linear models [43]. The EKF linearizes the non-linear model

by computing the derivative, which approximates the non-linear model function with a tangent line (or hyper-plane in higher dimensions). Specifically, the Jacobian of the non-linear observation function is evaluated at the current state mean and the resulting matrix is used in place of the **H** matrix. The rest of the EKF is the same as the Kalman filter.

Julier et al. [50, 51] proposed an improved method for filtering with non-linear models — the unscented Kalman filter. At the heart of the unscented Kalman filter is the *unscented transform*.

2.4.1 Unscented Transform

Under the linear and Gaussian assumptions, the predict and update steps of the Kalman filter can be implemented with a series of matrix equations. This is because normal distributions can be evaluated through a linear function analytically. However, closed-form function evaluation is, in general, only possible under this linearity assumption [128]. For general non-linear observation models, computing the posterior distribution poses an intractable integration problem [128] and the form of the posterior distribution will likely no longer be normal. The unscented Kalman filter calculates an approximate posterior distribution using the unscented transform.

Sigma Points

The unscented transform approximates the mean and covariance of a normal distribution which has passed through a non-linear function [51]. This transform uses a fixed set of algorithmically selected simulation points, called *sigma points*. The sigma points completely capture the first and second moments of the distribution [128]. Geometrically, the sigma points are located at the mean and along the eigenvectors of the covariance matrix, if the orthogonal matrix square root is used in their calculation [51]. $2d + 1$ sigma points are required, where d is the dimension of the state

space. The unscented transform addresses the non-linear function evaluation of a normal distribution using a different approach from the linearization used in the EKF. Linearization seeks a linear approximation to the function, while the unscented transform seeks to approximate the resulting distribution with a normal distribution, which is easier [49, 51].

To carry out the unscented transform, the set of sigma points is calculated from the state mean and covariance and evaluated through the non-linear function. The approximate mean and covariance of the distribution after function evaluation are then calculated by taking the weighted mean and weighted covariance of the sigma points. Details of the implementation are described in the following section. Van der Merwe [128] provides a detailed review of this procedure and similar approaches.

Example

Figure 2.2 on the next page shows an illustration of the unscented transform applied to a simplified version of the quadratic tuning model. The input, on the left, is the predicted state, a multivariate normal distribution with two dimensions, x_1 and x_2. x_1 and x_2 could represent the x- and y-axis positions or velocities. The black cross and ellipse are the mean and standard deviation of this distribution, respectively, and the light blue points are 5,000 random samples drawn from the distribution. The blue Xs are the 5 sigma points calculated from the mean and standard deviation using the Cholesky decomposition.

The sigma points are evaluated through the non-linear observation function $\mathbf{h}()$ describing two neurons y_1 and y_2, specified in the center. The resulting sigma points are shown as blue Xs in the right plot. The random sample points are also passed through the observation model to illustrate the true effect of the non-linear function on the distribution via Monte Carlo simulation. Note the distribution of sample points on the right is no longer a multivariate normal distribution. The black cross

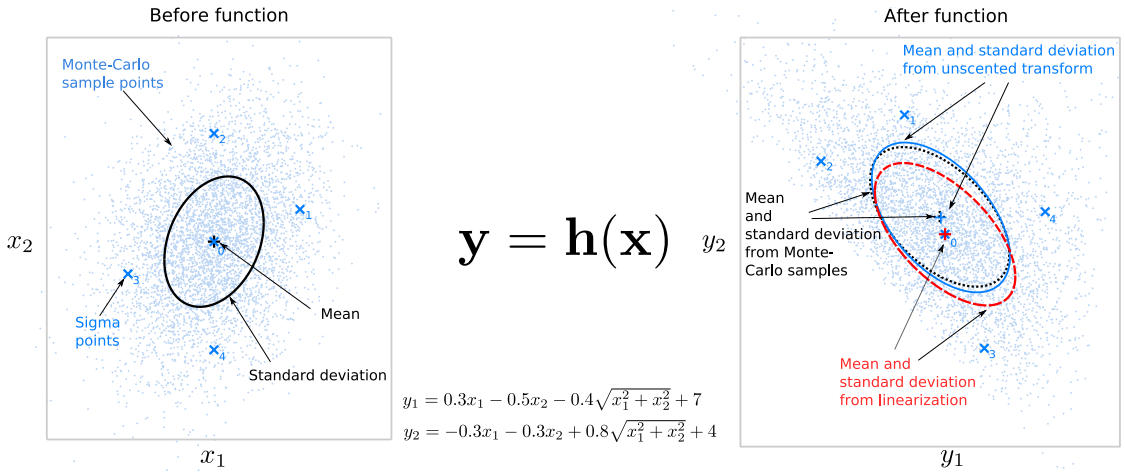

Figure 2.2: Example unscented transform. Left panel shows normal distribution in 2 dimensions before non-linear function evaluation; right panel shows same distribution after evaluation. The non-linear function, specified in middle, is a simplified quadratic tuning model.

and dotted ellipse on the right plot are the mean and standard deviation of the samples, respectively. The blue cross and ellipse are the mean and standard deviation calculated from the sigma points, respectively. The mean of the sigma points is very close to the mean of the Monte Carlo samples, and the standard deviation of the sigma points is a good approximation of the standard deviation of the samples.

EKF Comparison

For comparison, the result of using the linearization of the quadratic function to compute the mean and standard deviation is shown by the red cross and ellipse, respectively. The mean and standard deviation computed from linearization is considerably less accurate than the unscented transform results. The center sigma point, corresponding to the mean before the function evaluation, overlaps with the mean from linearization after function evaluation. However, the mean of all the sigma points is not close to the mean from linearization, because the other sigma points pull the mean towards the correct mean. This illustrates how the surrounding sigma

points contribute to the accuracy of the unscented transform.

Unscented Transform Approximation Error

Assume the non-linear function can be approximated with a unique Taylor series. The unscented transform computes precisely the effect of the third order and below terms of the Taylor series on the mean and covariance [51]. The presence of fourth order or higher terms in the Taylor series introduces error [51]. This implies that the mean and covariance of the predicted observations are calculated precisely by the unscented transform for the quadratic model of tuning. However, the non-linear observation function makes the distribution of the predicted observation no longer normal, and the higher order moments of the distribution are discarded, introducing approximation error.

Compared to the EKF, the unscented Kalman filter has better approximation accuracy for the same asymptotic computational cost and does not require computing the Jacobian [128].

2.4.2 n-th Order Unscented Kalman Filter

The n-th order unscented Kalman filter for BMI decoding is an unscented Kalman filter with a linear movement model, an augmented state space, and the quadratic model of tuning as the observation model. The quadratic model of tuning allows more accurate modeling of neuronal modulations, and the unscented Kalman filter allows fast estimation using this non-linear model. The state space is augmented so that it contains the kinematic variables of n recent time steps. This augmentation allows the relationship between observations and states at multiple time offsets to be modeled. It also makes the movement model an order n auto-regressive model, which can capture more complex movement patterns. The n-th order extension affects only the state and model specification, and does not change the UKF update equations. Panel A of

Figure 2.3 illustrates the HMM for the n-th order UKF. Panel B of Figure 2.3 gives an example of a simplified quadratic model of tuning. Panel C of Figure 2.3 gives an example portion of an AR n movement model.

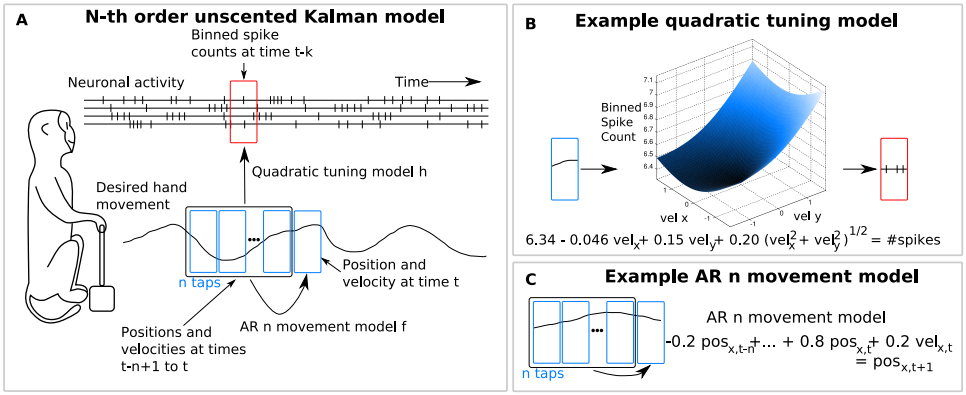

Figure 2.3: HMM for n-th order UKF decoding. A: AR n movement model predicts future position and velocity from n taps of recent position and velocity. Quadratic tuning model relates recent position and velocity with binned spike counts. B: Example simplified quadratic model of tuning with only velocity terms. C: Example AR n movement model for x-axis position.

Future Taps

The n taps of state need not all be in the past. A portion of the taps can represent beliefs of future kinematics. The parameter *future taps* describes the number of taps in the future (chronologically after the current observation) that is included in the state. Including k future taps is beneficial because it allows the observation model to capture relationships between neuronal activity at time t with kinematics up to time $t + k$ (in units of 100 ms). This allows modeling of causal control of movement by the brain. The non-future taps $n - k$, referred to as *past taps*, are also useful because neuronal activity may be on-going after a movement command is given, and information about past movements are still useful for predicting future movements using the movement model. Though kinematics in future taps and past

taps are estimated by the filter, the tap corresponding to zero time offset is used for brain-control and for accuracy evaluation.

State

The state vector $\mathbf{x}(t)$ of length $d = 4n$ contains the state variables at time t:

$$\mathbf{x}(t) = \begin{bmatrix} pos_x(t+k) \\ pos_y(t+k) \\ vel_x(t+k) \\ vel_y(t+k) \\ pos_x(t+k-1) \\ pos_y(t+k-1) \\ vel_x(t+k-1) \\ vel_y(t+k-1) \\ \vdots \\ pos_x(t+k-n+1) \\ pos_y(t+k-n+1) \\ vel_x(t+k-n+1) \\ vel_y(t+k-n+1) \end{bmatrix} \tag{2.16}$$

k is the number of future taps and n is the total number of taps.

During filtering, the state is represented by the estimated state mean column vector $\hat{\mathbf{x}}$ and the d by d estimated state covariance matrix $\mathbf{P}$. The column vector $\mathbf{y}_t$ of length N holds the observed binned spike counts at time t, where N is the number of neurons.

Predict Step

Since the n-th order UKF uses a linear transition model, the predict step is the same as the predict step for the Kalman filter:

$$\hat{\mathbf{x}}_{t'} = \mathbf{F}\hat{\mathbf{x}}_{t-1} \tag{2.17}$$

$$\mathbf{P}_{t'} = \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q} \tag{2.18}$$

$\hat{\mathbf{x}}_{t'}$ and $\mathbf{P}_{t'}$ are the mean and covariance of the predicted state. $\hat{\mathbf{x}}_{t-1}$ and $\mathbf{P}_{t-1}$ are the mean and covariance of the previous state. Matrix $\mathbf{F}$ represents the linear movement model, and $\mathbf{Q}$ is the covariance of the movement model noise. $\mathbf{F}$ and $\mathbf{Q}$ are square d by d matrices. Besides predicting kinematics from previous values, the matrix $\mathbf{F}$ implements the propagation of taps through time. The next section describes how parameter matrices are fitted.

Sigma Point Calculation

In the generic unscented Kalman filter, the sigma points are generated from $\hat{\mathbf{x}}_{t-1}$ and $\mathbf{P}_{t-1}$ and evaluated through the non-linear state transition and observation functions. For the n-th order UKF, the sigma points $\mathcal{X}_0 \dots \mathcal{X}_{2d}$ are generated from $\hat{\mathbf{x}}_{t'}$ and $\mathbf{P}_{t'}$:

$$\mathcal{X}_i = \begin{cases} \hat{\mathbf{x}}_{t'} & i = 0 \\ \hat{\mathbf{x}}_{t'} + \left(\sqrt{(d+\kappa)\mathbf{P}_{t'}}\right)_{(i)} & i = 1 \dots d \\ \hat{\mathbf{x}}_{t'} - \left(\sqrt{(d+\kappa)\mathbf{P}_{t'}}\right)_{(i-d)} & i = d+1 \dots 2d \end{cases} \tag{2.19}$$

The subscript outside the parentheses indicate the row taken from the matrix inside the parentheses. The square root is the matrix square root. For robustness, this computation is performed using the Cholesky decomposition. κ is a parameter which specifies how many "copies" of the center sigma point is used. Adjusting this parameter can improve the approximation of higher order moments [128]. In the work presented here, the conventional value of $\kappa = 1$ for normal distributions [51] is used.

Update Step

Next, the sigma points are evaluated in the quadratic model of tuning:

$$\mathcal{Z}_i = \mathbf{h}\left(\mathcal{X}_i\right) \qquad i = 0 \dots 2d \tag{2.20}$$

$\mathcal{Z}_0 \ldots \mathcal{Z}_{2d}$ denote the sigma points after function evaluation. These function evaluations can be implemented as $2d + 1$ matrix-vector multiplications of the form:

$$\begin{bmatrix} z_1(t) \\ z_2(t) \\ z_3(t) \\ \vdots \end{bmatrix} = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} \\ \vdots & & & & & \end{bmatrix} \begin{bmatrix} pos_x(t) \\ pos_y(t) \\ \sqrt{\mathbf{pos_x(t)^2 + pos_y(t)^2}} \\ vel_x(t) \\ vel_y(t) \\ \sqrt{\mathbf{vel_x(t)^2 + vel_y(t)^2}} \end{bmatrix} \tag{2.21}$$

$z_j(t)$ is the predicted (mean-subtracted) spike count for neuron j at time t. The vector on the left hand side of Equation 2.21 is one post-function sigma point $\mathcal{Z}_i$, and the right-most vector is one pre-function sigma point $\mathcal{X}_i$ with augmented terms. The augmented terms, in bold, are added to each of the sigma points using the sigma points' own values for position and velocity. Note Equation 2.21 shows the multiplication for the 1st order UKF. For higher values of n, there are more columns of model parameters in the parameter matrix in the middle and more rows in the vector $\mathcal{X}_i$ corresponding to the taps. The N by $6n$ matrix in the center of Equation 2.21 is denoted $\mathbf{H}$, since it has a similar function to the matrix $\mathbf{H}$ for the Kalman filter. In practice, all sigma points are multiplied in one matrix-matrix multiplication by horizontally concatenating the column vectors for each $\mathcal{X}_i$ into a matrix for the right-most multiplicand and horizontally concatenating each $\mathcal{Z}_i$ into a matrix for the left hand side.

The mean and covariance of the predicted binned spike counts are then found using weighted mean and weighted covariance:

$$\mathbf{z}_t = \sum_{i=0}^{2d} w_i \mathcal{Z}_i \tag{2.22}$$

$$\mathbf{P}_{zz,t} = w_0(\mathcal{Z}_0 - \mathbf{z}_t)(\mathcal{Z}_0 - \mathbf{z}_t)^T + \left[\sum_{i=1}^{2d} w_i(\mathcal{Z}_i - \mathcal{Z}_0)(\mathcal{Z}_i - \mathcal{Z}_0)^T\right] + \mathbf{R} \tag{2.23}$$

$\mathbf{R}$ is the covariance matrix of the tuning model noise. The weights are:

$$w_i = \begin{cases} \frac{\kappa}{d+\kappa} & i = 0 \\ \frac{1}{2(d+\kappa)} & i = 1 \dots 2d \end{cases} \tag{2.24}$$

Then, the Kalman gain is calculated:

$$\mathbf{P}_{xz,t} = w_0(\mathcal{X}_\mathbf{0} - \hat{\mathbf{x}}_{t'})(\mathcal{Z}_0 - \mathbf{z}_t)^T + \sum_{i=1}^{2d} w_i(\mathcal{X}_i - \mathcal{X}_0)(\mathcal{Z}_i - \mathcal{Z}_0)^T \tag{2.25}$$

$$\mathbf{K}_t = \mathbf{P}_{xz,t}\mathbf{P}_{zz,t}^{-1} \tag{2.26}$$

$\mathbf{P}_{xz,t}$ is the state-observation cross-covariance.

The Kalman gain is then used to update the state estimate:

$$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t'} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{z}_t) \tag{2.27}$$

$$\mathbf{P}_t = \mathbf{P}_{t'} - \mathbf{P}_{xz,t}(\mathbf{P}_{zz,t}^{-1})^T\mathbf{P}_{xz,t}^T \tag{2.28}$$

Equations 2.17 through 2.28 implement one iteration of the n-th order UKF. A side-by-side comparison of the equations for the Kalman filter and the n-th order UKF is shown in Table 2.1.

No Pre-Computation

Note that, unlike the Kalman filter, the Kalman gain of the UKF depends on the observations. Therefore, on-line calculation of all steps is unavoidable. Also, steady-state Kalman gains and state covariances are no longer guaranteed.

As a side note, the Kalman gain for the EKF depends on the estimated state, which in turn depends on the observations. Thus, pre-computation of Kalman gains is also not possible with the EKF.

Table 2.1: Update equations for the Kalman filter and the n-th order UKF. Predict step shown in upper block; update step shown in lower block. Note that a general unscented Kalman filter has a non-linear predict step, unlike the n-th order UKF for decoding shown.

Kalman filter	n-th order unscented Kalman filter
$\hat{\mathbf{x}}_{t'} = \mathbf{F}\hat{\mathbf{x}}_{t-1}$	$\hat{\mathbf{x}}_{t'} = \mathbf{F}\hat{\mathbf{x}}_{t-1}$
$\mathbf{P}_{t'} = \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q}$	$\mathbf{P}_{t'} = \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q}$
	$\mathcal{X}_i = \begin{cases} \hat{\mathbf{x}}_{t'} & i = 0 \\ \hat{\mathbf{x}}_{t'} + \left(\sqrt{(d+\kappa)\mathbf{P}_{t'}}\right)_{(i)} & i = 1 \dots d \\ \hat{\mathbf{x}}_{t'} - \left(\sqrt{(d+\kappa)\mathbf{P}_{t'}}\right)_{(i-d)} & i = d+1 \dots 2d \end{cases}$
	$w_i = \begin{cases} \frac{\kappa}{d+\kappa} & i = 0 \\ \frac{1}{2(d+\kappa)} & i = 1 \dots 2d \end{cases}$
$\mathbf{z}_t = \mathbf{H}\hat{\mathbf{x}}_{t'}$	$\mathcal{Z}_i = \mathbf{h}\left(\mathcal{X}_i\right) \qquad i = 0 \dots 2d$
	$\mathbf{z}_t = \sum_{i=0}^{2d} w_i \mathcal{Z}_i$
$\mathbf{S}_t = \mathbf{H}\mathbf{P}_{t'}\mathbf{H}^T + \mathbf{R}$	$\mathbf{P}_{zz,t} = w_0(\mathcal{Z}_0 - \mathbf{z}_t)(\mathcal{Z}_0 - \mathbf{z}_t)^T + \left[\sum_{i=1}^{2d} w_i(\mathcal{Z}_i - \mathcal{Z}_0)(\mathcal{Z}_i - \mathcal{Z}_0)^T\right] + \mathbf{R}$
	$\mathbf{P}_{xz,t} = w_0(\mathcal{X}_\mathbf{0} - \hat{\mathbf{x}}_{t'})(\mathcal{Z}_0 - \mathbf{z}_t)^T + \sum_{i=1}^{2d} w_i(\mathcal{X}_i - \mathcal{X}_0)(\mathcal{Z}_i - \mathcal{Z}_0)^T$
$\mathbf{K}_t = \mathbf{P}_{t'}\mathbf{H}^T\mathbf{S}_t^{-1}$	$\mathbf{K}_t = \mathbf{P}_{xz,t}\mathbf{P}_{zz,t}^{-1}$
$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t'} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{z}_t)$	$\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_{t'} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{z}_t)$
$\mathbf{P}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\mathbf{P}_{t'}$	$\mathbf{P}_t = \mathbf{P}_{t'} - \mathbf{P}_{xz,t}(\mathbf{P}_{zz,t}^{-1})^T\mathbf{P}_{xz,t}^T$

Initialization

In off-line reconstructions, the initial values $\hat{\mathbf{x}}_0$ and $\mathbf{P}_0$ were set to the mean and covariance of the state variables in the training data, respectively. When n was larger than 1, the means and covariances for the initial values were duplicated for each tap, so that the initial covariance matrix had a block-diagonal structure with n blocks.

For real-time filtering, the initial value $\hat{\mathbf{x}}_0$ was set as the joystick position and velocity at the start of brain-control, and the initial value $\mathbf{P}_0$ was set to the identity matrix for the experiments in Section 4.2. This corresponded to a variance of 1 cm for position and 10 cm/second for velocity. $\mathbf{P}_0$ was set to the covariance of the state variables in the training data for the experiments in Section 4.3. During real-time operation, the state covariance quickly approached a quasi-steady-state, suggesting performance was not sensitive to the initial covariance settings.

2.5 Model Fitting

The parameter matrices $\mathbf{F}$ and $\mathbf{H}$ are fitted using training data and linear regression. The matrices $\mathbf{Q}$ and $\mathbf{R}$ are estimated from the regression residuals. In the following, linear regression using a form of Tikhonov regularization called ridge regression is described. In some experiments, Bayesian regression (described in Chapter 3) was used in place of ridge regression. The fits from these two methods are very similar.

Movement Model

To fit $\mathbf{F}$, the training data of hand positions and velocities measured from the joystick are composed into the 4 by T matrix $\mathbf{X}$, where T is the number of data points (the time length of the training data). Then, a $4n$ by T matrix $\hat{\mathbf{X}}$ is constructed. Column i of $\hat{\mathbf{X}}$ is the vertical concatenation of columns $i-1, i-2, \ldots, i-n$ of matrix $\mathbf{X}$. $\hat{\mathbf{X}}$ is the matrix of kinematics taps one time step before the kinematics in $\mathbf{X}$. To avoid

the missing data problem when filling the first n columns of $\hat{\mathbf{X}}$, the first n columns of $\hat{\mathbf{X}}$ and $\mathbf{X}$ are omitted for the following operation. The intermediary matrix $\mathbf{F}'$ is found using ridge regression:

$$\mathbf{F}' = \mathbf{X}\hat{\mathbf{X}}^T(\hat{\mathbf{X}}\hat{\mathbf{X}}^T + \lambda_F\mathbf{I})^{-1} \tag{2.29}$$

λ_F is the ridge regression parameter. The selection of ridge regression parameters is discussed in Chapter 4. To make $\mathbf{F}$, $\mathbf{F}'$ is augmented with entries which propagate the history taps:

$$\mathbf{F} = \left[\begin{array}{c|c} \mathbf{F}'_{(4\times 4n)} & \\ \hline \mathbf{I}_{(4(n-1)\times 4(n-1))} & \mathbf{0}_{(4(n-1)\times 4)} \end{array}\right]_{(4n\times 4n)} \tag{2.30}$$

$\mathbf{I}$ is the identity matrix and $\mathbf{0}$ is the zero matrix. Subscripts indicate matrix sizes.

Constant-Velocity Movement Model

An alternative method for setting the movement model is to use the standard equations describing motion. One such model sets position as the integral of velocity over time and velocity as constant but subject to normally-distributed perturbations. This model is called the *constant velocity movement model*:

$$\mathbf{F}'_{cv} = \left[\begin{array}{cccc|c} 1 & 0 & dt & 0 & \\ 0 & 1 & 0 & dt & \mathbf{0}_{(4\times 4(n-1))} \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & \end{array}\right]_{(4\times 4n)} \tag{2.31}$$

dt is 0.1 to correspond to the 10 Hz filter execution rate. This model does not capture the patterns in the movements generated by the BMI user as well as a movement model fitted to training data. In practice, the fits to $\mathbf{F}'$ are similar to $\mathbf{F}'_{cv}$, except for modest perturbations.

Movement Model Noise Covariance

The movement model noise covariance matrix $\mathbf{Q}$ is estimated by first computing the intermediary $\mathbf{Q}'$:

$$\mathbf{Q}' = \frac{\mathbf{E}_F \mathbf{E}_F^T}{T - 5n} \tag{2.32}$$

$\mathbf{E}_F$ is the 4 by $T - n$ matrix of residuals from fitting $\mathbf{F}'$, i.e. $\mathbf{E}_F = \mathbf{X} - \mathbf{F}'\hat{\mathbf{X}}$. The division is executed per element. The $T - 5n$ denominator accounts for the fitting of $4n$ parameters using $T - n$ data points.

Then $\mathbf{Q}'$ is augmented to construct $\mathbf{Q}$:

$$\mathbf{Q} = \left[\begin{array}{c|c} \mathbf{Q}'_{(4\times4)} & \mathbf{0}_{(4\times4(n-1))} \\ \hline \multicolumn{2}{c}{\mathbf{0}_{(4(n-1)\times4n)}} \end{array}\right]_{(4n\times4n)} \tag{2.33}$$

Observation Model

To fit $\mathbf{H}$, the N by T matrix $\mathbf{Y}$ of mean-subtracted, binned spike counts from the training data is constructed. Then the $6n$ by T matrix $\tilde{\mathbf{X}}$ of quadratic features is constructed. Column i of $\tilde{\mathbf{X}}$ is the vertical concatenation of columns $i, i-1, \ldots, i-n+1$ of $\mathbf{X}$, with the bold quadratic terms in Equation 2.21 inserted. To implement the k future taps, $\mathbf{Y}$ must be shifted back in time by k steps. This is done by removing the last k columns of $\mathbf{Y}$ and the first k columns of $\tilde{\mathbf{X}}$. Subsequently, to avoid the missing data problem, the first $n - k - 1$ columns of $\tilde{\mathbf{X}}$ and $\mathbf{Y}$ are removed. Then, $\mathbf{H}$ is fitted using ridge regression:

$$\mathbf{H} = \mathbf{Y}\tilde{\mathbf{X}}^T(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T + \lambda_H \mathbf{I})^{-1} \tag{2.34}$$

λ_H is the ridge regression parameter.

Observation Model Noise Covariance

The N by N neuronal tuning model noise covariance matrix $\mathbf{R}$ is estimated using:

$$\mathbf{R} = \frac{\mathbf{E}_B \mathbf{E}_B^T}{T - 7n + 1} \tag{2.35}$$

$\mathbf{E}_B$ is the N by $T - n + 1$ matrix of residuals from fitting $\mathbf{H}$. The division is executed per element. The $T - 7n + 1$ denominator accounts for the fitting of $6n$ parameters using $T - n + 1$ data points.

An evaluation of the n-th order UKF on neuronal data collected from monkeys and comparison to commonly used decoders is presented in Chapter 4.

Chapter 3

Adaptive Decoding

For brain-machine interface controlled prosthetic systems to be clinically viable, they must provide robust function over periods of years. The brain is known to exhibit plasticity, or changes in organization and function, which can lead to changes in tuning [12, 63, 68, 87, 96, 120, 125] that potentially interfere with BMI decoders. To cope with these changes, the decoder can use adaptive filtering to update its tuning model. Adaptive filtering may also take advantage of changes in tuning from learning.

This chapter begins with a review of studies which have found changes in neuronal tuning for arm movement and efforts to cope with these changes, as well as general methods for updating models during Bayesian filtering. Then, three issues in adaptive decoding, identifiability, divergence, and alertness, are discussed. Then the Bayesian regression self-training method is described. Two formulations for Bayesian regression are presented. Finally, the use of Bayesian regression for initial parameter fitting and the transition model for parameters are discussed.

3.1 Prior Work

3.1.1 Time-Varying Tuning

Evidence Supporting Time-Varying Tuning

Some studies have found changes in neuronal representations for arm movement over time. Taylor et al. [120] found changes in neuronal tuning of monkeys after practice with operating a brain-controlled cursor. Li et al. [68] and Padoa-Schioppa et al. [87] found systematic changes in tuning as monkeys adapted to changing external force-fields. Lebedev et al. [66] found changes in neuronal tuning during the switch from hand-control of cursor to brain-control of cursor. Kim et al. [63] showed changes in tuning to be statistically significant using a bootstrap approach. They reported that the difference in tuning calculated from two data segments increased with the time interval between the segments. Rokni et al. [96] found slow random changes during the execution of familiar tasks and additional systematic changes while learning. Truccolo et al. [125] found statistically significant changes in the tuning of around 42% of recorded neurons from 2 humans.

Evidence Supporting Stable Tuning

Chestek et al. [12] found small but significant changes in tuning and changes in baseline firing rate. However, Chestek concluded that the changes they and other researchers have observed were due to measurement noise, learning, and behavioral changes rather than intrinsic randomness.

Ganguly and Carmena [31] used a Wiener filter with fixed parameters for BMI control over a time period of 19 days and reported stable behavioral performance after initial learning. The Wiener filter used a small (10 to 15 neurons) subset of all recorded neurons which were deemed stable from waveform and interspike interval characteristics. Tuning of these neurons was measured with respect to center-out

target location and found to change significantly at first but become stable after a few days. Stability in tuning corresponded with stability in behavioral performance using the BMI. Significant changes in baseline firing rate and depth of tuning modulation were found and attributed to practice using the BMI controller. Small changes in the Wiener filter parameters and exclusion of just a few neurons were found to harm behavioral performance. Though sensitive to changes in Wiener filter parameters, monkeys could learn to control the BMI driven cursor using a randomly shuffled (and subsequently fixed) set of Wiener filter parameters, given enough time. Ganguly and Carmena concluded that stable neuronal tuning emerges with practice when fixed Wiener filter parameters are used. However, their results may only apply to neurons which exhibit stable waveforms and interspike interval distributions.

Sources of Changes

While some studies have found evidence for the existence of changes in neuronal tuning, the work of Ganguly and Carmena raises the question of whether a BMI decoding method needs to adapt over time to provide useful control. To answer the question, the sources of the tuning changes need to be considered. Chestek et al. [12] suggest that evidence of changes in neuronal tuning can be attributed to learning, task switching, behavioral changes, and noisy measurements. They did not consider inherent randomness as a factor, unlike Rokni et al. [96]. Changes due to learning are likely beneficial to the accuracy of a static BMI decoding method. Changes in tuning due to task switching and behavioral changes indicate that the tuning model is insufficiently general. Any changes in tuning not related to learning, task changes, or behavior changes, if they exist, would likely harm the accuracy of a static BMI decoder. However, an adaptive decoder could compensate for these changes. For example, the static Wiener filter used by Ganguly and Carmena was sensitive to the loss of a few neurons [31]; an adaptive decoder would reduce the impact of such a

change.

An adaptive BMI decoder could exploit any beneficial changes in tuning more than a static decoder could. An example of a beneficial change is the increases in depth of modulation found by Ganguly and Carmena [31]. Neurons which increase in depth of modulation may be given higher influence in an adaptive decoder, whereas in a static decoder, their influence remains the same, which is especially suboptimal if the neuron had very shallow modulation depth during the initial parameter fit. With an adaptive decoder, instead of the user adapting to the BMI, the user and the BMI adapt to each other, or *co-adapt* [120]. This may improve accuracy and decrease training time. Also, an adaptive decoder may help compensate for tuning changes due to task switching or behavioral pattern changes, depending on the time scale of such changes.

3.1.2 Adaptive Filtering

The problem of sequential state estimation under time-varying models is known as simultaneous system identification and tracking or adaptive filtering. Adaptive filtering was a natural out-growth of state estimation under static models with Bayesian filters, and much work on adaptive filtering treats the changing model parameters as unknown hidden variables to be tracked like the state variables. This approach updates the models in a sequential manner. Alternatively, batch-mode updates of the model can be performed. Early work and basic methods are reviewed in the following; for a more complete review, see the work of Nelson [84].

Joint Filtering

Early work on this problem was presented by Kopp and Orford [64] and Cox [16], who used what is now called the *joint filtering* approach. In this approach, the time-varying model parameters are added to the state space, and the filter updates

both the hidden state and the model parameters. In the example problem by Kopp and Orford, only the transition model had unknown parameters and the observation model was identity, leading to an analytical solution for the unknown transition model parameters. Cox presented the general case of time-varying transition and observation models and objective functions for both batch-mode estimation and sequential estimation.

The observation model required for joint filtering is, in general, non-linear, and non-linear filtering methods, such as the extended Kalman filter (EKF), must be used. Ljung [77] presented convergence analysis for simultaneous identification and tracking of a system with unknown but static parameters using the EKF. Ljung found some weaknesses with using the EKF for joint filtering and proposed a modified method where the Kalman gain is identified, instead of the observation model. However, this method is only applicable to linear observation models. Wan et al. [136] applied the unscented Kalman filter to the joint filtering to avoid the convergence problems of the EKF for joint filtering.

Dual Filtering

Another approach, called *dual filtering*, was proposed by Nelson and Stear [85]. In dual filtering, two parallel filters are used, one to track the hidden variables and one to track the model parameters. Each filter assumes the output of the other is the truth, and the two filters operate in lock-step. By separating the joint filtering problem into two portions, each portion is made into a linear problem and Kalman filters can be used. More recently, Nelson [84] developed a family of dual Kalman filters which optimize several criteria.

Batch-Mode Update

In the batch-mode approach to model updating, the model parameters are periodically updated using batches of cached state estimates and observations. Since the algorithm is using its output to update its own parameters, the method can be described as *self-training*. One method of performing a batch update is by using the expectation-maximization (EM) algorithm [19]. Shumway and Stoffer [111] used the EM algorithm with the Kalman filter and smoother to perform batch-mode system identification and tracking on linear systems. Ghahramani and Roweis [38] proposed an extension to non-linear systems by using an artificial neural network and extended Kalman filter and smoother. Beal [6] proposed a variational Bayesian approach for batch mode system identification and tracking for linear systems.

Characterizing Updates

While self-training approaches use a batch of data to fit model parameters, sequential approaches can also carry in their state space a history of past predictions to act as a small batch. Conversely, self-training can occur at every filter iteration, making it similar to dual filtering. Thus, updating techniques can be characterized in two (amongst many) ways: (1) how often updates occur, and (2) how much past data is used in each update. Joint filtering and dual filtering update at the same frequency as the state filter and use one or a small number of past data points. Self-training updates can use many past data points. The update frequency can be higher than the inverse of the number of data points, meaning there is overlap in the data used by consecutive updates. Alternatively, updates can occur without data overlap.

3.1.3 Adaptive Decoding for BMI

Parameter Fitting for Adaptation

Some BMI studies have proposed parameter fitting approaches which can be modified to adapt to changes in neuronal tuning, since adaptation is just parameter fitting, but with more prior information available.

The *co-adaptive* training method used by Taylor et al. [120] was designed for fitting decoder parameters without example hand movements, but can be used for adapting to changes. Taylor assumed the availability of the target location and performance metrics on the decoded trajectory for calculating the parameter updates. Another method for fitting parameters for decoding without example movements was proposed by DiGiovanna et al. [21]. Their method used reinforcement learning, an optimal control method for a class of problems called Markov Decision Processes. DiGiovanna et al. tested the approach on a two-target robot-arm control task with rats.

Other work on learning of initial parameters which may be applicable to updating parameters include those by Musallam et al. [81], Wolpaw and McFarland [143], and Gage et al. [30]. These studies used knowledge of the target identity, location, or value to update the decoder parameters.

Wahnoun et al. [134, 135] and Hochberg et al. [44] used *visual following* — instructing the BMI user to imagine trying to move a cursor which is controlled by a computer or a technician, and updating parameters as if the cursor's movements are the user's desired movements.

More recently, Darmanjian and Principe [18] developed a hierarchical linked-mixtures hidden Markov model for unsupervised clustering of movement states. However, to reconstruct trajectories, a Wiener filter fit with supervised learning was used with each state.

Updating Using Hand Movements or Assumptions

Wu and Hatsopoulos [150] proposed an algorithm for the real-time updating of the linear models used in Kalman and Wiener filters. Their main focus was to ensure fast computation of the updates, which occurred in an incremental manner. In off-line reconstructions of monkey reaching, Wu and Hatsopoulos showed that their update method improved reconstruction accuracy over the course of a hand-controlled session. Their experiment used the actual hand movements for input during the updates, and they speculated on ways to perform the updates without the knowledge of the hand movements.

Shpigelman et al. [107] used an incremental parameter update method with their kernel-ARMA decoder. For the supervisory signal, they used a weighted combination of the target location and the decoded cursor location, assuming the BMI user is trying to move the cursor closer to the target.

Updating Using Feedback

Rotermund et al. [97] proposed an update method which uses a hypothetical evaluation signal recorded from the brain. This evaluation signal is assumed to convey information about the accuracy of the decoding. It is used to pick among small random changes to parameters in an on-going stochastic exploration process. Schalk et al. [103] found a signal indicating decoding error in EEG recordings, suggesting that the hypothetical evaluation signal should also exist for single-unit recordings.

Joint and Dual Filtering

Sykacek and Roberts [117] and Sykacek et al. [118] proposed a variational Bayesian Kalman filter to adapt parameters in a classifier for EEG signals. The classifier parameters are modeled using a state-space model. To handle the non-linearity in the classifier, novel variational approximations are devised. The hierarchical Bayesian

model in their approach allows Bayesian inference on the parameter change rate and does not require knowledge of the correct classification target.

Eden et al. [24] showed in simulation that the stochastic state point-process filter could track both the state and the observation model parameters of a simple toy problem by performing joint filtering. Srinivasan et al. [112] extended this work to handle other forms of input and demonstrated in simulation the ability to adapt to neuron loss and discovery.

Rickert et al. [94] proposed a joint unscented Kalman filter for simultaneously decoding and updating neuronal tuning parameters. The method was shown to adapt to changes in the tuning of simulated neurons. Liao et al. [74] used a dual Kalman filter to adapt to model changes of myoelectric recordings.

Self-Training

Gilja et al. [39] updated the tuning model of a Kalman filter decoder using self-training. The old tuning model was completely replaced by the newly fitted tuning model in their approach. Gilja reported that this approach was able to, in closed-loop experiments, make Kalman filter decoded center-out reaches straighter than reaches made with a non-updated Kalman filter.

3.2 Adaptive Filtering Considerations

3.2.1 Identifiability

In adaptive filtering with a time-varying observation model, some or all parameters of the model may not be *identifiable* [78]. A parameter in a model is identifiable if the parameter can be uniquely determined from the observations and the structure of the model. A parameterized model is identifiable if all of its parameters are identifiable. Ljung and Söderström [78] gave two main causes for lack of identifiability: over-parameterization and non-exciting data.

An example of the former is when two parameters always appear in the observation model function together as a product. In this situation, there is no way to tease apart how each parameter contributes to the model. The latter occurs when data is not rich enough to determine parameters. One example of non-exciting data in linear regression for parameter identification is a singular design matrix due to linear dependence in the features [78].

When All Parameters Change

If all parameters of the observation model may change, or if all parameters determining one observation dimension may change and they do not affect any other observation dimensions, then the model is unidentifiable without further assumptions. The problem of finding the best observation model is ill-posed if no initial condition is given or no constraints are placed on how the parameters may change.

This situation can be illustrated with a simple example: given an observation model where the one-dimensional state $x(t)$ is related to the one-dimensional observation $y(t)$ by a multiplicative parameter c (i.e. $y(t) = c(t) \cdot x(t) + noise$), after observing a series of $y(1) \dots y(T)$, any sequence of states $x(1) \dots x(T)$ is equally plausible if $c(t)$ is not initialized or is allowed to change arbitrarily. $x(1) \dots x(T)$ can be arbitrarily large or small depending on the initial value $c(0)$. A decrease in $y(t)$ at time t can be either due to a decrease in $x(t)$ or an increase in $c(t)$, or a combination of both. There is no way to distinguish between the possibilities. If the transition model for x makes no change in x the most likely possibility, then any change in y will be solely attributed to a change in c.

Identifiability in Decoding

The situation in the above example is analogous to the situation in adaptive filtering for BMI decoding, as the parameters for neurons are not shared. This way of modeling

tuning makes the model parameters unidentifiable and the parameter update problem ill-posed without initial conditions and assumptions about how parameters may change.

Fortunately, changes in neuronal tuning arise from biological processes, such as changes in the strength of synapses through actions of second-messengers and growth of axons, dendrites, and new synapses. These processes occur at a time-scale that is probably slower than the time-scale of limb movements. Thus, changes in neuronal tuning can be assumed to occur slowly and with no or infrequent discontinuous jumps. By limiting the rate of tuning change, providing initial parameters, and using a suitable movement model, the changes in tuning parameters can be identifiable.

For example, if a strict bound is placed on how fast c can change and the transition model for x is the identity transition with unimodal noise, then the update for c will report that c changes as much as allowed to account for changes in y, and the remainder, if any, is attributed to changes in x.

Transition Models for Parameters

A more elegant way of constraining the rate of parameter change is to model the change using a probabilistic transition model. A typical transition model for parameters is the Brownian motion model, i.e. identity transition with small normally distributed noise. The effect of this model is to add uncertainty to the distributions of the parameters over time, to reflect the time-varying nature of the parameters. Then, Bayesian updates of the parameters (similar to dual filtering) can find the proper compromise in attributing the changes in y to changes in x and c. For example, if the transition model for x is a Brownian motion model with large noise variance and the transition model for c is a Brownian motion model with small noise variance, then the proper adaptive filter output is to attribute most of the change in y to changes in x and a small portion to changes in c.

The probabilistic transition model approach has the benefit of leveraging the information in the transition models for x and c. For example, these transition models may not be Brownian, but rather express a preference for an increase or decrease. Note that, for the parameters to be uniquely determinable, the transition models must not allow multiple combinations of states and parameters to have maximum posterior probability, i.e. there needs to be a unique maximum probability combination of values. Using unimodal noise distributions for both the state and parameter transition models is sufficient to avoid this problem.

Initialization

As mentioned above, for parameters to be identifiable in a model where all parameters are allowed to change, they must be initialized as well as subject to a transition model. For decoding, the standard initialization method is to perform supervised parameter fitting using actual hand movements. The problem of performing the parameter fit without actual hand movements has been addressed in several ways, as discussed in the review of prior work. Thus, in this work, the standard initialization method of performing a supervised parameter fit is used, to focus on the problem of parameter updates.

3.2.2 Divergence

Even if the observation model parameters are identifiable, their inferred values may not be the same as the true values, i.e. those fitted in a supervised manner using actual hand movements, when all parameters in the observation model may change. Error arises due to noise in the data used for updates, model mis-specification, and approximations in the update method. Incorrect observation, state transition, or parameter transition models can all lead to error. The state and parameter estimates will diverge from the true values if any model is mis-specified, since the maximum of

the conditional probability of the states given the observations and parameters will not be at the correct values. Another way to characterize the situation is that there is no solid link between inferred states and their correct values, since the observation model that connects them is allowed to change.

Vicious Cycle of Error

Noise from the filtering process speeds up divergence: errors due to noise in filtering will lead to errors in model parameter updates, which in turn will lead to more errors in filtering, forming a vicious cycle of error. Since there is no solid link to true states, once error accumulates, it cannot be removed by the adaptive filter. Therefore, reducing the accumulation of error is a high priority. Prior work in updating decoder parameters do not acknowledge this issue. Most prior work has either used actual movement information in updates, avoiding the divergence problem, or shown results only in simulation, in which the filter's model is matched to the synthetic data's model.

Redundancy

Given the possibility of divergence, how can an adaptive decoder operate over long time spans? Part of the answer lies in the redundancy of tuning in neuron populations. Narayanan et al. [82] found that almost all neurons in motor cortex of rat contributed redundant predictive information, and redundancy is seen in small groups of 8 to 15 neurons. The redundancy arises because the number of neurons which are tuned to the state is typically much larger than the dimension of the state, and because neurons are often tuned to several dimensions of the state at once.

The redundancy in tuning means that changes in tuning of a small subset of neurons can be detected using the information from the other neurons. For example, if neurons 1 through 10 change their tuning and thus are incorrectly modeled by the

decoder, the decoder can still use all the neurons 1 through 100 to estimate the state, albeit less accurately. Then the estimated state can be used to update the models for neurons 1 to 10. The error from this process is small if the subset of changing neurons is small and the amount of tuning change is small.

As the ratio of non-changing neurons to changing neurons increases, the accuracy of filtering without adaptation increases. This is because the modeling error in the neurons which change is mitigated by the information conveyed through the neurons which are correctly modeled. In other words, the larger the ratio of information from correctly modeled neurons compared to error from incorrectly modeled neurons, the more accurate the filtering result. More accurate filtering leads to more accurate parameter updates and more accurate subsequent filtering. This line of reasoning suggests that larger neuronal populations or populations with more redundant tuning make adaptive decoding more accurate and able to handle faster changes.

Feedback

The risk of divergence can also be reduced by using the closed-loop nature of BMI prosthetic systems. The prosthetic user can compensate for small errors in the observation model and practice troublesome movements to train the model. If divergence occurs, the observation model can be reset to previous settings which are known to be good. Updates can be suspended when control is satisfactory and re-enabled if changes in neuronal tuning compromise control. Also, if assumptions of true desired movements are available for updating, for example by using a training session with designated trajectories that the user tries to follow, the update methods proposed in this chapter can still be applied.

3.2.3 Alertness

A key trade-off in adaptive filtering is between *alertness* and *noise sensitivity* [78], that is, between the ability to handle quickly-changing models and robustness to noise. This is because increased sensitivity to changes in the model also expose the filter to increased sensitivity to noise. Given the noise characteristics of the system, a more alert adaptive filter will necessarily be less robust to noise and vice versa.

Determinants of Alertness

For non-Bayesian update methods, the alertness is determined by the update frequency, data window size, and any forgetting factors in the algorithm which weigh more recent data more heavily than past data. In general, lower update frequencies and larger batch sizes are less alert but more robust to noise. Bayesian update methods explicitly parameterize alertness in their transition model and apply the transition model at the update frequency.

Optimal Alertness

To optimally track changing model parameters, their transition characteristics must be known, just as the true transition model of the state is required to optimally track the state. Using a transition model with a rate-of-change parameter that is too slow results in adaptation that is slower than the actual change (alertness too low), but this also reduces sensitivity to noise. Using a transition model with a rate-of-change that is too fast results in unnecessary sensitivity to noise. The accuracy of the updates is limited by the true rate-of-change of parameters and the amount of noise in the system; if both are high, then there is little hope of accurate updates.

Setting Alertness

In general, the rate-of-change of the observation model parameters is unknown *a priori* and may vary over time. This presents another model for identification, which may be possible using a hierarchical Bayesian model, as shown by Sykacek et al. for EEG classification [117, 118].

Alternatively, one may try to fit the rate-of-change using experiments where truth is available, or assume reasonable values. The rate-of-change should also be adjustable by the BMI user. This allows the user to settle on satisfactory tuning models, as well as adapt to tuning changes when they occur. This also facilitates designated practice intervals during which adapting of tuning parameters is turned on or increased in alertness.

3.3 Bayesian Regression Self-Training

The Bayesian regression self-training method combines the self-training approach with Bayesian regression and the Kalman smoother to update models of neuronal tuning. Figure 3.1 on the next page provides an overview of the procedure. Probability distributions, instead of point estimates, are used to represent the model parameters. These distributions are parameterized by a set of hyper-parameters. By representing model parameters as distributions, Bayesian computations can be performed on them. The Bayesian regression self-training method essentially applies a state-space model on the observation model parameters and performs Bayesian filtering, which is similar to the dual filtering paradigm. However, this approach differs from the dual filtering in two ways.

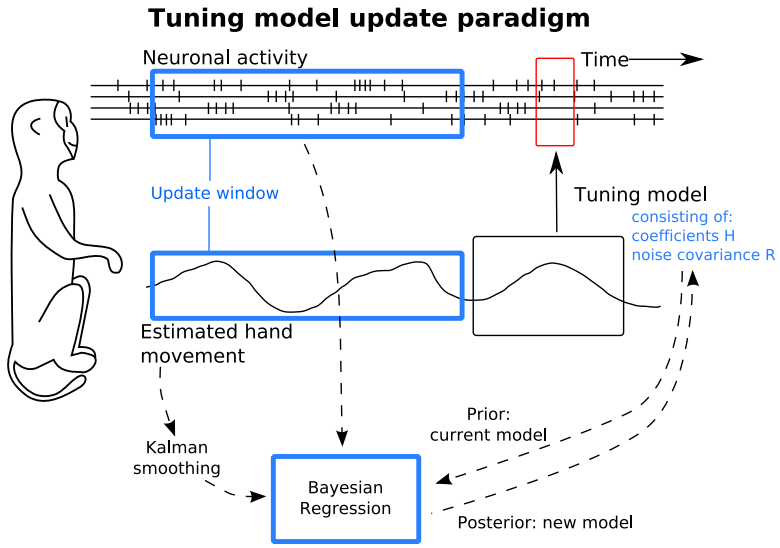

Figure 3.1: Tuning model update paradigm. Updates occur periodically, using the neuronal data and filter predictions in the update window.

Batch-Mode Updates

In dual filtering, model updates are performed at every iteration of the state filter. In this approach, updates are performed periodically in batch mode. This allows the Bayesian regression self-training method to reject noise better because noise is averaged among many data points in the regression. This feature is important for slowing the accumulation of error. With dual filtering, the vicious cycle of error occurs at the frequency of the state filter. With batch-mode updates, the vicious cycle occurs at the slower update frequency, and modeling error is reduced by using more data points in the inference. This higher noise robustness comes at the cost of reduced alertness.

Smoothing

The batch update approach also allows the use of the Kalman smoother, also called the Rauch-Tung-Striebel smoother [92], to refine predictions before they are used to update the model. The Kalman smoother is analogous to the backwards pass in the

forwards-backwards algorithm [7] for hidden Markov models. Together, the Kalman filter and Kalman smoother generate the Viterbi path for linear-Gaussian state-space models [7]. The Kalman smoother's recursive updates operate backwards in time, so it can only operate after filtering on a batch of observations has been completed. Thus, it cannot help parameter estimates in joint and dual filtering.

Updates

The state filter, which is the n-th order unscented Kalman filter described in Chapter 2, estimates movement commands given neuronal recordings using the expectations of the observation model parameters. Every T filter iterations, an update of the observation model occurs. The observation model parameters are passed through a transition model, discussed in Section 3.7 on page 75. T, together with this transition model, specifies the trade-off between alertness and sensitivity to noise, as described in the previous subsection.

During each update, Kalman smoothing is applied to the filtered movements. Then the quadratic features of the quadratic model of tuning are calculated from the mean of the smoothed estimates.

Bayesian Regression Update

Next, Bayesian multivariate linear regression calculates the update. The dependent variables are the T data points of neuronal recordings, and the independent variables are the T data points of quadratic model features from the smoothed estimates. The prior for the Bayesian regression is the previous observation model, from either the initial fit or the previous update. The prior is specified by the hyper-parameters of the observation model parameters' distributions. The posterior observation model hyper-parameters are recorded for the next update and used to calculate the posterior mean of the observation model parameters. The posterior mean of observation model

parameters are used in subsequent filtering. In an on-line implementation, the Kalman smoothing and Bayesian regression steps can execute in a low-priority thread while filtering continues; once Bayesian regression is completed, the posterior mean is used by the UKF and caching of data for the next update begins.

Bayesian Regression Variants

Two ways for representing the observation model parameters are explored in this work. The first approach represents the tuning model coefficients in the $\mathbf{H}$ matrix and the tuning model error covariance in the $\mathbf{R}$ matrix jointly using a normal-inverse-Wishart distribution. This approach allows analytical computation of the posterior of the Bayesian regression. The second approach represents the tuning model coefficients in the $\mathbf{H}$ matrix independently for each neuron using a normal distribution and the tuning model covariance in the $\mathbf{R}$ matrix using a separate inverse-Wishart distribution. This factorized approach allows greater flexibility — each neuron can be updated separately — but requires an iterative approximation algorithm, variational Bayes (VB), to compute the posterior of the Bayesian regression.

3.4 Regression with Joint Distribution

A joint distribution for $\mathbf{H}$ and $\mathbf{R}$ allows analytical calculation of the posterior of the Bayesian regression, because the prior formulation is conjugate to the likelihood of the regression. This means that the form of the posterior distribution is the same as the form of the prior distribution, and only the hyper-parameters are changed. The Bayesian regression presented in this section is a modification of the Bayesian multivariate linear regression given by Rencher and Schaalje [93]; in this work, the likelihood function has a full covariance, $\mathbf{R}$.

Model of Linear Regression

The model of Bayesian regression using a joint model for $\mathbf{H}$ and $\mathbf{R}$ is shown in graphical form in Figure 3.2 on the next page. The model for multivariate linear regression and the model for Bayesian regression are given below:

$$\mathbf{y}_t = \mathbf{H}\mathbf{x}_t + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \mathbf{R}) \tag{3.1}$$

$$\mathbf{y}_t | \mathbf{x}_t, \mathbf{H}, \mathbf{R} \sim \mathcal{N}(\mathbf{H}\mathbf{x}_t, \mathbf{R}) \tag{3.2a}$$

$$\{\mathrm{vec}\,(\mathbf{H}), \mathbf{R}\} \sim \mathcal{MW}^{-1}\left(\mathrm{vec}\,(\mu), \mathbf{\Lambda}^{-1}, \mathbf{\Psi}, m\right) \tag{3.2b}$$

t is the index on data (or time instants included in the batch), and T is the total amount of data in the regression or the update window size. N is the number of neurons, and D is the number of dimensions in the kinematic features. $\mathbf{x}_t$ is a column vector of length D of regression features. $\mathbf{x}_t$ is constructed by augmenting the quadratic terms from the quadratic model of tuning (bold terms in Equation 2.21 on page 34) to the smoothed state mean for time t. $\mathbf{y}_t$ is a column vector of length N of binned spike counts at time t. $\mathbf{H}$ is the N by D matrix of tuning model coefficients (middle matrix in Equation 2.21 on page 34), and $\mathbf{R}$ is the N by N covariance matrix of the tuning model noise. The function vec() converts a matrix into a column vector by vertically concatenating the columns, with the left-most column arranged at the top. The hyper-parameters μ, $\mathbf{\Lambda}$, $\mathbf{\Psi}$, and m are described below.

Normal-Inverse-Wishart Prior

The prior distribution on $\mathbf{H}$ and $\mathbf{R}$ conjugate to the likelihood in Equation 3.2a has the form of a normal-inverse-Wishart distribution, with parameters $\tilde{\mu}$, $\tilde{\mathbf{\Lambda}}$, $\tilde{\mathbf{\Psi}}$, and $\tilde{m}$. The normal-inverse-Wishart distribution is the product of a multivariate normal

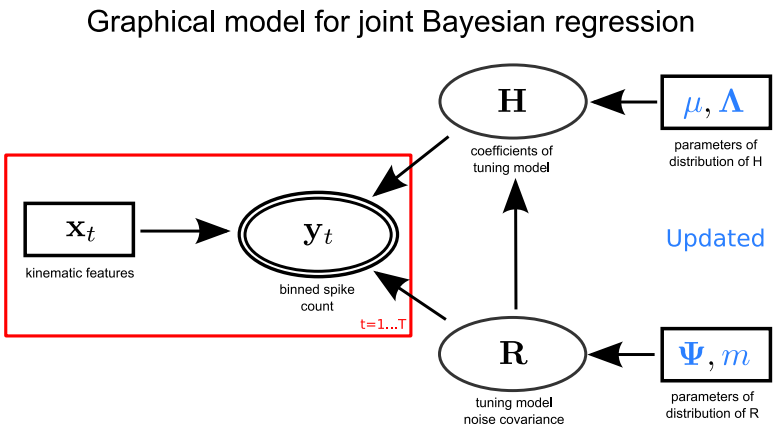

Figure 3.2: Graphical model for joint Bayesian regression. Random variables are in ellipses. Observed random variables are in double ellipses. Observed point values are in rectangles. An arrow from A to B indicates B is parameterized by A. Red rectangles indicate plates of replicated variables.

distribution and an inverse-Wishart distribution, with a shared term:

$$\mathcal{MW}^{-1}\left(\mathrm{vec}\left(\mathbf{H}\right),\mathbf{R}|\,\mathrm{vec}\left(\tilde{\mu}\right),\tilde{\mathbf{\Lambda}}^{-1},\tilde{\mathbf{\Psi}},\tilde{m}\right)=$$

$$\mathcal{N}\left(\mathrm{vec}\left(\mathbf{H}\right)|\,\mathrm{vec}\left(\tilde{\mu}\right),\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}\right)\mathcal{W}^{-1}\left(\mathbf{R}|\tilde{\mathbf{\Psi}},\tilde{m}\right) \quad (3.3)$$

$\otimes$ is the Kronecker product. $\tilde{\mu}$, an N by D matrix, is the prior mean of the coefficients. $\tilde{\mathbf{\Lambda}}$ is a D by D matrix. $\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}$ is the prior covariance of the coefficients. $\tilde{\mathbf{\Psi}}$ and $\tilde{m}$ are the N by N prior inverse scale matrix and prior degrees of freedom of the noise covariance, respectively. Note that if vec() converts a matrix into a column vector by horizontally concatenating the rows (instead of the opposite, as defined above) then transposing, the order of the Kronecker multiplicands will need to be reversed.

Rewriting the Likelihood

The log likelihood of observing a batch of $\mathbf{y}$ and $\mathbf{x}$ is:

$$\begin{aligned}
\ln P(\mathbf{Y}|\mathbf{X},\mathbf{H},\mathbf{R}) &= \ln \prod_{t=1}^{T} \mathcal{N}(\mathbf{y}_t|\mathbf{H}\mathbf{x}_t,\mathbf{R}) \\
&= \sum_{t=1}^{T} -\frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\mathbf{R}| - \frac{1}{2}\left(\mathbf{y}_t - \mathbf{H}\mathbf{x}_t\right)^T \mathbf{R}^{-1}\left(\mathbf{y}_t - \mathbf{H}\mathbf{x}_t\right) \\
&= -\frac{TN}{2}\ln(2\pi) + \frac{T}{2}\ln|\mathbf{R}^{-1}| - \frac{1}{2}\operatorname{tr}\left(\mathbf{Y}^T\mathbf{R}^{-1}\mathbf{Y}\right) \\
&\quad + \operatorname{tr}\left(\mathbf{Y}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{X}\right) - \frac{1}{2}\operatorname{tr}\left(\left(\mathbf{H}\mathbf{X}\right)^T\mathbf{R}^{-1}\left(\mathbf{H}\mathbf{X}\right)\right) \\
&= \begin{bmatrix} \operatorname{vec}\left(\mathbf{Y}\mathbf{X}^T\right) \\ -\frac{1}{2}\operatorname{vec}\left(\mathbf{X}\mathbf{X}^T\right) \\ -\frac{1}{2}\operatorname{vec}\left(\mathbf{Y}\mathbf{Y}^T\right) \\ \frac{T}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}\left(\mathbf{R}^{-1}\mathbf{H}\right) \\ \operatorname{vec}\left(\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\right) \\ \operatorname{vec}\left(\mathbf{R}^{-1}\right) \\ \ln|\mathbf{R}^{-1}| \end{bmatrix} - \frac{TN}{2}\ln(2\pi)
\end{aligned} \tag{3.4}$$

$\mathbf{X}$ is the D by T matrix of features of the quadratic model of tuning created by horizontally concatenating the vectors $\mathbf{x}_{1\ldots T}$. $\mathbf{Y}$ is the N by T matrix of binned spike counts created by horizontally concatenating the vectors $\mathbf{y}_{1\ldots T}$.

Rewriting the Prior

The log of the prior distribution can be rewritten:

$$\begin{aligned}
\ln\mathcal{MW}^{-1}\left(\mathrm{vec}\left(\mathbf{H}\right),\mathbf{R}|\,\mathrm{vec}\left(\tilde{\mu}\right),\tilde{\mathbf{\Lambda}}^{-1},\tilde{\mathbf{\Psi}},\tilde{m}\right) &= -\frac{DN}{2}\ln(2\pi)-\frac{1}{2}\ln|\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}| \\
&-\frac{1}{2}\left(\mathrm{vec}\left(\mathbf{H}\right)-\mathrm{vec}\left(\tilde{\mu}\right)\right)^{T}\left(\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}\right)^{-1}\left(\mathrm{vec}\left(\mathbf{H}\right)-\mathrm{vec}\left(\tilde{\mu}\right)\right) \\
&+\frac{\tilde{m}}{2}\ln|\tilde{\mathbf{\Psi}}|-\frac{\tilde{m}+N+1}{2}\ln|\mathbf{R}|-\frac{1}{2}\mathrm{tr}\left(\tilde{\mathbf{\Psi}}\mathbf{R}^{-1}\right)-\frac{\tilde{m}N}{2}\ln(2)-\ln\Gamma_{N}\left(\frac{\tilde{m}}{2}\right) \\
&= -\frac{DN}{2}\ln(2\pi)-\frac{N}{2}|\tilde{\mathbf{\Lambda}}^{-1}|-\frac{D}{2}\ln|\mathbf{R}|-\frac{1}{2}\mathrm{vec}\left(\mathbf{H}\right)^{T}\left(\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}\right)^{-1}\mathrm{vec}\left(\mathbf{H}\right) \\
&+\mathrm{vec}\left(\mathbf{H}\right)^{T}\left(\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}\right)^{-1}\mathrm{vec}\left(\tilde{\mu}\right)-\frac{1}{2}\mathrm{vec}\left(\tilde{\mu}\right)^{T}\left(\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}\right)^{-1}\mathrm{vec}\left(\tilde{\mu}\right) \\
&+\frac{\tilde{m}}{2}\ln|\tilde{\mathbf{\Psi}}|-\frac{\tilde{m}+N+1}{2}\ln|\mathbf{R}|-\frac{1}{2}\mathrm{tr}\left(\tilde{\mathbf{\Psi}}\mathbf{R}^{-1}\right)-\frac{\tilde{m}N}{2}\ln(2)-\ln\Gamma_{N}\left(\frac{\tilde{m}}{2}\right) \\
&= \begin{bmatrix} \mathrm{vec}\left(\tilde{\mu}\tilde{\mathbf{\Lambda}}\right) \\ -\frac{1}{2}\mathrm{vec}\left(\tilde{\mathbf{\Lambda}}\right) \\ -\frac{1}{2}\mathrm{vec}\left(\tilde{\mathbf{\Psi}}+\tilde{\mu}\tilde{\mathbf{\Lambda}}\tilde{\mu}^{T}\right) \\ \frac{\tilde{m}+N+1+D}{2} \end{bmatrix}^{T} \begin{bmatrix} \mathrm{vec}\left(\mathbf{R}^{-1}\mathbf{H}\right) \\ \mathrm{vec}\left(\mathbf{H}^{T}\mathbf{R}^{-1}\mathbf{H}\right) \\ \mathrm{vec}\left(\mathbf{R}^{-1}\right) \\ \ln|\mathbf{R}^{-1}| \end{bmatrix} \\
&-\frac{DN}{2}\ln(2\pi)-\frac{N}{2}\ln|\tilde{\mathbf{\Lambda}}^{-1}|+\frac{\tilde{m}}{2}\ln|\tilde{\mathbf{\Psi}}|-\frac{\tilde{m}N}{2}\ln(2)-\ln\Gamma_{N}\left(\frac{\tilde{m}}{2}\right)
\end{aligned} \tag{3.5}$$

$\Gamma_N()$ is the multivariate Gamma function. Note the use of a property of Kronecker products and determinants:

$$|\tilde{\mathbf{\Lambda}}^{-1}\otimes\mathbf{R}| = |\tilde{\mathbf{\Lambda}}^{-1}|^{N}|\mathbf{R}|^{D} \tag{3.6}$$

The Kronecker product is required to substitute the multiplications of matrices with multiplications of vectorized matrices.

Posterior

Using Bayes' rule, the posterior probability of the regression model parameters given the data is proportional to the likelihood of the data multiplied by the prior probability of the model parameters. The multiplication becomes an addition in log probabilities. By examining the forms of the likelihood in Equation 3.4 and prior in Equation 3.5, it is noted that the right-hand column vectors of the first dot product are the same. Thus, when adding the values in Equations 3.4 and 3.5, the dot products can be factored with respect to the right-hand column. This phenomenon is due to conjugacy. The terms outside the first dot product are normalization terms to ensure proper probability distributions [7]. The posterior is:

$$
\begin{aligned}
&\ln\mathcal{MW}^{-1}\left(\operatorname{vec}(\mathbf{H}),\mathbf{R}|\operatorname{vec}(\mu),\mathbf{\Lambda}^{-1},\mathbf{\Psi},m\right) \\
&= \begin{bmatrix} \operatorname{vec}(\mu\mathbf{\Lambda}) \\ -\frac{1}{2}\operatorname{vec}(\mathbf{\Lambda}) \\ -\frac{1}{2}\operatorname{vec}\left(\mathbf{\Psi}+\mu\mathbf{\Lambda}\mu^T\right) \\ \frac{m+N+1+D}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}(\mathbf{R}^{-1}\mathbf{H}) \\ \operatorname{vec}\left(\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\right) \\ \operatorname{vec}(\mathbf{R}^{-1}) \\ \ln|\mathbf{R}^{-1}| \end{bmatrix} + Z_1 \\
&= \ln\mathcal{MW}^{-1}\left(\operatorname{vec}(\mathbf{H}),\mathbf{R}|\operatorname{vec}(\tilde{\mu}),\tilde{\mathbf{\Lambda}}^{-1},\tilde{\mathbf{\Psi}},\tilde{m}\right) + \sum_{t=1}^{T}\ln\mathcal{N}(\mathbf{y}_t|\mathbf{x}_t\mathbf{H},\mathbf{R}) \qquad (3.7) \\
&= \begin{bmatrix} \operatorname{vec}\left(\tilde{\mu}\tilde{\mathbf{\Lambda}}\right)+\operatorname{vec}\left(\mathbf{Y}\mathbf{X}^T\right) \\ -\frac{1}{2}\operatorname{vec}\left(\tilde{\mathbf{\Lambda}}\right)-\frac{1}{2}\operatorname{vec}\left(\mathbf{X}\mathbf{X}^T\right) \\ -\frac{1}{2}\operatorname{vec}\left(\tilde{\mathbf{\Psi}}+\tilde{\mu}\tilde{\mathbf{\Lambda}}\tilde{\mu}^T\right)-\frac{1}{2}\operatorname{vec}\left(\mathbf{Y}\mathbf{Y}^T\right) \\ \frac{\tilde{m}+N+1+D}{2}+\frac{T}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}(\mathbf{R}^{-1}\mathbf{H}) \\ \operatorname{vec}\left(\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\right) \\ \operatorname{vec}(\mathbf{R}^{-1}) \\ \ln|\mathbf{R}^{-1}| \end{bmatrix} + Z_2
\end{aligned}
$$

Z_1 and Z_2 hold the normalization terms needed to ensure proper probability distributions. By rewriting the prior and likelihood functions in this manner, it can be seen that the hyper-parameters of the posterior are the sum of the hyper-parameters of the prior and some statistics of the data.

Hyper-Parameter Updates

With some algebraic manipulation, the updates for the individual hyper-parameters are:

$$\mathbf{\Lambda} \leftarrow \tilde{\mathbf{\Lambda}} + \mathbf{X}\mathbf{X}^T \tag{3.8}$$

$$\mu \leftarrow \left(\tilde{\mu}\tilde{\mathbf{\Lambda}} + \mathbf{Y}\mathbf{X}^T\right)\mathbf{\Lambda}^{-1} \tag{3.9}$$

$$\mathbf{\Psi} \leftarrow \tilde{\mathbf{\Psi}} + \tilde{\mu}\tilde{\mathbf{\Lambda}}\tilde{\mu}^T + \mathbf{Y}\mathbf{Y}^T - \mu\mathbf{\Lambda}\mu^T \tag{3.10}$$

$$m \leftarrow \tilde{m} + T \tag{3.11}$$

The Bayesian regression for the joint distribution on $\mathbf{H}$ and $\mathbf{R}$ is performed by executing the updates in Equations 3.8-3.11. This procedure is exact and fast, with the inverse in Equation 3.9 the asymptotically slowest step. Note the Cholesky decomposition can be used to perform the inverse, as $\mathbf{\Lambda}$ should be symmetric and positive-definite.

3.4.1 Drawback

The drawback of the joint distribution for $\mathbf{H}$ and $\mathbf{R}$ is that the tuning model coefficients for each neuron cannot be updated separately. The entire population shares the same $\mathbf{\Lambda}$ parameter. If some neurons need to be updated while others do not, Bayesian regression cannot be used, as updating $\mathbf{\Lambda}$ affects the future updates for the parameters of all neurons. Though the variance of coefficients for neurons can be different, the differences are entirely due to differences in $\mathbf{R}$ through the Kronecker product, as can be seen in the definition of the normal-inverse-Wishart distribution for $\mathbf{H}$ and $\mathbf{R}$ in Equation 3.3 on page 61.

Allowing a separate $\mathbf{\Lambda}_i$ hyper-parameter per neuron makes the prior no longer conjugate to the likelihood. This is because the operator between $\mathbf{\Lambda}_i^{-1}$ and $\mathbf{R}$ necessary

for Equation 3.5 on page 63 to hold can create a non-symmetric matrix if $\mathbf{\Lambda}_i$ for different neurons are different.

Importance of Flexibility

The ability to update tuning model coefficients for subsets of neurons is important because of the flexibility this allows. It may be advantageous to stop updating coefficients for neurons which are observed to be stable in tuning, thus decreasing sensitivity to noise. Temporary recording noise or recording failure may corrupt the recordings for only subsets of neurons; these events can be detected using failure detection techniques [140] and updates for these neurons can be paused.

If new neurons are detected during spike sorting, they can be added to the tuning model. The ideal way to add new neurons is to initialize their tuning coefficients to have zero means and large variances, reflecting a diffuse prior. However, with the joint model, new neurons must start with the same $\mathbf{\Lambda}$ parameter as the existing neurons, which reflects a strong prior. Initializing with large $\mathbf{\Psi}$ entries for new neurons does not address this problem, because subsequent updates for the μ entries of the new neurons depend on $\mathbf{\Lambda}$.

3.5 Regression with Factorized Distribution

Due to the limited flexibility of the normal-inverse-Wishart distribution for $\mathbf{H}$ and $\mathbf{R}$, an alternative representation for $\mathbf{H}$ and $\mathbf{R}$ is desired. However, the joint distribution for $\mathbf{H}$ and $\mathbf{R}$ which allows different precision entries for each neuron is not conjugate to the likelihood, and a slow Markov Chain Monte Carlo (MCMC) [95] procedure would be required to perform inference with it.

A fast alternative to MCMC is the variational Bayes (VB) [2, 6, 7, 48, 141] approximation approach. However, the joint distribution for $\mathbf{H}$ and $\mathbf{R}$ needs to be factorized for this approach to be applied.

3.5.1 Review of Variational Bayes

The variational Bayes approach assumes a factorization of the joint posterior of the hidden variables given the visible variables. In this case, the hidden variables are $\mathbf{H}$ and $\mathbf{R}$ and the visible variable is $\mathbf{Y}$. The factorization allows conjugacy in the links of the graphical model, which allows inference on the hidden variables to be performed analytically. The posterior is computed using fixed-point equations in a manner that can be thought of as a Bayesian generalization of the expectation-maximization [19] algorithm [6]. The fixed-point equations minimize the Kullback-Leibler (KL) divergence between the true joint posterior on the hidden variables and the factorized posterior. The algorithm is guaranteed to converge to a (local) minimum of the KL divergence [141]. Detailed description of the VB approach can be found in works by Jordan et al. [48], Attias [2], Beal [6], Winn and Bishop [141], and Bishop [7].

Benefits and Drawbacks

Variational Bayes allows much faster calculation of posteriors on hidden variables than MCMC. However, VB is subject to local optima and limited to models with conjugate relationships [7]. Usually, probabilistic models require additional factorization not already included in the model to ensure conjugacy, which makes the VB solution an approximation [7]. The KL divergence that is minimized uses the inferred posterior distribution as the reference, instead of the true joint posterior as the reference [141].

3.5.2 Factorized Model

Variational Bayesian regression (VBR) approximates the joint distribution on $\mathbf{H}$ and $\mathbf{R}$ with a factorized distribution. The model of variational Bayesian regression is

shown in graphical form in Figure 3.3 and specified below:

$$\mathbf{y}_t|\mathbf{x}_t, \mathbf{H}, \mathbf{R} \sim \mathcal{N}(\mathbf{H}\mathbf{x}_t, \mathbf{R}) \tag{3.12a}$$

$$\mathbf{H}_{(i)} \sim \mathcal{N}(\mu_i, \mathbf{\Lambda}_i^{-1}) \tag{3.12b}$$

$$\mathbf{R} \sim \mathcal{W}^{-1}(\mathbf{\Psi}, m) \tag{3.12c}$$

i is the index on the neurons. Subscripts in parentheses indicate the row number of the matrix. μ_i are the mean hyper-parameters of the tuning model coefficients for each neuron (column vectors of length D), corresponding to the rows of $\mathbf{H}$. $\mathbf{\Lambda}_i$ are the precision hyper-parameters of the tuning model coefficients for each neuron (size D by D). $\mathbf{\Psi}$ is the N by N inverse scale matrix hyper-parameter of the tuning model error covariance, and m is the degrees of freedom hyper-parameter of the tuning model error covariance.

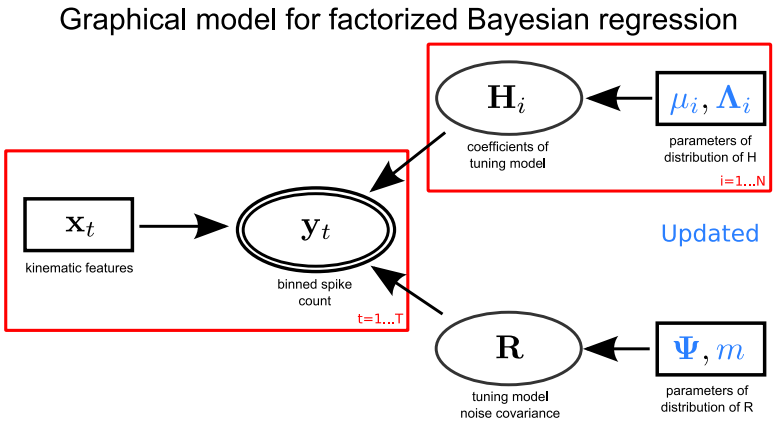

Figure 3.3: Graphical model for factorized Bayesian regression. Random variables are in ellipses. Observed random variables are in double ellipses. Observed point values are in rectangles. An arrow from A to B indicates B is parameterized by A. Red rectangles indicate plates of replicated variables.

The functions $Q(\mathbf{H}_{(i)})$ and $Q(\mathbf{R})$ are used to denote the inferred posterior distributions for $\mathbf{H}_{(i)}$ and $\mathbf{R}$, respectively. The inferred joint posterior Q is the product of

the Q functions for each variable:

$$Q = Q(\mathbf{R}) \prod_{i=1}^{N} Q(\mathbf{H}_{(i)}) \tag{3.13}$$

Despite the factorization among neurons of the tuning coefficient distributions, there still may be correlated noise among neurons which is not due to tuning to movement. Thus, a full noise covariance matrix is needed. To update the full covariance in a Bayesian manner, the inverse-Wishart distribution is used instead of a set of Gamma distributions.

3.5.3 Variational Updates

To understand how the VB approach minimizes the KL divergence between Q, the inferred joint posterior, and the true joint posterior, note the decomposition of the log marginal probability on $\mathbf{X}$ and $\mathbf{Y}$ into a *lower bound* $\mathcal{L}(Q)$ and the KL divergence:

$$\ln P(\mathbf{X}, \mathbf{Y}) = \mathcal{L}(Q) + \mathrm{KL}(Q||P) \tag{3.14}$$

$$\mathcal{L}(Q) = \int_{\mathbf{H},\mathbf{R}} Q(\mathbf{H}) Q(\mathbf{R}) \ln \frac{P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y})}{Q(\mathbf{H}) Q(\mathbf{R})} \tag{3.15}$$

$$\mathrm{KL}(Q||P) = \int_{\mathbf{H},\mathbf{R}} Q(\mathbf{H}) Q(\mathbf{R}) \ln \frac{P(\mathbf{H}, \mathbf{R}|\mathbf{X}, \mathbf{Y})}{Q(\mathbf{H}) Q(\mathbf{R})} \tag{3.16}$$

$Q(\mathbf{H})$ is shorthand for $\prod_{i=1}^{N} Q(\mathbf{H}_{(i)})$. P is shorthand for the true joint posterior on the hidden variables $P(\mathbf{H}, \mathbf{R}|\mathbf{X}, \mathbf{Y})$. Since the KL-divergence is non-negative, $\mathcal{L}(Q)$ is a lower bound on the log marginal probability of the observed variables. Since $\ln P(\mathbf{X}, \mathbf{Y})$ is fixed given the kinematic features and neuronal activity, by maximizing the lower bound $\mathcal{L}(Q)$, the KL divergence is minimized.

Fixed-Point Procedure

It can be shown [141] that $\mathcal{L}(Q)$ is maximized when:

$$Q(\mathbf{H}_{(i)}) = \frac{1}{Z_i} \exp\langle \ln P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y}) \rangle_{\sim Q(\mathbf{H}_{(i)})} \tag{3.17}$$

$$Q(\mathbf{R}) = \frac{1}{Z_R} \exp\langle \ln P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y}) \rangle_{\sim Q(\mathbf{R})} \tag{3.18}$$

The Z_i and Z_R terms are the normalization factors needed to make the Q functions valid probability distributions. The expectations are taken with respect to the other Q functions, as indicated by the subscripts.

The parameters of $Q(\mathbf{H}_{(i)})$ and $Q(\mathbf{R})$ which satisfy the above equations can be approximated using a fixed-point procedure [7, 141]. The procedure updates the parameters for each Q function, in turn, until convergence, which can be detected by lack of change in the lower bound $\mathcal{L}(Q)$. Each update sets the parameters of a Q function so that the Q function satisfies the criteria shown in Equations 3.17 or 3.18 for the current values of the other Q functions.

The update equations for the parameters of $Q(\mathbf{H}_{(i)})$ and $Q(\mathbf{R})$ are derived from Equations 3.17 and 3.18 by factorizing the Bayesian network of the regression model and performing algebraic manipulations similar to those in Equations 3.5 and 3.7. See Appendix A on page 182 for the derivations.

The update equations are grouped into the updates for the parameters of $\mathbf{H}_{(i)}$ and the updates for the parameters of $\mathbf{R}$. An overview of the variational Bayesian regression procedure is shown in Figure 3.4 on the following page.

H update

$$\mathbf{\Lambda}_i \leftarrow \tilde{\mathbf{\Lambda}}_i + \mathbf{X}\mathbf{X}^T \langle 1/\mathbf{R}_{(i,i)} \rangle \qquad i = 1...N \tag{3.19}$$

$$\mu_i \leftarrow \mathbf{\Lambda}_i^{-1} (\tilde{\mathbf{\Lambda}}_i \tilde{\mu}_i + \mathbf{X}\mathbf{Y}_{(i)}^T \langle 1/\mathbf{R}_{(i,i)} \rangle) \qquad i = 1...N \tag{3.20}$$

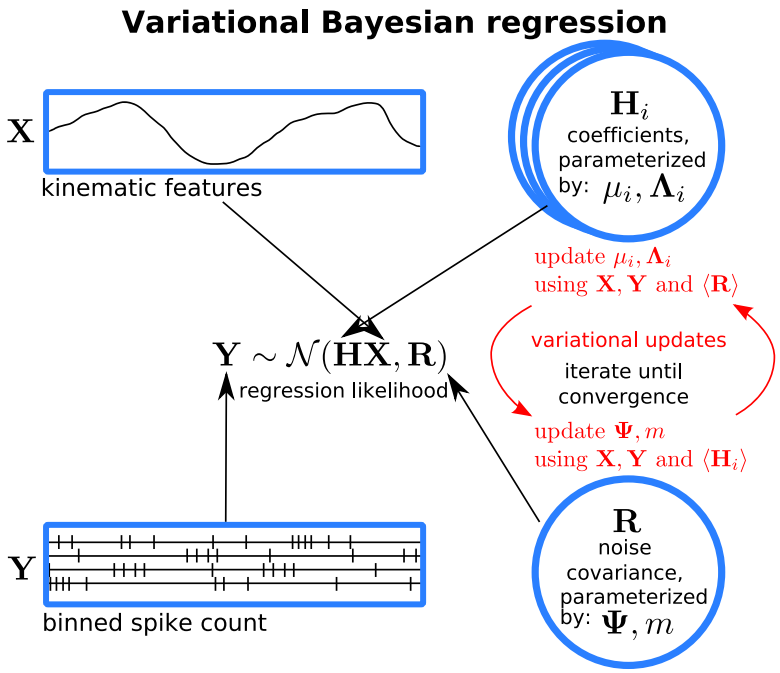

Figure 3.4: Variational Bayesian regression overview

$\tilde{\mathbf{\Lambda}}_i$ and $\tilde{\mu}_i$ indicate prior values of the hyper-parameters $\mathbf{\Lambda}_i$ and μ_i. $\mathbf{Y}_{(i)}$ is the length T row vector of binned spike counts for neuron i. $\langle 1/\mathbf{R}_{(i,i)} \rangle$ is the (i,i) entry of the expectation of $\mathbf{R}^{-1}$:

$$\langle 1/\mathbf{R}_{(i,i)} \rangle = (m\mathbf{\Psi}^{-1})_{(i,i)} \tag{3.21}$$

R update

$$m \leftarrow \tilde{m} + T \tag{3.22}$$

$$\mathbf{\Psi} \leftarrow \tilde{\mathbf{\Psi}} + \langle (\mathbf{Y} - \mathbf{HX})(\mathbf{Y} - \mathbf{HX})^T \rangle \tag{3.23}$$

$\tilde{m}$ and $\tilde{\mathbf{\Psi}}$ indicate the prior values of the hyper-parameters m and $\mathbf{\Psi}$. The expectation of the quadratic term in the update for $\mathbf{\Psi}$ complicates the computation. Writing out the contribution of the covariance of $\mathbf{H}$, the $\mathbf{\Psi}$ update is:

$$\mathbf{\Psi} \leftarrow \tilde{\mathbf{\Psi}} + (\mathbf{Y} - \langle \mathbf{H} \rangle \mathbf{X})(\mathbf{Y} - \langle \mathbf{H} \rangle \mathbf{X})^T + \mathbf{W} \tag{3.24}$$

$$\mathbf{W}_{(i,j)} = \begin{cases} \text{tr}(\mathbf{X}^T \mathbf{\Lambda}_i^{-1} \mathbf{X}), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{3.25}$$

$\mathbf{W}$ is an N by N temporary matrix, $\mathbf{W}_{(i,j)}$ is the (i,j) entry of $\mathbf{W}$, and $\langle \mathbf{H} \rangle$ is the matrix formed by the vertical stacking of the row vectors μ_i^T.

The prior values $\tilde{\mu}, \tilde{\mathbf{\Lambda}}, \tilde{\mathbf{\Psi}}$, and $\tilde{m}$ are the values from the previous VBR update or the initial parameter fit. At the beginning of the fixed point iterations, μ, $\mathbf{\Lambda}$, $\mathbf{\Psi}$, and m are initialized to their prior values.

3.5.4 Lower Bound Calculation

The lower bound $\mathcal{L}(Q)$ is calculated to detect convergence. It also serves as a useful diagnostic tool for the implementation, since the lower bound should not decrease during the update of any random variable. The equation for the lower bound from Equation 3.14 can be rewritten as:

$$\mathcal{L}(Q) = \int_{\mathbf{H},\mathbf{R}} Q(\mathbf{H}) Q(\mathbf{R}) \ln \frac{P(\mathbf{H}, \mathbf{R}, x, y)}{Q(\mathbf{H}) Q(\mathbf{R})} \tag{3.26}$$

$$= \langle \ln P(\mathbf{H}, \mathbf{R}, x, y) \rangle - \langle \ln Q(\mathbf{H}) + \ln Q(\mathbf{R}) \rangle \tag{3.27}$$

The expectations are taken with respect to $Q(\mathbf{H})$ and $Q(\mathbf{R})$. By factorizing the Bayesian network of the model [141], the lower bound can be computed as the sum of three terms $\mathcal{L}_H$, $\mathcal{L}_R$, and $\mathcal{L}_y$:

$$\mathcal{L}_H = \langle \ln P(\mathbf{H} | \tilde{\mu}_{1\ldots N}, \tilde{\mathbf{\Lambda}}_{1\ldots N}) \rangle - \langle \ln Q(\mathbf{H}) \rangle \tag{3.28}$$

$$\mathcal{L}_R = \langle \ln P(\mathbf{R} | \tilde{\mathbf{\Psi}}, \tilde{m}) \rangle - \langle \ln Q(\mathbf{R}) \rangle \tag{3.29}$$

$$\mathcal{L}_y = \langle \ln P(\mathbf{Y} | \mathbf{X}, \mathbf{H}, \mathbf{R}) \rangle \tag{3.30}$$

After substituting the probability density functions, re-arranging, and taking expectations, $\mathcal{L}_H$, $\mathcal{L}_R$, and $\mathcal{L}_y$ can be written as:

$$\begin{aligned}\mathcal{L}_H =& \sum_{i=1}^{N} \begin{bmatrix} \tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i - \mathbf{\Lambda}_i\mu_i \\ \frac{1}{2}\operatorname{vec}(\mathbf{\Lambda}_i - \tilde{\mathbf{\Lambda}}_i) \end{bmatrix}^T \begin{bmatrix} \mu_i \\ \operatorname{vec}(\mathbf{\Lambda}_i^{-1} + \mu_i\mu_i^T) \end{bmatrix} \\ &+ \frac{1}{2}\left(\mu_i^T\mathbf{\Lambda}_i\mu_i - \tilde{\mu}_i^T\tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i + \ln|\tilde{\mathbf{\Lambda}}_i| - \ln|\mathbf{\Lambda}_i|\right)\end{aligned} \tag{3.31}$$

$$\begin{aligned}\mathcal{L}_R =& -\frac{1}{2}(m-N-1)\left[\operatorname{tr}(\tilde{\mathbf{\Psi}}\mathbf{\Psi}^{-1}) - N\right] - \frac{\tilde{m}-m}{2}\langle\ln|\mathbf{R}|\rangle \\ &+ \frac{1}{2}\left(\tilde{m}\ln|\tilde{\mathbf{\Psi}}| - m\ln|\mathbf{\Psi}|\right) - \frac{1}{2}N(\ln 2)(\tilde{m}-m) \\ &- \sum_{i=1}^{N}\ln\Gamma\left(\frac{\tilde{m}+1-i}{2}\right) - \ln\Gamma\left(\frac{m+1-i}{2}\right)\end{aligned} \tag{3.32}$$

$$\begin{aligned}\mathcal{L}_y =& \sum_{t=1}^{T} \begin{bmatrix} \langle\mathbf{R}^{-1}\rangle\langle\mathbf{H}\rangle\mathbf{X}_{(t)} \\ -\frac{1}{2}\operatorname{vec}(\langle\mathbf{R}^{-1}\rangle) \end{bmatrix}^T \begin{bmatrix} \mathbf{Y}_{(t)} \\ \operatorname{vec}(\mathbf{Y}_{(t)}\mathbf{Y}_{(t)}^T) \end{bmatrix} \\ &+ \frac{1}{2}\left(\langle\ln|\mathbf{R}^{-1}|\rangle - \langle(\mathbf{H}\mathbf{X}_{(t)})^T\mathbf{R}^{-1}(\mathbf{H}\mathbf{X}_{(t)})\rangle - N\ln 2\pi\right)\end{aligned} \tag{3.33}$$

Where

$$\langle\ln|\mathbf{R}^{-1}|\rangle = N\ln 2 + \ln|\mathbf{\Psi}^{-1}| + \sum_{i=1}^{N}\psi\left(\frac{m+1-i}{2}\right) \tag{3.34}$$

$$\langle\ln|\mathbf{R}|\rangle = -\langle\ln|\mathbf{R}^{-1}|\rangle \tag{3.35}$$

$$\langle(\mathbf{H}\mathbf{X}_{(t)})^T\mathbf{R}^{-1}(\mathbf{H}\mathbf{X}_{(t)})\rangle = (\langle\mathbf{H}\rangle\mathbf{X}_{(t)})\langle\mathbf{R}^{-1}\rangle(\langle\mathbf{H}\rangle\mathbf{X}_{(t)})^T + \operatorname{tr}(\mathbf{\Sigma}_t\langle\mathbf{R}^{-1}\rangle) \tag{3.36a}$$

$$(\mathbf{\Sigma}_t)_{(i,j)} = \begin{cases} \mathbf{X}_{(t)}^T\mathbf{\Lambda}_i^{-1}\mathbf{X}_{(t)}, & \text{if } i=j \\ 0, & \text{otherwise} \end{cases} \tag{3.36b}$$

$$\langle\mathbf{R}^{-1}\rangle = m\mathbf{\Psi}^{-1} \tag{3.37}$$

$\mathbf{\Sigma}_t$ is an N by N temporary matrix. $\Gamma()$ is the Gamma function, and $\psi()$ is the digamma function.

3.6 Parameter Fitting with Bayesian Regression

As the Bayesian regression updates require prior distributions on the tuning model coefficients and noise covariance, ordinary least squares and ridge regression are unsuitable for performing the initial fit. Instead, the initial neuronal tuning model is fit using Bayesian regression (either joint or factorized, depending on the type used in the update) with specific priors chosen to act as regularization.

Initial fits were performed using hand position recorded from the joystick. The problem of fitting models in the absence of joystick movements was not addressed in this work, since it has been addressed by numerous studies [22, 120, 130, 134, 135].

Regularization Prior

For all experiments in Section 4.3, the following priors were used for initial parameter fitting:

$$\tilde{\mu}_i = \mathbf{0} \tag{3.38}$$

$$\tilde{\mathbf{\Lambda}}_i^{-1} = \lambda \mathbf{I} \tag{3.39}$$

$$\tilde{\mathbf{\Psi}} = \mathbf{I} \tag{3.40}$$

$$\tilde{m} = N + 2 \tag{3.41}$$

$\mathbf{0}$ is the zero matrix and $\mathbf{I}$ is the D by D identity matrix. For joint distribution Bayesian regression, there is only one $\tilde{\mu}$, which is assigned the N by D zero matrix, and there is only one $\tilde{\mathbf{\Lambda}}$. For factorized distribution variational Bayesian regression, the $\tilde{\mu}_i$ and $\tilde{\mathbf{\Lambda}}_i$ for each neuron must be initialized and the zero matrix is D by 1. The prior variance λ of the coefficients acts in the same way as the ridge regression λ_H parameter for Equation 2.34 on page 39. For off-line experiments, the value for λ was chosen by optimizing for the reconstruction accuracy. For closed-loop experiments, the optimal λ from off-line reconstructions was used.

Fitting the Movement Model

Bayesian regression with similar priors was also used to fit the movement model coefficients and the movement model noise covariance. However, $\lambda = 10^{-8}$ and $\tilde{m} = d + 2$.

Baseline Firing Rate

In the presentation of the n-th order UKF in Chapter 2, the observations were mean-subtracted before being used by the filter. This mean acts as a baseline firing rate parameter for each neuron. Studies have found changes in baseline firing rates of neurons over time [12, 31]. Thus, these means should be updated. Two methods for updating the baseline firing rate parameters were tested in off-line reconstructions: adding a bias term to the quadratic model of tuning, omitting the mean subtraction, and updating the bias coefficient as any other tuning coefficient, or setting the baseline firing rate to the sample mean in the self-training data window. The two approaches produced similar results. For off-line reconstructions, the results from using the former approach are reported in Chapter 4. For closed-loop experiments, the latter method was used.

3.7 Transition Model for Parameters

A transition model for the tuning model parameters is required to make them identifiable and to account for their assumed time-varying nature.

Transition Model for Coefficients

For the tuning model coefficients $\mathbf{H}$, the transition model is the identity transition plus normally distributed noise, i.e. Brownian motion. The effect of this transition model is to add the covariance of the noise to the covariance of the distributions for the coefficients. This amount of noise is the *model drift* parameter δ. δ is added to

the diagonal of the tuning coefficient covariance matrix:

$$\mathbf{\Lambda}_i^{-1} \leftarrow \mathbf{\Lambda}_i^{-1} + \delta \mathbf{I} \tag{3.42}$$

$\mathbf{I}$ is the D by D identity matrix. For joint distribution Bayesian regression, there is only one $\mathbf{\Lambda}$. For factorized distribution variational Bayesian regression, the $\mathbf{\Lambda}_i$ for each neuron must be updated in this fashion.

Transition Model for Noise Covariance

The uncertainty of the observation noise covariance $\mathbf{R}$ is modeled with an inverse-Wishart distribution. There is a straight-forward way to increase the variance of an inverse-Wishart distribution without changing the mean: the degrees of freedom hyper-parameter m is reduced by a factor and the inverse scale matrix hyper-parameter $\mathbf{\Psi}$ is reduced proportionally. Two rules for applying this change are considered.

In the first rule, the hyper-parameters $\mathbf{\Psi}$ and m are adjusted if the degrees of freedom m exceed a threshold called the *degrees of freedom cap*, $\hat{m}$. The adjustment enforces that the threshold $\hat{m}$ is not exceeded:

$$\mathbf{\Psi} \leftarrow \frac{\hat{m}}{m} \mathbf{\Psi} \tag{3.43}$$

$$m \leftarrow \hat{m} \tag{3.44}$$

The multiplication is applied per element. This rule enforces a lower bound on the variance of the distribution for the noise covariance matrix.

The second rule is to apply a multiplier $0 < \gamma \leq 1$ at each transition:

$$m \leftarrow \gamma m \tag{3.45}$$

$$\mathbf{\Psi} \leftarrow \gamma \mathbf{\Psi} \tag{3.46}$$

Comparison of Methods

The multiplier rule acts on every transition, and the size of the multiplier can be adjusted for different update frequencies. The two rules can produce the same asymptotic behavior given appropriate settings. However, the cap rule allows more data to contribute to the inference on the noise covariance during the transient phase, at the cost of assuming no changes in the noise covariance during the transient phase. The multiplier rule assumes changes are on-going and can be applicable to updates which occur at varying frequencies.

In off-line experiments, the cap rule produced more accurate reconstructions, likely due to its usage of more data during the transient phase. Thus, the cap rule is used for the off-line and closed-loop experiments presented in Chapter 4.

Chapter 4

Experiments on Non-Linear Adaptive Decoding

This chapter presents synthetic experiments, off-line reconstructions, and closed-loop brain-control experiments which demonstrate the utility of the n-th order unscented Kalman filter with Bayesian regression self-training described in Chapters 2 and 3. The experiments used data from 3 Rhesus macaque monkeys cortically implanted with micro-wire electrodes.

This chapter is organized as follows. First, experimental methods are described. Second, experiments which demonstrate the improved accuracy of the n-th order unscented Kalman filter versus previous algorithms are presented. Third, experiments which demonstrate the ability of Bayesian regression self-training to adapt UKF parameters to changes in tuning are presented.

4.1 Experimental Methods

4.1.1 Implants

All surgical and experimental procedures conformed to the National Research Council's Guide for the Care and Use of Laboratory Animals (1996) and were approved by the Duke University Animal Care and Use Committee. Three Rhesus monkeys (Macaca

mulatta) were implanted with micro-wire electrode arrays of different geometries in multiple areas related to movement and sensation (Figure 4.1 on the following page). The monkeys were trained to perform behavioral tasks on a computer by controlling a cursor either with a joystick which they manipulated with their hands, or through decoding algorithms which operated on their neuronal activity.

Monkeys C and G

Monkey C, which used its left hand to manipulate the joystick, was implanted with four 32-micro-wire arrays in arm and hand portions of primary motor cortex (M1), dorsal premotor cortex (PMd), posterior parietal cortex (PP), and supplementary motor area (SMA) in the right hemisphere. Monkey G, which used its right hand to manipulate the joystick, was implanted with six 32-micro-wire arrays in arm and hand portions of M1, primary somatosensory cortex (S1), and PMd of both hemispheres.

In the arrays for these two monkeys, electrodes were grouped into 16 pairs. The separation between adjacent pairs was 1 mm. Each pair consisted of two micro-wire electrodes placed tightly together with one electrode 300 micron longer than the other. The longer electrode in each pair was equal or larger in diameter.

Monkey C was implanted with stainless steel and tungsten micro-wire electrodes of 46 and 51 micron diameter in areas SMA and M1 and tungsten micro-wire electrodes of 51 micron diameter in areas PMd and PP. Monkey G was implanted with stainless steel micro-wire electrodes of 40 and 63 micron diameter (Figure 4.1, left).

Monkey M

Monkey M, which used its left hand to manipulate the joystick, was implanted with four 96-micro-wire arrays in arm and leg portions of M1 and S1 bilaterally. Only recordings from the 96 electrodes recording from right hemisphere arm areas were used in the experiments presented here. In the arrays for monkey M, electrodes were

grouped into two 4-by-4 uniform grids of 16 triplets. The separation between adjacent triplets was 1 mm within each grid and 2 mm across the gap separating the grids. Each triplet consisted of three micro-wire electrodes placed tightly together with 300 micron differences in length. The electrodes were of various diameters (approximately 50 micron) and constructed from stainless steel. One array of 96 electrodes was placed over the arm portions of M1 and S1 of the right hemisphere so that the gap between the grids spanning the central sulcus (Figure 4.1, right).

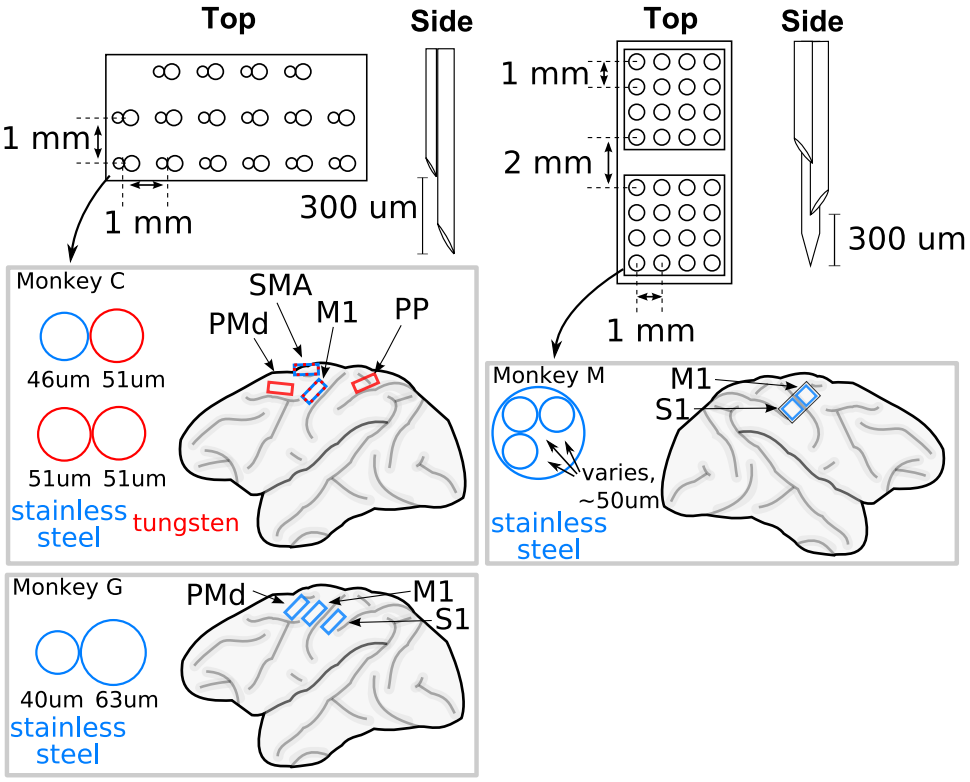

Figure 4.1: Micro-wire electrode array geometry and placement. Width of micro-wires are specified by diameter, in microns. Only the placement of arrays from which data was used are shown.

Spike Acquisition

The arrays with the best quality of neuronal signals were selected, and not all arrays were utilized in every session. Extracellular neuronal signals were amplified,

digitized, and high-pass filtered using Multichannel Acquisition Processors (Plexon, Inc.). Neuronal action potentials were discriminated by thresholding and sorted on-line by matching to waveform templates. These templates were set by human experts (not the author) using Plexon spike-sorting software or produced by custom-built automatic spike-sorting software [40]. Single and multi-units were not treated differently for prediction purposes.

Automatic Spike-Sorting

The custom spike-sorting software [40] clusters waveforms using a modified expectation-maximization [19] algorithm. Features for the clustering procedure were created by taking the three largest principle components of the time-derivatives of the waveforms. Noise was rejected by matching against templates for typical noise patterns. Spurious clusters were removed by thresholding on various criteria. Multiple re-starts of expectation-maximization were performed, and a custom metric was designed to pick amongst clusterings. This metric preferred clusterings which had tightly-grouped waveforms in each cluster.

4.1.2 Experimental Setup

During the experimental sessions, each monkey sat in a primate chair (Figure 4.2 on the next page). Their heads were unrestrained, and the recording system was connected to the implants using light, flexible wires. A two degrees-of-freedom (left-right and forward-backwards) analog joystick was mounted vertically at the monkey's waist level. The joystick was 30 cm in length and had a maximum deflection of 12 cm. The monkeys were trained to manipulate the joystick with their hands. Monkey C and monkey M manipulated the joystick with the left hand, and monkey G manipulated with the right hand. An electrical-resistance-based touch sensor or an optical touch sensor on the joystick handle measured whether the monkey was holding the joystick.

An LCD projector projected white visual stimuli on a black background on to a screen mounted 1.5 m in front of the monkeys (Figure 4.2).

Using the joystick, monkeys moved a round cursor, defined by a white ring 1.6 cm in diameter. Forward, backward, rightward, and leftward movements of the joystick translated to upward, downward, rightward, and leftward movements of the cursor, respectively. The joystick to cursor gain varied between 3.2 and 6.4, depending on session, i.e. a 1 cm movement of the joystick translated into a 3.2 to 6.4 cm movement of the cursor. Targets were defined by white rings 16 to 20.8 cm in diameter on the screen. The median speeds at which monkeys C and G moved the joystick were approximately 3.5 to 5.5 cm/s, depending on the session. The median speed at which monkey M moved the joystick was approximately 0.5 to 1.1 cm/s.

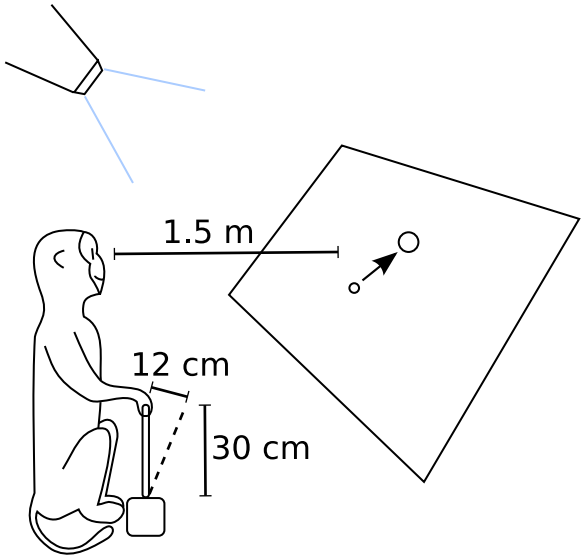

Figure 4.2: Primate experiment setup. Rhesus monkeys are seated in a primate chair facing a screen onto which stimuli are projected. Monkeys control a cursor on the screen using either a hand-held joystick or the brain-machine interface.

4.1.3 Behavioral Tasks

Each behavioral task required placing the computer cursor over the target using the joystick or decoding algorithm. The monkeys performed three tasks: center-out task, pursuit task with Lissajous trajectories, and pursuit task with point-to-point trajectories (Figure 4.3 on the following page).

Center-Out Task

The center-out task (Figure 4.3A) used stationary targets that appeared at randomly chosen points on a fixed-radius ring around the center of the screen. At the beginning of a trial, the monkey had to hold the cursor at the stationary center target at the screen center. After the center target disappeared and a peripheral target appeared, the monkey had to move the cursor to the peripheral target and keep the cursor inside the target until it received a juice reward, which ended the trial. The trials in which the monkey failed to put the cursor over the target or failed to fulfill the hold requirement were error trials and not rewarded. The inter-trial interval that followed a successful trial was 500 ms. The inter-trial interval after an error trial was 700 to 1000 ms. Hold times varied per session from 350 to 1050 ms. After a trial was finished, the center target appeared again to start the next trial. Data collected during the center-out task were treated as a continuous stream and not segmented by trial or movement onset for the analysis reported here.

Pursuit Task with Lissajous Trajectories

The pursuit task with Lissajous trajectories (Figure 4.3B) used a continuously moving target which followed a Lissajous curve:

$$x = A\sin(avt + \delta) \tag{4.1a}$$

$$y = B\sin(bvt) \tag{4.1b}$$

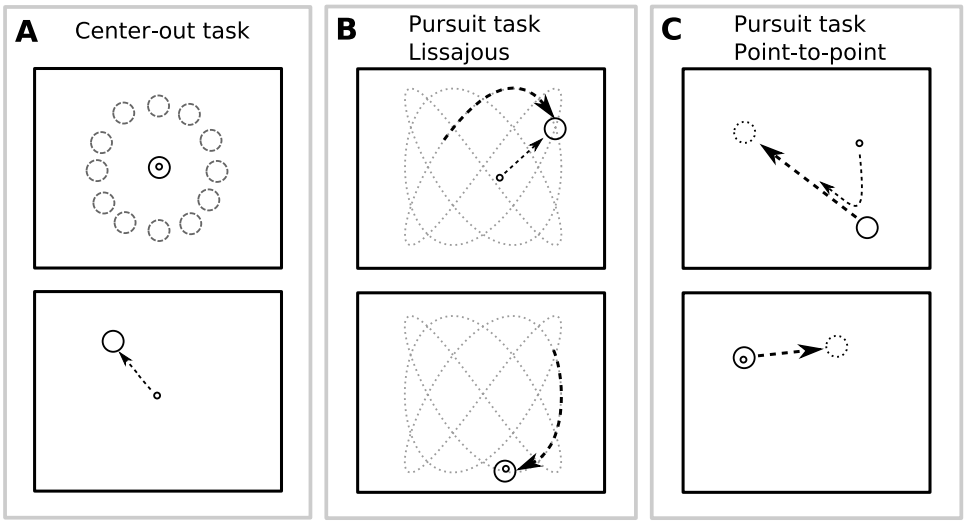

Figure 4.3: Behavioral tasks. A: After holding the cursor at the screen center, the monkeys moved it to a peripheral target that appeared at a random angle and fixed radius. B: Monkeys kept the cursor inside a continuously moving target whose trajectory was a Lissajous curve. C: Similar to B, except that the target trajectory was generated by moving between randomly generated points under velocity and acceleration limits.

x and y are the x- and y-axis coordinates and t is time in seconds. The parameter values $a = 3$, $b = 4$, $v \in \{0.1, 0.15, 0.2\}$ Hz, $\delta = 0.5\pi$, and $A = B = 22.4$ cm (in screen scale) were used. The temporal frequency was different for the x- and y-axes, making the two coordinates uncorrelated. The monkey had to keep the cursor within the moving target to receive periodic juice rewards. Rewards were larger when the cursor was closer to the center of the target to motivate the monkey to pursue the target accurately.

Pursuit Task with Point-to-Point Trajectories

The pursuit task with point-to-point trajectories (Figure 4.3C) used a continuously moving target which moved from one randomly generated point to another. The points were drawn from a uniform distribution on the 76.8 cm wide by 57.6 cm tall (in screen scale) workspace. The points only determined the target trajectory and

were not visible to the monkey. The target's acceleration (≤ 3.2 cm/sec^2, screen scale) and velocity (≤ 9.6 cm/sec, screen scale) were limited to create smooth trajectories. Rewards were larger when the cursor was closer to the center of the target to motivate the monkey to pursue the target accurately.

4.1.4 Data Preprocessing

For all algorithms, spike counts were calculated in 100 ms non-overlapping bins to estimate the instantaneous firing rate. Joystick position was recorded at 1 KHz and down-sampled to 10 Hz to match the binning rate. Velocity was calculated from position by two-point digital differentiation. Position and velocity data had their means subtracted, except in the off-line reconstructions for testing Bayesian regression self-training (see Section 3.6 on page 74). Spike counts had their means subtracted for the Kalman-based filters, except in the reconstructions for testing Bayesian regression self-training.

Data recorded while the monkey did not hold the joystick or while the joystick was stationary for longer than k seconds were disregarded. k was 2 or 3 for different sessions.

Off-line analysis was conducted using MATLAB software (MathWorks, Inc). Real-time decoders were implemented as part of a custom-built BMI software suite running on a workstation with either an Intel Xeon 2.2 Ghz processor or an Intel Core i7 2.66 Ghz processor.

4.1.5 Analysis Procedure

To quantify algorithm accuracy in off-line reconstructions, estimated hand trajectories were compared to actual hand trajectories as measured by the joystick. To quantify accuracy in on-line, closed-loop experiments, the trajectory of the cursor controlled by the monkey using the algorithm was compared to the target trajectory (only pursuit

tasks were used in closed-loop experiments). Two metrics were used: the signal-to-noise ratio (SNR) and the correlation coefficient (CC), also known as Pearson's r.

To aggregate results, mean SNR and CC among cross-validation folds, if any, and between the x- and y-axis predictions were computed. Where shown, standard error of the mean was calculated using each fold and axis as an observation. To test for significant effects, two-sided, paired sign tests were used with an $\alpha = 0.05$ significance level.

Signal-to-Noise Ratio

SNR, in decibels (dB), was calculated using:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \cdot \log_{10}\left(\frac{\mathrm{VAR}}{\mathrm{MSE}}\right) \tag{4.2}$$

VAR is the sample variance of the desired values (hand or target trajectory), and MSE is the mean squared-error of the predicted values (predicted hand or cursor trajectory) from the desired values. Position, velocity, and the x- and y-axes were evaluated separately. When SNR was calculated in segments of a session, the same signal variance (numerator) from the entire session was used for the SNR calculation on each segment, to allow comparison among segments.

The equivalent MSE reduction can be found for SNR improvements using this equation:

$$\text{MSE reduction } (\%) = 100 \cdot \left(1 - \frac{1}{10^{\mathrm{SNR}_{\mathrm{dB}}\text{ improvement}/10}}\right) \tag{4.3}$$

For convenience, Table 4.1 on the following page shows example conversions calculated using the above equation.

In general, SNR and CC are not comparable because CC is scale and translation invariant. There is an approximate relationship between SNR and CC under two

Table 4.1: Equivalent MSE reductions for various SNR improvements

SNR improvement (dB)	MSE reduction %
0.01	0.23
0.05	1.14
0.10	2.28
0.20	4.50
0.30	6.67
0.40	8.80
0.50	10.87
0.75	15.86
1.00	20.57
1.50	29.21
2.00	36.90

assumptions. The assumptions are that the mean of the desired and predicted signals are both zero and the standard deviations of the desired and predicted signals are equal.

$$\mathrm{SNR}_{\mathrm{dB}} \approx 10 \cdot \log_{10}\left(\frac{1}{2 - 2\mathrm{CC}}\right) \tag{4.4}$$

The approximation is due to the substitution of the population covariance for the sample covariance. The inverse relationship is provided below for convenience:

$$\mathrm{CC} \approx 1 - \frac{1}{2 \cdot 10^{\mathrm{SNR}_{\mathrm{dB}}/10}} \tag{4.5}$$

Table 4.2 on the next page shows example conversions calculated using the above equation.

The signal-to-noise ratio can be interpreted as the inverse of the normalized mean squared-error, where the normalization factor is the power of the desired signal. SNR is widely used in engineering and has been previously used to measure BMI decoding accuracy [61, 99]. The SNR is unit-less and comparable across experimental setups. This contrasts with the mean squared-error, which is usually incomparable

Table 4.2: Equivalent CC for various values of SNR. These values are calculated using the assumptions that the mean of the desired and predicted signals are both zero and their standard deviations are equal. These values are approximate due to the substitution of the population covariance for the sample covariance.

SNR (dB)	CC	CC^2 (i.e. r^2)
0.0	0.500	0.250
1.0	0.603	0.363
2.0	0.685	0.469
3.0	0.749	0.562
4.0	0.801	0.642
5.0	0.842	0.709
6.0	0.874	0.765
7.0	0.900	0.810
8.0	0.921	0.848
10.0	0.950	0.902
15.0	0.984	0.969

between studies due to differences in movement magnitudes. In this respect the SNR is similar to the CC. However, the SNR is sensitive to translation and scale, unlike the CC. This gives the SNR an advantage as a metric because the CC will not detect translation and scaling errors. Furthermore, since the CC saturates at 1, its scale is compressed as it approaches 1, making similar increments at smaller values and higher values incomparable. Despite its disadvantages, the CC is reported in the algorithm comparison results, since many prior studies report CC.

4.2 Non-Linear Decoding

4.2.1 Model Validation

The predictive accuracy of the quadratic tuning model of the n-th order unscented Kalman filter was measured on previously recorded sessions. Firing rates of single neurons were predicted from hand position and velocity using the quadratic model of tuning with $n = 1$ and $n = 10$ taps.

Linear Model for Comparison

For comparison, predictions were also made using the linear model of tuning described by Equation 2.6 on page 21. This model relates 1 bin of spike counts with 1 tap of kinematics, which were measured at the end time of the bin.

Procedure

The parameters of the models were fit with linear regression using the Moore-Penrose pseudoinverse, which is equivalent to ridge regression with the parameter λ approaching 0. Ten-fold cross-validation was used to test predictive accuracy. Results are reported as signal-to-noise ratio (SNR), where the desired signal is the recorded binned spike count, and correlation coefficient (CC).

Data

Neuronal data came from 16 sessions collected with monkeys C (6 sessions) and G (10). The sessions were between 9 and 25 minutes in length. The monkeys performed either the center-out task (8) or the pursuit task with Lissajous trajectories (8) by controlling the cursor with their hand. The number of neurons in each session ranged between 94 and 240 (mean 142).

Results

The $n = 1$ tap quadratic model of tuning (SNR $= 0.03 \pm 0.29$ dB, CC $= 0.10 \pm 0.09$; mean $\pm$ standard deviation) was more predictive ($p < 0.001$, two-sided, paired sign-test, 2273 observations) than the linear model (SNR $= 0.01 \pm 0.27$ dB, CC $= 0.07 \pm 0.08$). 1753 out of 2273 units (approximately 77%) were better predicted using the quadratic model of tuning. The $n = 10$ tap quadratic model (SNR $= 0.05 \pm 0.32$ dB, CC $= 0.11 \pm 0.10$) was more predictive ($p < 0.001$) than the $n = 1$ tap quadratic model. About 900 or approximately 40% of units were better predicted.

Illustration

The better fits made using the quadratic model of tuning can be seen in the contour plots in Figure 4.4 on the next page. The plots show the tuning to position and velocity of 8 representative neurons and parameter fits using the linear and quadratic ($n = 1$) models. The x and y coordinates in the plots indicate x- and y-axis positions or velocities. The brightness of the shading indicates the predicted firing rate in spikes per second (Figure 4.4, left 2 columns) and true firing rate (Figure 4.4, right-most column). For clarity, the fits to velocity (Figure 4.4, top 4 rows) and position (Figure 4.4, bottom 4 rows) are shown separately. The right-most column of Figure 4.4 shows the actual firing rate estimated on a 50 by 50 grid using Gaussian kernel smoothing. The axes where chosen to span plus and minus 3 standard deviations of the position or velocity values observed during the experimental session. The plots were made square by using the smaller of the standard deviations for x and y. The kernel width was one standard deviation of the observed values (smaller of the standard deviations for x and y).

For velocity tuning (Figure 4.4, top 4 rows), the quadratic model captures the low-center, high-surround tuning pattern seen in many neurons and in prior work [66,148], while the linear model cannot capture this pattern because it is restricted to fitting

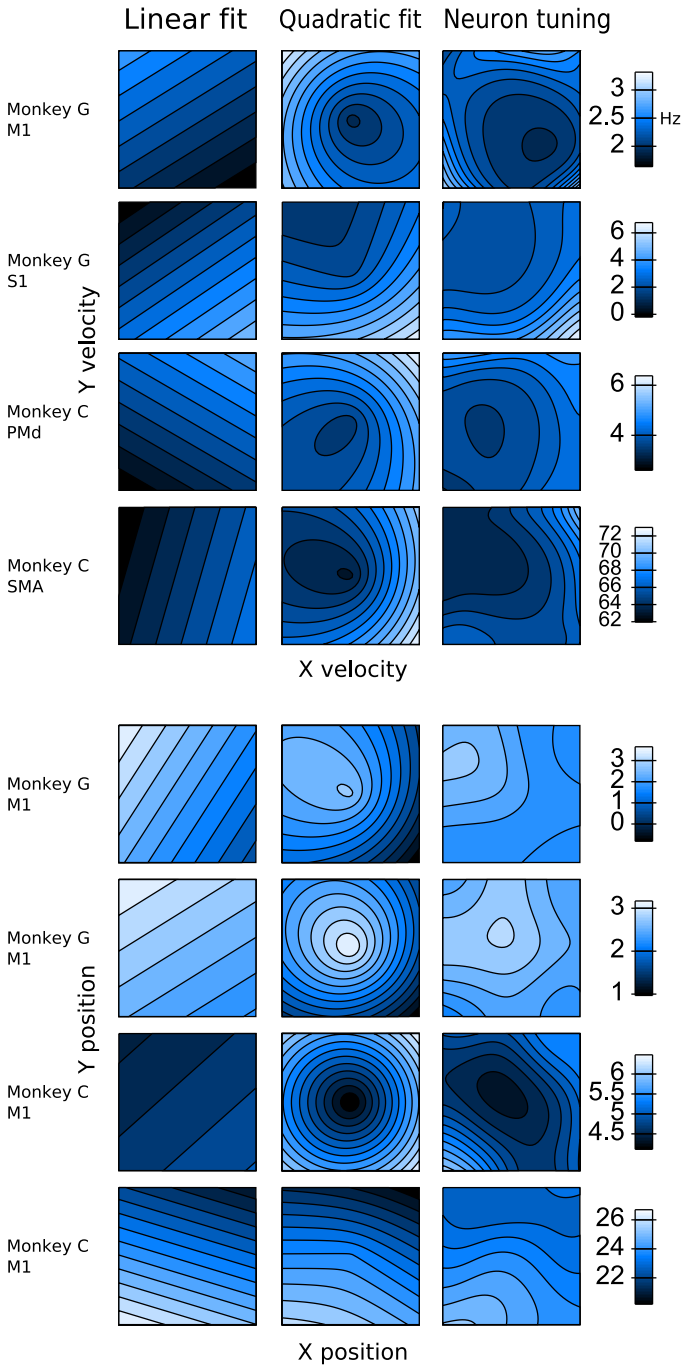

Figure 4.4: Contour plots of model fits. The plot axes are the x- and y-axis of the hand position or velocity. Brighter intensity of shading indicates higher firing rate, in spikes/sec. The right-most column depicts the smoothed true firing rate. The quadratic model captures the trends of neuronal modulations better than the linear model for most neurons.

a plane in the (rate, x, y) space. For position tuning (Figure 4.4, bottom 4 rows), the more expressive quadratic model captures the position tuning patterns better than the linear model. While more sophisticated models, such as higher-order or non-parametric models, may model neuronal activity more accurately, the quadratic model of tuning is relatively simple, fast to fit and evaluate, and supported by prior work [79].

4.2.2 Off-Line Reconstructions

Methods Tested

The n-th order unscented Kalman filter using the quadratic model of tuning and several comparison methods were evaluated off-line using data described in the previous subsection. The n-th order UKF used $n = 10$ taps, with 5 future taps and 5 past taps. The UKF with $n = 1$ past taps was also tested to evaluate the benefit of $n = 10$ taps. A standard Kalman filter, with the linear model described in Equation 2.6 on page 21, was evaluated to determine the benefit of using the quadratic model of tuning. For comparison against algorithms commonly used for closed-loop brain-control, a Wiener filter with 10 taps and the population vector method used by Taylor et al. [120] were evaluated.

Parameter Fitting

Ten-fold cross-validation was conducted to maximize use of data. Both the movement and neuronal tuning models were fit for each cross-validation fold. Ridge regression parameters for every algorithm fitted using ridge regression were chosen by optimizing for highest position reconstruction accuracy on the first cross-validation fold of each session: prediction was performed on the first segment with parameters fit from the remaining segments, and this process was repeated for different choices of λ (for the UKF, λ_F and λ_H were sought independently). Prediction on this first fold was

omitted when aggregating accuracy metrics. For on-line experiments, ridge regression parameters were set to $\lambda_F = \lambda_H = 15$ for the 10th order UKF, $\lambda_F = \lambda_H = 1$ for the 1st order UKF and Kalman filter, and $\lambda = 225$ for the Wiener filter. Wiener filter parameters were also fit with ordinary least squares (OLS) to demonstrate the benefit of regularization.

Kalman Filter for Comparison

The Kalman filter used for comparison had the same state variables as the 1st order UKF. The linear model used with the Kalman filter is described in Equation 2.6 on page 21. The parameters of the linear model were fitted in a similar way as the 1st order UKF.

Population Vector for Comparison

The original formulation of the population vector method predicted velocity and did not predict position directly. To make position predictions, the Cartesian position coordinates was substituted for the velocity components. The version of the population vector method used by Taylor et al. [120] was implemented with one slight modification: the baseline firing rate (mean) and normalization constant (standard deviation) of neurons were fit once from training data, instead of updated during filtering using a sliding window of spiking history. The neuronal weights were fit using ordinary least squares without regularization.

Results

The mean off-line reconstruction accuracies of the 10th order unscented Kalman filter (UKF), the 1st order unscented Kalman filter, the standard Kalman filter, the 10 tap Wiener filter fitted with ridge regression (RR), the 10 tap Wiener filter fitted with ordinary least squares (OLS), and the population vector method used by Taylor et al. [120] are shown in Figure 4.5 on the following page, grouped by monkey. The

y-axis shows the SNR of the hand position reconstruction. For the UKF and Kalman filter, the estimated mean state vector was used for the accuracy calculation. The x- and y-coordinate SNRs were averaged. Error bars indicate plus and minus one standard error over the 9 cross-validation folds and the x- and y-axis, for a total of 108 observations for monkey C and 180 observations for monkey G. Reconstruction accuracies for position and velocity are shown in Table 4.3 on the next page, grouped by behavioral task. Each cell in the table shows mean ± standard deviation for SNR (left) and CC (right). Bold entries indicate best values in each column.

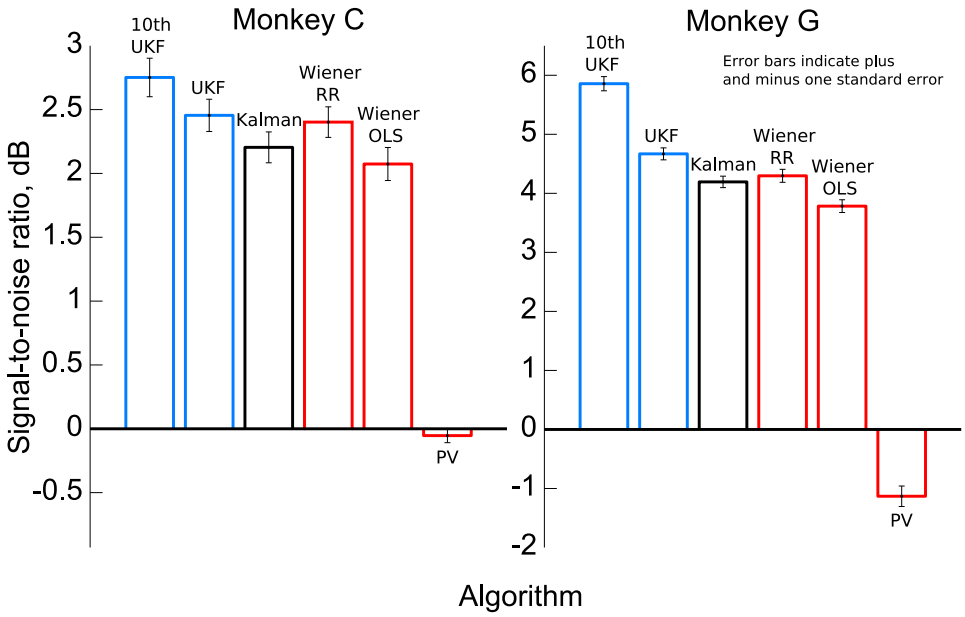

Figure 4.5: Reconstruction accuracy comparison among decoders. SNR of x- and y-axis predictions are averaged. Error bars indicate plus and minus one standard error.

Position: 10th order UKF

In terms of position estimates, the 10th order UKF with the quadratic model of tuning was consistently more accurate than the other algorithms tested. The two-sided, paired sign test with 288 observations (16 sessions, 9 folds, 2 dimensions) and

Table 4.3: Reconstruction accuracy comparison among decoders. Results grouped by behavioral task. Each entry shows SNR and CC ± standard error of 144 data points. The last column shows mean difference of each decoder compared against the Kalman filter, where larger numbers are better. Bold numbers indicate the best value in each column.

Method	Sessions 1-8 Center-out	Sessions 9-16 Pursuit(Lissajous)	Mean diff. from KF
	Position: SNR(db) ⋆ CC		
10th UKF	**3.37** ± 0.16 ⋆ **0.74** ± 0.01	**6.01** ± 0.13 ⋆ **0.87** ± 0.01	**1.24** ⋆ **0.07**
1st UKF	2.89 ± 0.12 ⋆ 0.70 ± 0.01	4.78 ± 0.11 ⋆ 0.82 ± 0.01	0.39 ⋆ 0.02
KF	2.59 ± 0.12 ⋆ 0.68 ± 0.01	4.31 ± 0.11 ⋆ 0.79 ± 0.01	0.00 ⋆ 0.00
WF RR	2.84 ± 0.12 ⋆ 0.69 ± 0.01	4.33 ± 0.13 ⋆ 0.78 ± 0.01	0.14 ⋆ 0.00
WF OLS	2.29 ± 0.11 ⋆ 0.67 ± 0.01	4.00 ± 0.12 ⋆ 0.77 ± 0.01	-0.31 ⋆ -0.01
PV	-0.26 ± 0.07 ⋆ 0.40 ± 0.01	-1.20 ± 0.21 ⋆ 0.47 ± 0.01	-4.17 ⋆ -0.30
	Velocity: SNR(db) ⋆ CC		
10th UKF	**1.77** ± 0.09 ⋆ **0.57** ± 0.01	**2.10** ± 0.06 ⋆ **0.61** ± 0.01	**0.36** ⋆ **0.05**
1st UKF	1.53 ± 0.07 ⋆ 0.53 ± 0.01	1.80 ± 0.06 ⋆ 0.57 ± 0.01	0.09 ⋆ 0.01
KF	1.43 ± 0.07 ⋆ 0.52 ± 0.01	1.72 ± 0.06 ⋆ 0.56 ± 0.01	0.00 ⋆ 0.00
WF RR	1.47 ± 0.07 ⋆ 0.52 ± 0.01	1.82 ± 0.06 ⋆ 0.58 ± 0.01	0.07 ⋆ 0.01
WF OLS	0.87 ± 0.07 ⋆ 0.49 ± 0.01	1.37 ± 0.05 ⋆ 0.54 ± 0.01	-0.46 ⋆ -0.02
PV	-0.58 ± 0.08 ⋆ 0.38 ± 0.01	-0.76 ± 0.14 ⋆ 0.42 ± 0.01	-2.24 ⋆ -0.14

significance level $\alpha = 0.05$ was used to evaluate significance. The 10th order UKF produced position estimates with significantly higher SNR than the 1st order UKF ($p < 0.001$, mean difference 0.85 dB), the standard Kalman filter ($p < 0.001$, mean difference 1.25 dB), the 10 tap Wiener filter fitted with ridge regression ($p < 0.001$, mean difference 1.11 dB), the 10 tap Wiener filter fitted with ordinary least squares ($p < 0.001$ mean difference 1.55 dB), and Taylor's variant of the population vector method ($p < 0.001$, mean difference 5.42 dB). When sessions of pursuit task and center-out task were analyzed separately, the 10th order UKF was 1.23 dB more accurate than the 1st order UKF in the pursuit task and 0.48 dB more accurate in the center-out task.

Position: 1st order UKF

The 1st order UKF produced position estimates with significantly higher SNR than the standard Kalman filter ($p < 0.001$, mean difference 0.39 dB), the 10 tap Wiener filter fitted using ridge regression ($p < 0.001$, mean difference 0.25 dB), the 10 tap Wiener filter fitted with ordinary least squares ($p < 0.001$, mean difference 0.70 dB), and Taylor's variant of the population vector method ($p < 0.001$, mean difference 4.57 dB).

Velocity: 10th order UKF

For predicting velocity, the 10th order UKF produced estimates with significantly higher SNR than the 1st order UKF ($p < 0.001$, mean difference 0.27 dB), the standard Kalman filter ($p < 0.001$, mean difference 0.36 dB), 10 tap Wiener filter fitted with ridge regression ($p < 0.001$, mean difference 0.29 dB), the 10 tap Wiener filter fitted with ordinary least squares ($p < 0.001$ mean difference 0.82 dB), and Taylor's variant of the population vector method ($p < 0.001$, mean difference 2.60 dB).

Velocity: 1st order UKF

The 1st order UKF produced velocity estimates with significantly higher SNR than the standard Kalman filter ($p < 0.001$, mean difference 0.09 dB), the 10 tap Wiener filter fitted with ordinary least squares ($p < 0.001$ mean difference 0.55 dB), and Taylor's variant of the population vector method ($p < 0.001$, mean difference 2.33 dB).

Similar results were obtained when the correlation coefficient was used as the metric for decoder accuracy.

4.2.3 UKF Analyses

Several analyses were performed on the UKF. The accuracy dependency on the number of taps was measured. The contributions of the movement model and observation model were quantified. The benefit of using a 10 tap quadratic model of tuning versus a 1 tap model was measured. The approximation error arising from the unscented transform was measured. Accuracy versus number of neurons was measured for the UKF and comparison algorithms. Finally, computational complexity and execution speed of the algorithm in MATLAB are presented.

Accuracy Versus Future and Past Taps

The quadratic model of tuning relates neuronal activity with behavior both prior to and after the time instant of neuronal activity. The parameters past taps and future taps, in units of 100 ms, described the time offsets prior to and after the instant of neuronal activity between which tuning was modeled. The relationship between the number of future and past taps and reconstruction accuracy for the n-th order UKF was measured by making reconstructions with various settings. The ridge regression parameter was optimized for each setting using the first fold of ten-fold cross-validation, and the accuracy on the remaining nine folds is reported.

Plots of the mean position accuracy for various numbers of future and past taps for two sessions are shown in Figure 4.6 on the following page. The left panel shows session 1 with the center-out task, and the right panel shows session 16 with the pursuit task with Lissajous trajectories. The number of future taps is shown on the x-axis and each setting of past taps is depicted as a separate curve. For the pursuit task session, reconstruction accuracy steadily increases with the number of future taps and increases slowly with the number of past taps. For the center-out task, the accuracy was maximized when 15 past and 2 future taps were used. A large number of future taps resulted in decreased accuracy, while the number of past taps had small

effects on accuracy.

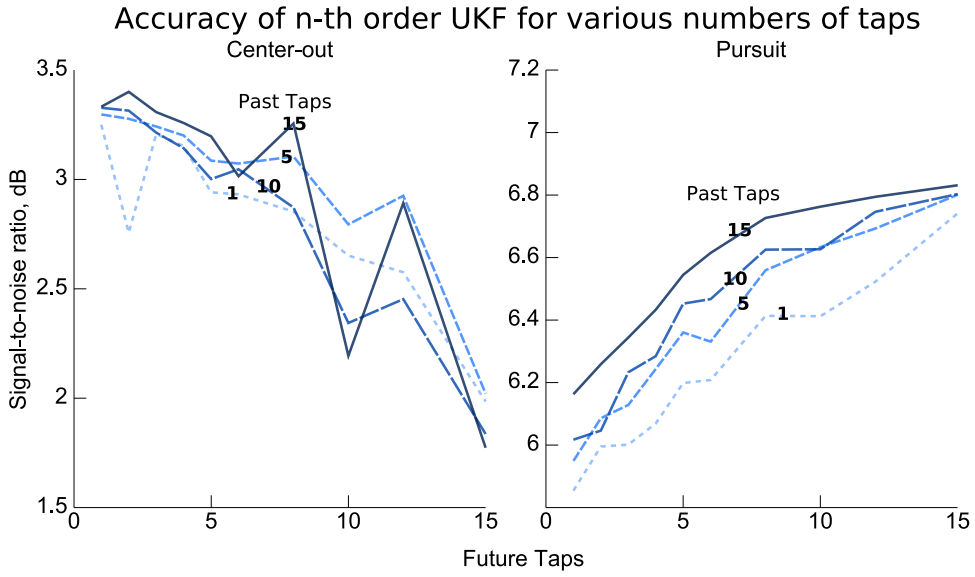

Figure 4.6: UKF Reconstruction accuracy versus taps. Larger number of taps helps with pursuit task.

Model Contribution

To evaluate the capacity of the movement model to predict hand trajectories, two analyses were conducted. In the first analysis, the update step of the 10th order UKF was disabled so that the filter ignored neuronal activity and used only the movement model to dead-reckon. In the second analysis, the movement model was not fit to the training data but set so that position was the discrete integral of velocity and velocity remained constant except for noise perturbations, called the constant-velocity movement model (See Section 2.5 on page 37 and Equation 2.31 on page 38). The movement model noise covariance was fitted to the training data under the constant-velocity movement model by calculating the mean squared-error matrix of the residuals when this model was used to predict next states.

Figure 4.7 on the next page shows example traces of reconstructions under these two conditions on session 16. The true position of the hand, measured from the

joystick, is shown by the thick dashed curve. The dead-reckoning filter (dash-dotted curve) produced useless predictions shortly after filtering began, showing that the movement model could not reconstruct the hand trajectory alone, even though the monkey tried to follow a deterministic Lissajous curve. The 10th order UKF with the constant-velocity movement model (dotted curve) produced less accurate predictions than the filter with movement model fitted from the data. The position estimate SNR of the 10th order constant-velocity movement model filter was 3.88 ± 0.27 dB (mean $\pm$ standard error, 18 observations) for the pursuit task session (session 16) and 2.89 ± 0.44 dB in the center-out task session (session 1). For the fully-functional 10th order UKF, the SNR was 6.25 ± 0.23 dB for the pursuit task session and 4.08 ± 0.36 dB for the center-out task session. These results demonstrate a large benefit to using a fitted movement model, especially for the pursuit task.

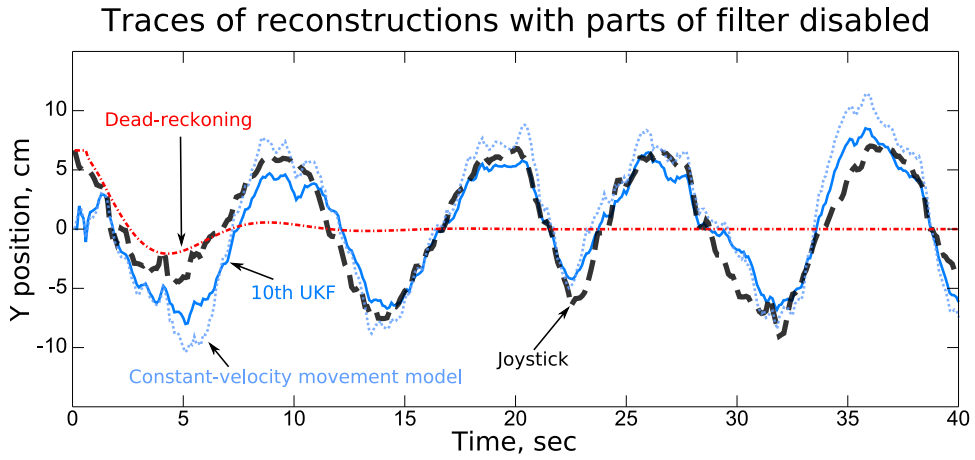

Figure 4.7: Traces of reconstruction with parts of filter disabled. The 10th order UKF with the constant-velocity movement model produced less accurate reconstructions than the fully-functional 10th order UKF. The 10th order UKF which ignored observations (dead-reckoning) could not produce useful reconstructions.

Taps in Tuning Model

To evaluate the benefit of the 10 tap quadratic model of tuning versus the 1 tap quadratic model of tuning, a 1 tap constant-velocity movement model filter was evaluated. Since the constant-velocity movement model was not fitted to data and the movement model noise covariance matrices were identical for the 1 tap and 10 tap cases, the difference in accuracy between the 1st order and 10th order constant-velocity movement model filters must arise from the different accuracies of the 1 tap and 10 tap quadratic models of tuning. The position estimate SNR of the 1st order constant-velocity movement model filter for the pursuit task was 1.57 ± 0.76 dB (compare to 3.88 ± 0.27 dB for 10th order). For the center-out task, accuracy was 0.54 ± 0.59 dB (compared to 2.89 ± 0.44 dB). The large differences in accuracy (2.30 and 2.35 dB) shows the benefit of modeling neuronal tuning across multiple time offsets simultaneously. However, much of this benefit likely comes from the autocorrelation of movements, which is also captured by the 10th order movement model fitted to data.

Approximation Penalty of Unscented Transform

To quantify the extent the approximations of the unscented transform in the UKF affected accuracy, reconstructions using particle filters [23] with identical models as the 1st and 10th order UKF were made. The particle filters used 50,000 particles and the same model parameters, initial conditions, and test data as the UKF. Since there were many sessions and cross-validation folds for comparison, only one particle filter run was performed per session and cross-validation fold. The posterior mean of the particles was the output used for comparison.

For the 1st order model, the particle filter produced significantly more accurate position reconstructions than the UKF (two-sided, paired sign-test, 288 observations, $p < 0.001$, mean difference 0.07 dB). For the 10th order model, the difference in

accuracy was not significant at the $\alpha = 0.05$ level, with the UKF having a nominal 0.02 dB advantage in mean SNR. This was likely due to the large state space (40 dimensions) associated with the 10th order model. With such a large state space, even 50,000 particles could not represent the state distribution as well as a multivariate normal distribution, hence the UKF provided similar accuracy even with the unscented transform approximation.

Accuracy Versus Number of Neurons

Figure 4.8 on the following page shows reconstruction accuracy for a pursuit task session when different-sized subsets of the neurons are used ("neuron dropping curves"). For each setting of the number of neurons on the x-axis, 10 subsets of neurons of that cardinality were randomly selected with uniform probability and each algorithm was evaluated on these subsets using ten-fold cross-validation. To ensure comparability, all algorithms used the same subsets of neurons. The first fold was reserved for finding optimal ridge regression parameters, and the mean position reconstruction accuracies on the nine remaining folds are plotted in Figure 4.8.

The 1st and 10th order UKF reconstructs position more accurately than the Kalman filter, Wiener filter, and population vector method even for small numbers of neurons.

Wiener Filter and Ridge Regression

The Wiener filter fitted with ridge regression approaches the accuracy of the 1st order UKF as the number of neurons increases. The benefit of ridge regression versus ordinary least squares for fitting the Wiener filter grows as the number of neurons, and with it the number of parameters, increases.

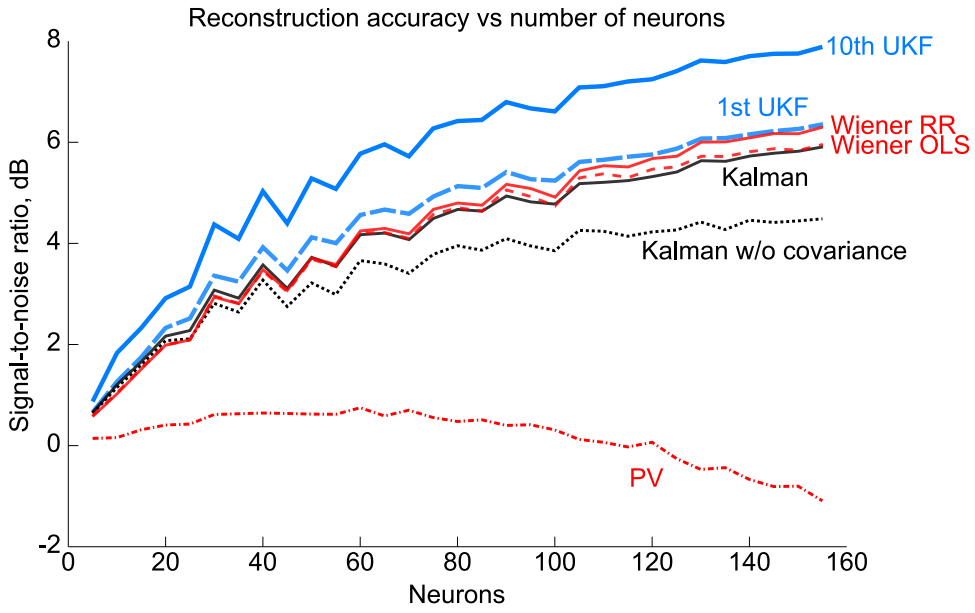

Figure 4.8: Reconstruction accuracy versus number of neurons. The y-axis depicts the mean accuracy among 10 random subsets of neurons. For each value of the number of neurons, all algorithms used the same 10 subsets. The curve labeled Kalman w/o covariance indicates the accuracy of a Kalman filter with the off-diagonal entries of the tuning model noise covariance matrix set to zero.

Importance of Modeling Noise Covariance

Modeling the noise covariance between neurons becomes more important as the number of neurons increases. This effect can be seen by the lower accuracy of a modified Kalman filter which does not model tuning noise covariance (Kalman w/o covariance) compared to the unmodified Kalman filter. The tuning model noise covariance matrix of the Kalman w/o covariance filter has all entries not on the diagonal set to zero. This result supports the findings of Wu et al. [149]

Population Vector Method

The population vector method peaks in accuracy at around 60 neurons and then decreases in accuracy. Results for velocity reconstruction are similar. This demonstrates the suboptimality of the parameter fitting procedure for the population vector

method. The fitting procedure works on neurons separately, ignoring the covariance among neurons. When the population size increases and the amount of redundancy in modulation increases, the penalty for ignoring the covariance increases. Eventually the penalty reverses any gains from using more neurons.

Computational Complexity and Speed

The empirically slowest computation in the UKF is the Cholesky decomposition used in the matrix square root for generating the sigma points (Equation 2.19 on page 33) and for inverting the observation noise covariance (Equation 2.28 on page 35). Standard Cholesky decomposition algorithms have $O(m^3)$ computational complexity, where m is the number of rows or columns of the square input matrix. In the sigma point calculation, the input matrix size is the dimensionality of the state ($m = d = 4n$). In the observation noise covariance inversion, the input size is the number of neurons ($m = N$). Overall, the computational complexity of one filtering iteration with the quadratic model of tuning is $O(d^3 + N^3)$.

In terms of execution speed, the MATLAB implementation of the 10th order UKF on an Intel Pentium 4 class desktop computer used 0.012 ± 0.005 seconds per iteration (mean $\pm$ standard deviation), or around 80 Hz on average. The 30th order UKF (15 future and 15 past taps) used 0.0360 ± 0.0001 seconds per iteration, or around 28 Hz on average. The real-time implementation of the UKF in C++ using Automatically Tuned Linear Algebra Software (ATLAS library) executed much faster than 10 Hz, the binning frequency.

4.2.4 Closed-Loop Control

The 10th order UKF was compared to the Kalman filter and Wiener filter in on-line, closed-loop brain-control in 6 recording sessions: 3 with monkey C and 3 with monkey G. In each session, the monkey first performed pursuit task with Lissajous trajectories

using joystick control for 6 to 10 minutes. Five minutes of data from this time period was used to fit parameters for the algorithms. In each session, all algorithms were fit on the same data. Then the monkey performed pursuit task using brain-control with each algorithm, in turn, for 5 to 8 minutes each. The evaluation order of the algorithms was switched between sessions. However, not all orderings could be used in the 3 sessions for each monkey. During brain-control, the monkey was required to hold the joystick as an indication of active participation; time periods when the monkey did not hold the joystick were omitted from the analysis.

Performance was measured by comparing the position trajectory of the target (the desired signal for SNR calculations) and the trajectory of the algorithm-controlled cursor. Table 4.4 on page 106 shows the SNR and CC for each algorithm in each session, with mean taken over the x and y-coordinates. Bold entries indicate best values in each row. The Wiener filter was not evaluated in session 17. The two-sided, paired sign-test was used to measure significant difference in control accuracy between different algorithms. The two axes were treated separately, and the significance value was set to $\alpha = 0.05$.

Results

Figure 4.9 on the following page shows example traces of the brain-controlled cursor and target positions in session 19. In terms of SNR, the monkeys controlled significantly more accurately when using the 10th order UKF than when using the Kalman filter ($p < 0.05$, 12 observations) and 10 tap Wiener filter fitted with ridge regression ($p < 0.05$, 10 observations). In terms of CC, no comparison was significantly different at the $\alpha = 0.05$ level.

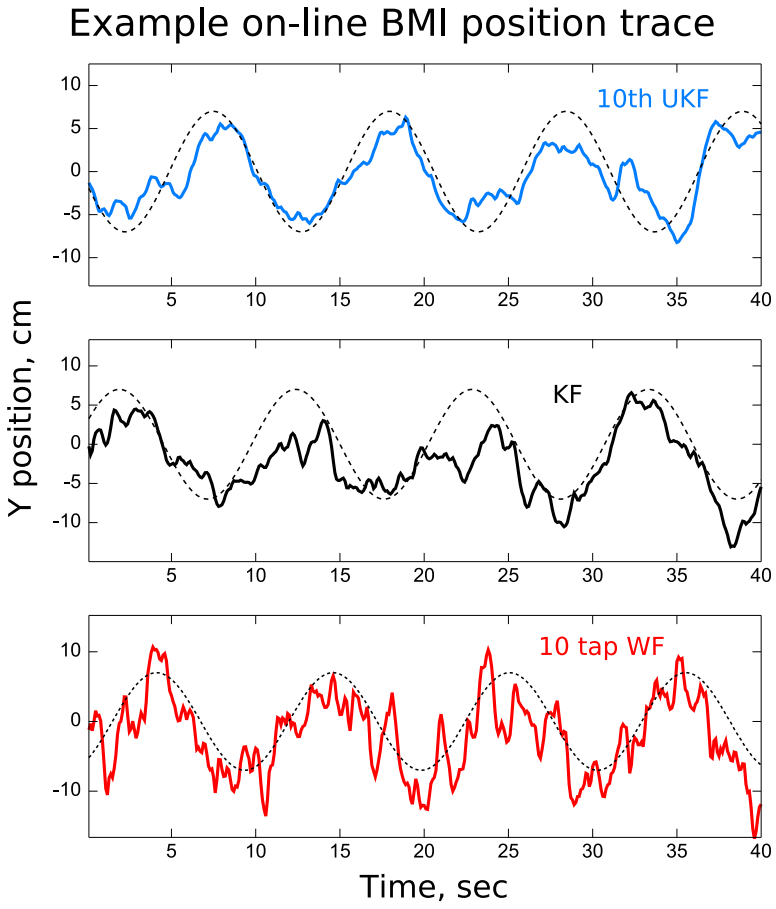

Figure 4.9: Example closed-loop brain-control traces. First 40 seconds of brain-control with each algorithm are shown from session 19. Dashed curves indicate target position.

Table 4.4: Closed-loop brain-control accuracy comparison among decoders. Bottom row shows mean difference of each algorithm compared against the Kalman filter, where larger numbers are better. Bold numbers indicate the best value in each row.

Session	Monkey	10th UKF	KF	WF
		SNR (dB) ⋆ CC		
17	C	**2.70 ⋆ 0.69**	0.70 ⋆ 0.47	NA
18	C	**2.73 ⋆ 0.72**	2.42 ⋆ 0.60	-1.13 ⋆ 0.54
19	C	**2.51 ⋆ 0.71**	0.80 ⋆ 0.53	0.07 ⋆ 0.68
20	G	-2.12 ⋆ 0.10	**-1.49 ⋆ 0.15**	-3.23 ⋆ 0.07
21	G	**1.58** ⋆ 0.56	1.55 ⋆ 0.57	0.77 ⋆ **0.58**
22	G	**3.23 ⋆ 0.71**	0.39 ⋆ 0.48	-0.06 ⋆ 0.47
Mean difference from KF		**1.04 ⋆ 0.12**	0.00 ⋆ 0.00	-1.45 ⋆ 0.00

4.3 Adaptive Decoding

4.3.1 Synthetic Experiments

Synthetic experiments were conducted with a toy problem to demonstrate the ability of Bayesian regression self-training to update observation models. The toy problem involves tracking a moving target in one dimension using sensors which observe position, velocity, and acceleration. The state-space has the variables position, velocity, and acceleration, in that order. The target's transition model is a slightly modified constant-acceleration model:

$$\mathbf{F} = \begin{bmatrix} 1 & 1 & 0 \\ -10^{-5} & 1 - 10^{-3} & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10^{-3} \end{bmatrix} \tag{4.7}$$

The observation model matrix is initially the identity matrix but changes during filtering. The observation model noise covariance matrix is:

$$\mathbf{R} = \begin{bmatrix} 10^{10} & 0 & 0 \\ 0 & 10^{4} & 0 \\ 0 & 0 & 10^{-2} \end{bmatrix} \tag{4.8}$$

A random state trajectory with length 10,000 iterations and corresponding set of observations were generated using this model for the experiments.

Filter Settings

A 1st order UKF, as described in Chapter 2, with factorized distribution variational Bayesian regression, as described in Chapter 3, was used to perform adaptive filtering. The UKF used 1 past tap. The transition and observation models of the UKF were fit using a separate training data set with 10,000 iterations generated using the above

transition model and a stationary, identity observation model. The prior transition and observation model coefficient variance for fitting was $\lambda = 10^{-8}$. The fitted observation model used position, velocity, acceleration, and a bias term as features, in that order.

System with One Changing Sensor

In the first experiment, the system's observation model entry for velocity linearly increased from 1 to 3 during the 10,000 iterations of testing data:

$$\mathbf{H}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 + \frac{2t}{10000} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

The filter had no direct access to the system's transition and observation models; instead, parameter fitting was performed as described above.

Filter Given Which Sensor Changes

After parameter fitting, the precisions of the observation model coefficients were set so that the velocity entry of the observation model was allowed to change while all other entries were fixed. To do this, the $(2, 2)$ entry of $\mathbf{\Lambda}_2$ was set to 10^3, all other diagonal entries of each $\mathbf{\Lambda}_i$ were set to 10^{15}, and all non-diagonal entries of $\mathbf{\Lambda}_i$ were set to 0.

In this situation, allowing only one sensor to change avoids divergence. Velocity can be inferred from position and acceleration observations, which are not subject to change. Then the estimated velocity can be used to update the observation model parameter for velocity.

Results

The 1st order UKF without updates (*static*) and 1st order UKF with factorized distribution variational Bayesian regression self-training (*adaptive*) were used to filter

the testing data. Updates used 30 iterations of data with model drift parameter set to 10^{-2} for velocity and zero for position, acceleration, and bias. Degrees of freedom cap was set to 20,000.

Figure 4.10 shows traces of filter outputs and the observation model parameter for velocity found by the UKF with VBR. For this system without risk of divergence, the adaptive filter estimated much more accurately than the non-adaptive filter and could track the changing observation model parameter.

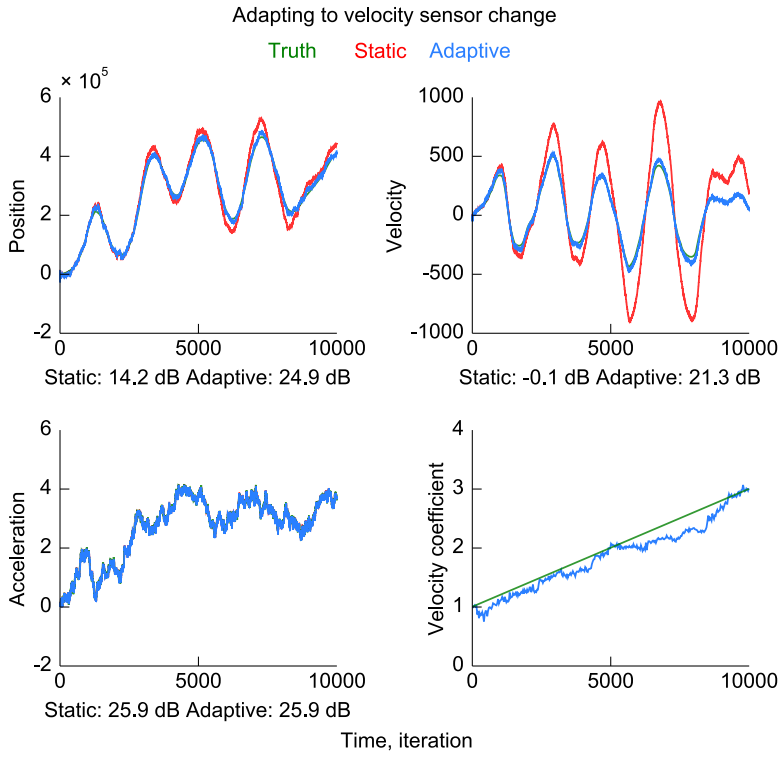

Figure 4.10: Adaptive filtering on synthetic system with one changing sensor. Only the velocity sensor changed, and the adaptive filter was set to track parameter changes in the velocity sensor while keeping other sensors fixed.

System with Multiple Redundant Changing Sensors

In the second experiment, the observation model entries for position, velocity, and acceleration were all allowed to change and the adaptive filter had no knowledge of

which entries changed. The observation model was expanded to include 4 duplicate sensors for each state variable, for a total of 12 observation dimensions.

For generating the training data, the system's observation model was four identity matrices of size 3 by 3 stacked vertically. The observation noise covariance, for both training and testing data, was four copies of the matrix in Equation 4.8 placed on the block-diagonal of a 12 by 12 matrix. In the testing data, the system's observation model varied with time t:

$$\mathbf{H}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 + \frac{t}{10000} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 + \frac{t}{10000} & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 - \frac{t}{10000} \end{bmatrix} \tag{4.10}$$

Filter Allows All Sensors to Change

The filter was not given knowledge of which sensors changed; instead it assumed all sensors may change. After parameter fitting on the training data, the precisions of the observation coefficients were set in the following ways. The $(1, 1)$ entry of $\mathbf{\Lambda}_1$, $\mathbf{\Lambda}_4$, $\mathbf{\Lambda}_7$, and $\mathbf{\Lambda}_{10}$ (the position sensors), the $(2, 2)$ entry of $\mathbf{\Lambda}_2$, $\mathbf{\Lambda}_5$, $\mathbf{\Lambda}_8$, and $\mathbf{\Lambda}_{11}$ (the velocity sensors), and the $(3, 3)$ entry of $\mathbf{\Lambda}_3$, $\mathbf{\Lambda}_6$, $\mathbf{\Lambda}_9$, and $\mathbf{\Lambda}_{12}$ (the acceleration sensors) were set to 10^{12}. All other diagonal entries of each $\mathbf{\Lambda}_i$ were set to 10^{15}, and all non-diagonal entries were set to 0. These settings ensure that each sensor may only observe its assigned state variable. Every sensor may change, meaning the inferred states and parameters may diverge from the true values. Alertness is decreased with a larger initial precision of 10^{12} (versus 10^3 for the previous experiment). This change

was necessary because adaptive filtering on this system is more vulnerable to noise.

Results

The 1st order UKF without updates and 1st order UKF with factorized distribution variational Bayesian regression self-training were used to filter the testing data. Updates used 30 iterations of data with model drift parameter set to 10^{-4} for all observation model coefficients allowed to change (those with $\mathbf{\Lambda}$ entries set to 10^{12}), and zero otherwise. Degrees of freedom cap was set to 20,000.

Figure 4.11 on the following page shows traces of filter outputs and the observation model parameters found by the UKF with VBR. For this system where all sensors are allowed to change, the adaptive filter was more accurate in position and velocity and slightly more accurate in acceleration than the non-adaptive filter. Overall, accuracy in this experiment was worse than accuracy for the system with a single changing sensor, as expected. The adaptive filter could track changes in the position and velocity coefficients. The changing acceleration coefficient was also tracked, but the non-changing acceleration coefficients were estimated incorrectly. However, this example shows adaptive filtering in a system with redundant sensors which are all allowed to change can be better than non-adaptive filtering.

Errors in Acceleration Coefficient Estimates

The errors in the acceleration coefficient estimates are illustrative of the difficulty of adaptive filtering for a system where all sensors are allowed to change. From the top right panel of Figure 4.11, it can be seen that the acceleration was larger than zero for the duration of the trajectory. The adaptive filter reports that both the acceleration and the acceleration sensors changed. The amount of change in each is a compromise calculated given the state and parameter transition models. Due to this compromise, the estimated acceleration is smaller than the true value, and

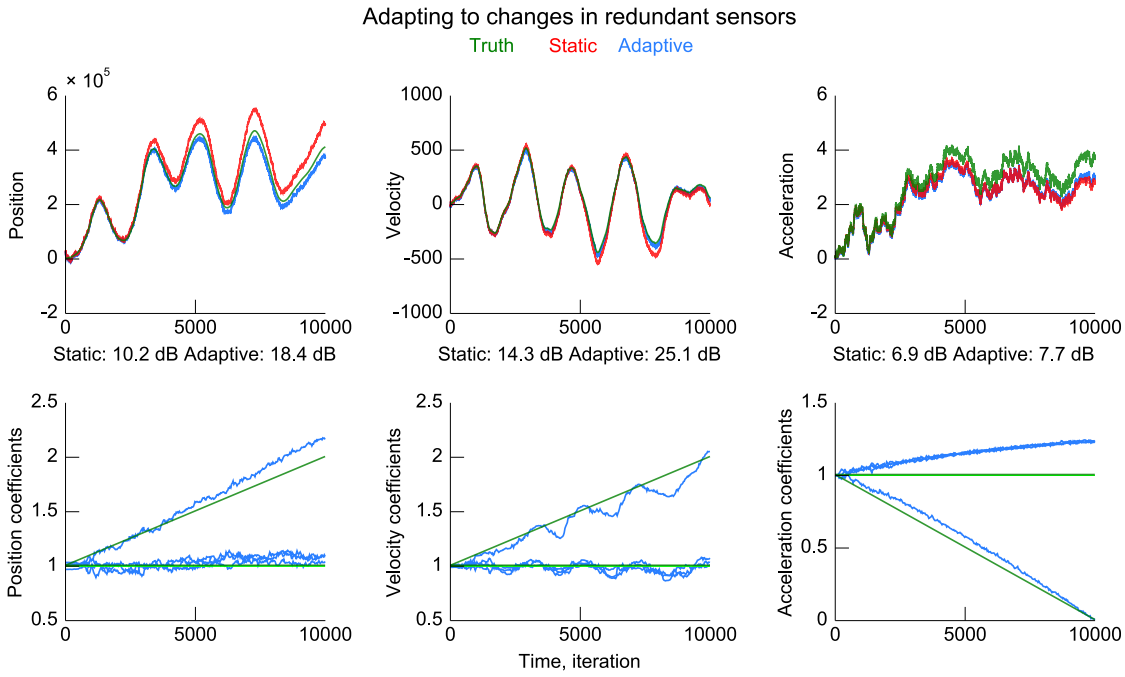

Figure 4.11: Adaptive filtering on synthetic system where all sensors are allowed to change. Out of the 4 redundant sets of sensors for position, velocity, and acceleration (12 sensors total), one each of position, velocity, and acceleration sensors changed. The adaptive filter was not given knowledge of which sensors were changing.

the estimated observation model coefficients for acceleration are larger than the true values. Note that, because of the batch-mode updates, the compromise occurs over multiple updates instead of during each filtering iteration, as would be the case for joint filtering.

Errors in Position Coefficient Estimates

A similar story occurs with the position estimates, but the errors are smaller. The transition model has no noise for position, so changes in position are only due to changes in velocity, which are in turn affected by acceleration. Thus, the velocity and acceleration observations provide useful information for the position estimates, which help the estimation of position coefficients.

Advantage of Using Kalman Smoothing

Position and velocity observations provide useful information for acceleration coefficient estimates, even though the **F** matrix's row for acceleration has position and velocity entries of 0. This is due to the Kalman smoothing performed before each update. The Kalman smoothing applies the transition model backwards in time, which allows position and velocity estimates to improve acceleration estimates. In the joint filtering and dual filtering approaches, inference through state variable interactions in the backwards time-direction does not occur, because no smoothing is performed before model updates.

4.3.2 Off-Line Reconstructions

The ability of the Bayesian regression self-training method to adapt the tuning model of the n-th order UKF decoder was tested in off-line reconstructions of previously-recorded sessions during which monkeys controlled the cursor with their hands. Eighteen sessions from 3 monkeys were used, each with hand-controlled behavior lasting longer than 12 minutes. Some of these sessions were used in the reconstructions in the previous section. The number of neurons in each session ranged between 94 and 240 (mean 143). The sessions either used the center-out task, the Lissajous pursuit task, or the point-to-point pursuit task. Information about sessions is shown in Table 4.5.

Data and Parameter Settings

For each session, the first minute was discarded and the following 2 minutes of recording were used for parameter fitting, using Bayesian regression with the joint and factorized distributions as described in Section 3.6.

Then, the n-th order unscented Kalman filter described in Chapter 2 with Bayesian regression self-training was used to reconstruct hand trajectories on the remainder of

the session. Updates using the joint distribution variant of Bayesian regression were performed upon initial fits made with the joint variant of Bayesian regression; updates using the factorized distribution variant of Bayesian regression were performed upon initial fits made with the factorized Bayesian regression.

The update interval was set to 2 minutes, corresponding to 1,200 data points in each update. The model drift parameter was systematically varied to find the most suitable value. The degrees of freedom cap was set to 12,000, corresponding to 20 minutes of data samples. The UKF used $n = 5$ taps, with 2 future taps and 3 past taps. The movement model was reduced to be autoregressive order 1 instead of 5, to focus on the effects of the tuning model.

Test Conditions

For comparison to a non-adaptive algorithm, the unscented Kalman filter with the same settings and parameter fit, but without self-training updates, was used to reconstruct hand trajectories. Furthermore, to measure how well Bayesian regression self-training updates would perform if they were supplied with hand movements, the unscented Kalman filter updated with hand movements recorded from the joystick instead of filter outputs was also used to make reconstructions. Note that hand movements were not smoothed using the Kalman smoother. Comparing to this condition provides an estimate of the penalty for decoding error and model misspecification in self-training. It provides an idea of how far the accuracy of the self-training paradigm is from the best accuracy one could reasonably expect.

The 6 test conditions were: non-updated UKF using parameters fit by joint distribution Bayesian regression (condition *static-BR-fit*), non-updated UKF using parameters fit by factorized distribution variational Bayesian regression (*static-VBR-fit*), UKF updated with BR self-training (*BR*), UKF updated with VBR self-training (*VBR*), UKF updated with BR using actual hand trajectories (*BR-hand*), and UKF

updated with VBR using actual hand trajectories (*VBR-hand*).

Metric

Accuracy was measured by calculating the signal-to-noise ratio of the hand positions collected from the joystick (desired signal) versus the reconstructed hand positions. The estimated mean state vector from filtering (not smoothing) was used for the accuracy calculation. Only the reconstructions after the first 2 minutes of testing data were included in this accuracy calculation, since prior to the first update at the 2-minute mark, the static and updated conditions output exactly the same state estimates. The x- and y-axis SNR were averaged to produce one number per condition per session.

Results: Self-Training Updates

Figure 4.12 on the next page shows the mean SNR of 5 conditions for each monkey, with the static fit with VBR condition omitted for clarity. Table 4.5 on page 117 shows the results for all 6 conditions for each session. For the BR, BR-hand, VBR, and VBR-hand conditions, the best result among the settings of model drift for each session is shown.

Except on one session (17), BR and VBR consistently produced more accurate reconstructions than static. On average, the improvement of BR versus static-BR-fit was 0.54 dB (two-sided, paired sign-test, n=18, $p < 0.001$). The improvement of VBR versus static-VBR-fit was 0.50 dB ($p < 0.001$). BR was significantly more accurate than VBR ($p < 0.05$), with an advantage of 0.10 dB. The static condition fitted with BR produced, on average, more accurate reconstructions than the static condition fitted with VBR (0.064 dB mean difference, $p < 0.05$).

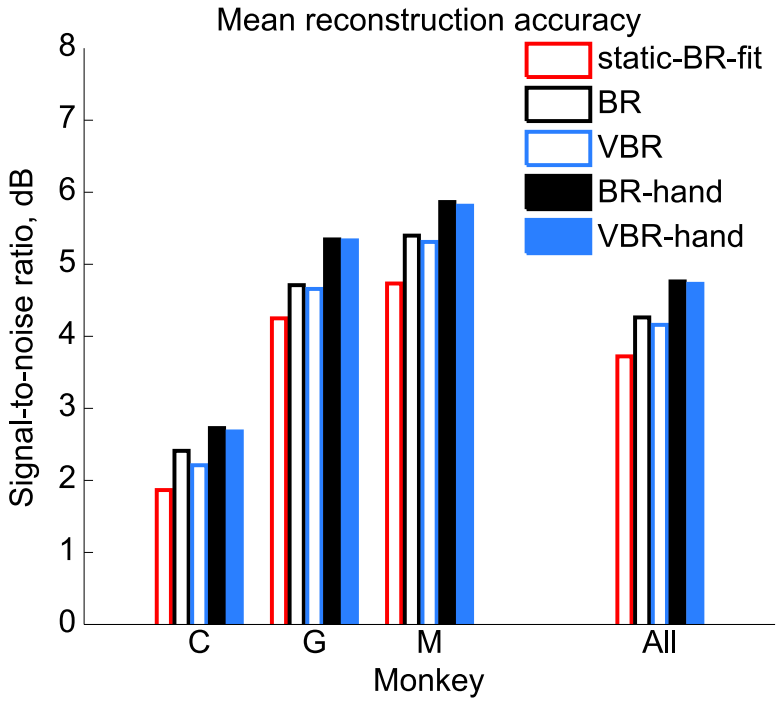

Figure 4.12: Reconstruction accuracy of UKF with BR and VBR updates. The static condition used parameters fit with joint distribution Bayesian regression. Accuracy of UKF with self-training updates were significantly better than accuracy of UKF without updates. Accuracy of UKF with updates using actual hand movements were significantly better than accuracy of UKF with self-training updates.

Results: Ground Truth Updates

The BR-hand reconstructions were on average 1.04 dB more accurate than the static-BR-fit reconstructions ($p < 0.002$) and 0.50 dB more accurate than the BR reconstructions ($p < 0.05$). VBR-hand reconstructions were on average 1.07 dB more accurate than the static-VBR-fit reconstructions ($p < 0.002$) and 0.57 dB more accurate than the VBR reconstructions ($p < 0.05$).

In session 17, the only session where BR and VBR produced less accurate reconstructions than static, BR-hand and VBR-hand produced even less accurate reconstructions than BR and VBR. In this session, the BR and VBR errors may not be due to self-training, but due to some other aspects of the method or to particular

Table 4.5: Reconstruction accuracy of UKF with BR and VBR updates. Accuracy values in dB. Static UKF used parameters fitted with Bayesian regression (Static-BR-fit) or variational Bayesian regression (Static-VBR-fit).

Session/ Monkey	Length (min)	Neurons	Static-BR-fit	BR	BR-hand	Static-VBR-fit	VBR	VBR-hand
1 G	20	139	3.90	4.49	5.79	3.89	4.27	5.79
2 G	19	200	4.70	5.15	5.68	4.51	5.05	5.67
3 G	15	189	5.53	6.47	6.68	5.44	6.42	6.66
4 G	10	157	6.34	6.77	6.90	6.38	6.77	6.89
5 G	22	94	4.33	4.45	3.91	4.29	4.43	3.95
6 G	13	103	4.27	4.45	5.34	4.26	4.49	5.33
7 G	12	103	3.76	4.51	5.30	3.80	4.40	5.29
8 G	12	240	1.15	1.39	3.15	1.12	1.43	3.08
9 C	10	101	3.59	4.06	4.03	3.38	3.84	4.03
10 C	11	101	1.55	2.45	2.71	1.47	2.17	2.68
11 C	19	109	2.15	2.55	3.33	2.15	2.48	3.27
12 C	18	131	0.63	1.21	1.55	0.34	0.93	1.54
13 C	10	157	1.40	1.78	2.02	1.16	1.62	1.87
14 M	33	196	5.24	6.03	5.97	5.29	6.09	5.97
15 M	17	145	3.93	4.47	4.80	3.87	4.18	4.79
16 M	54	134	4.03	5.00	5.06	3.96	4.99	4.98
17 M	72	134	7.12	7.01	6.72	7.09	6.93	6.65
18 M	59	134	3.35	4.49	6.77	3.43	4.36	6.67
Mean	23.7	142.6	3.72	4.26	4.76	3.66	4.16	4.73

properties of the session. Two other sessions (5 and 14) also have the BR-hand and VBR-hand conditions producing less accurate reconstructions than BR and VBR. The reason for this is unclear.

Accuracy Versus Time

The solid lines in Figure 4.13 on page 119 show the reconstruction accuracy in one-minute windows versus time for each session. The static-BR-fit condition is shown in red, the BR condition is shown in black, and the VBR condition is shown in blue. The dotted lines indicate the mean within the entire session for each condition. The BR

and VBR reconstruction accuracy is generally higher than the static-BR-fit accuracy. In several sessions the BR accuracy is slightly higher than the VBR accuracy.

Improvement Versus Time Analysis

To examine whether BR and VBR improvement over static increased or decreased with time, accuracy differences (BR minus static-BR-fit, VBR minus static-VBR-fit) in one-minute windows were correlated with time in the session. Fifteen of eighteen sessions showed a positive correlation. However, only the positive correlations in two sessions were statistically significant (t-test, $p < 0.05/18$ Bonferroni correction for multiple comparisons) for BR and one session for VBR, likely because of insufficient data. Note that two of the three sessions with negative correlations were short sessions (13 and 12 minutes), and the third session is session 17 (72 minutes).

Table 4.6 on page 120 shows the session length, correlation coefficient r between time and BR minus static-BR-fit, p-value of the correlation coefficient, and slope of the linear best-fit trend-line for each session. For BR, the mean slope over all sessions was 0.036 dB/min, and for BR-hand, the mean slope was 0.078 dB/min. Table 4.7 on page 121 shows the r, p-value, and slopes for VBR minus static-VBR-fit. For VBR, the mean slope was 0.033 dB/min, and for VBR-hand, the mean slope was 0.082 dB/min.

Accuracy Fluctuations

Accuracy of all conditions fluctuated widely over time; this fluctuation in reconstruction accuracy is also seen in prior work by Wu and Hatsopoulos [150]. These fluctuations may arise from several sources: rapid changes in tuning, recording noise, changing mental states of the monkey, or insufficient generality of the tuning model.

To measure the amount of fluctuation, the standard deviation of the one-minute-windowed SNR within each session was measured. The mean (across sessions) of the

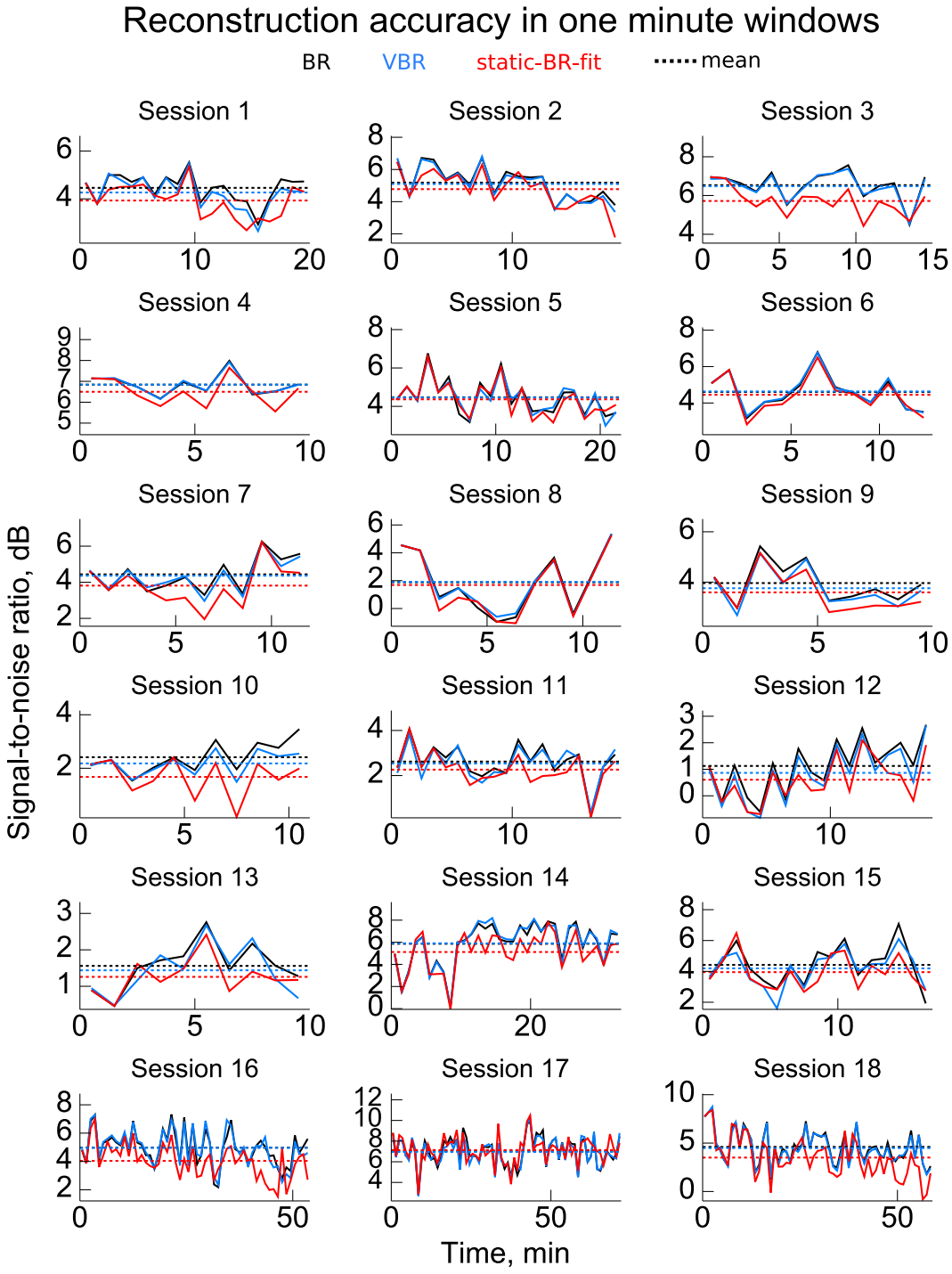

Figure 4.13: Reconstruction accuracy versus time within sessions

Table 4.6: Improvement in reconstruction accuracy versus time with BR. Correlation coefficient and slope of best-fit line calculated between BR improvement over static-BR-fit in one-minute windows and time in minutes. Bold p-values are significant.

Session	Length (min)	Corr. coef. r	p-value	Slope (dB/min)
1	20	0.402	0.079	0.032
2	19	0.185	0.447	0.018
3	15	0.346	0.207	0.044
4	10	0.307	0.388	0.036
5	22	0.146	0.517	0.007
6	13	-0.072	0.815	-0.003
7	12	0.494	0.102	0.075
8	12	-0.142	0.659	-0.014
9	10	0.786	0.007	0.062
10	11	0.819	**0.002**	0.153
11	19	0.229	0.345	0.016
12	18	0.515	0.029	0.045
13	10	0.503	0.138	0.049
14	33	0.432	0.012	0.035
15	17	0.312	0.223	0.046
16	54	0.282	0.039	0.018
17	72	-0.074	0.535	-0.003
18	59	0.538	**0.000**	0.037
Mean	23.7	0.334		0.036

SNR standard deviation for static-BR-fit was 1.09 dB. The mean for BR was 1.06 dB, and the mean for BR-hand was 1.03 dB. The mean for static-VBR-fit was 1.11 dB. The mean for VBR was 1.05 dB, and the mean for VBR-hand was 1.05 dB. Two-sided paired sign-tests ($n = 18$, $\alpha = 0.05$) detected no significant difference between static and self-training update for BR and VBR, between self-training update and hand update, and between static-BR-fit and static-VBR-fit. These results indicate that fluctuations were not affected by using self-training or hand movement updates.

Table 4.7: Improvement in reconstruction accuracy versus time with VBR. Correlation coefficient and slope of best-fit line calculated between VBR improvement over static-VBR-fit in one-minute windows and time in minutes. Bold p-values are significant.

Session	Length (min)	Corr. coef. r	p-value	Slope (dB/min)
1	20	0.178	0.452	0.013
2	19	0.265	0.273	0.028
3	15	0.329	0.232	0.044
4	10	0.399	0.254	0.043
5	22	0.182	0.418	0.011
6	13	-0.071	0.816	-0.004
7	12	0.265	0.406	0.036
8	12	-0.313	0.323	-0.030
9	10	0.780	0.008	0.074
10	11	0.698	0.017	0.096
11	19	0.313	0.192	0.023
12	18	0.513	0.030	0.048
13	10	0.722	0.018	0.087
14	33	0.358	0.041	0.031
15	17	0.422	0.092	0.048
16	54	0.298	0.029	0.019
17	72	-0.046	0.704	-0.002
18	59	0.534	**0.000**	0.036
Mean	23.7	0.324		0.033

BR Versus VBR

The correlation between the difference in accuracy BR minus VBR and static-BR-fit accuracy was calculated to determine if BR performed better than VBR on more noisy or less noisy sessions. The correlation was nominally negative — suggesting the advantage of BR was larger for noisier sessions — but not significant ($r = -0.44$, t-test, $p = 0.069$).

Accuracy Versus Model Drift Parameter

Figures 4.14 on the following page and 4.15 on page 124 show pseudo-colored tables of the improvement in accuracy of BR and VBR versus static (with the respective fitting method) for different values of the model drift parameter. Blue indicates better accuracy than static, white indicates no difference, and red indicates worse accuracy. Note that some red values in the tables saturate the color scale.

The model drift parameter is the amount of variance added to the variance of the tuning model coefficients' distributions before each update. Since the neuronal and hand movement data are whitened to unit standard deviation during parameter fitting and subsequent updating, the values of the model drift parameter have no units. The tested values of the model drift parameter were equally spaced in log scale.

The tables indicate that the optimal model drift for each session is different and can vary widely. However, using the best overall value of $4.5 \cdot 10^{-5}$ results in significant improvement versus static (BR: mean 0.36 dB, two-sided, paired sign-test, $n = 18$, $p < 0.01$; VBR: mean 0.33 dB, $p < 0.01$). The accuracy improvement versus model drift function is generally unimodal within each session, though the optimum model drift may not have been included in the range tested.

Best Model Drift Versus Static Accuracy

One possible explanation for the wide variation in optimal model drift is variation in quality of the training data at the beginning of each session. Sessions with higher quality training data may be reconstructed better with smaller updates. To test this hypothesis, the correlation between the logarithm of the best model drift parameter and the static-BR-fit accuracy (in dB) was calculated. This uses the assumption that higher quality training data produces more accurate reconstructions. The correlation was nominally negative (in support of the hypothesis) but not significant ($r = -0.38$, t-test, $p = 0.12$).

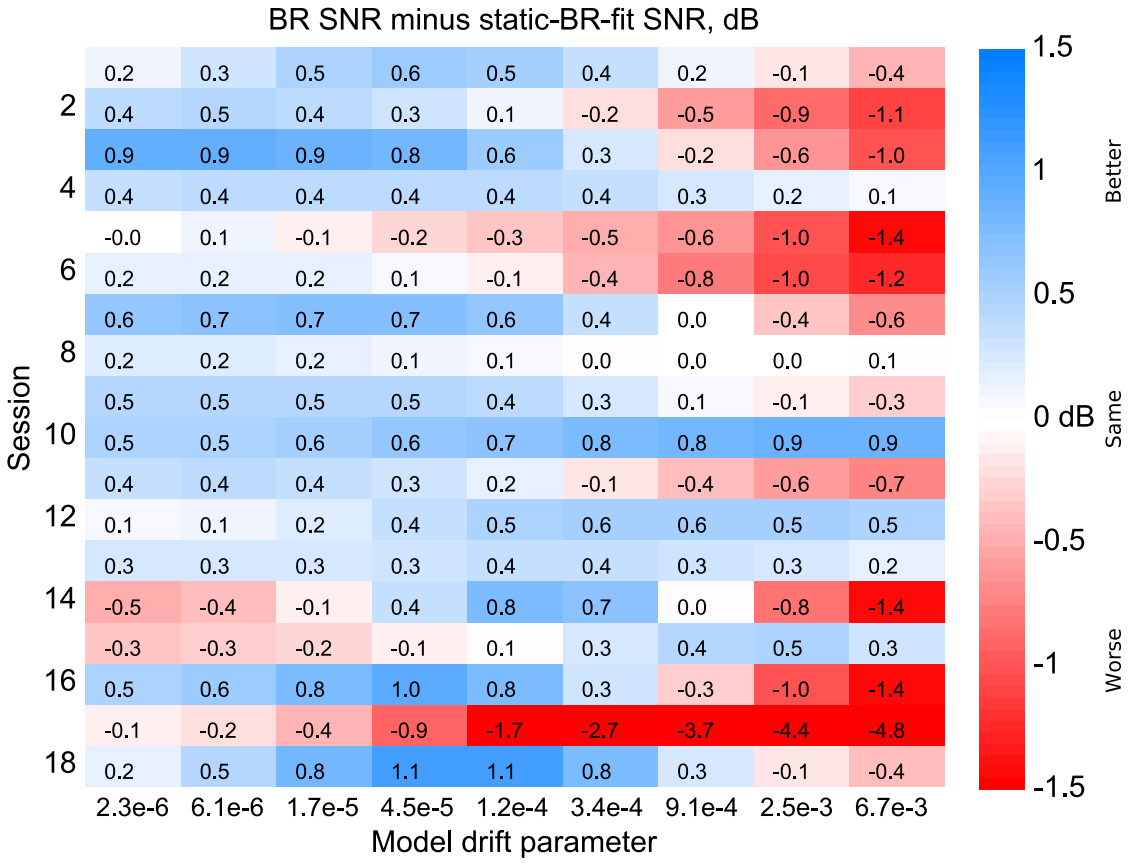

Figure 4.14: BR reconstruction accuracy improvement versus model drift parameter. Larger cell entries indicate more improvement from using BR updates. Note, some values saturate the color scale.

Computational Complexity and Speed

Empirically, the slowest step in the Bayesian regression self-training updates was the Kalman smoothing preprocessing. Each iteration of Kalman smoothing has complexity $O(d^3)$ from matrix inversions, where d is the state dimensionality. $T-1$ iterations of smoothing are needed, making the total complexity $O(Td^3)$.

The analytical calculation for joint distribution Bayesian regression has computational complexity $O(TN^2 + TD^2 + DN^2 + D^3)$, where T is the update batch size, N is the number of neurons, and D is the feature dimensionality. Each *iteration* for the variational inference of the factorized distribution Bayesian regression has

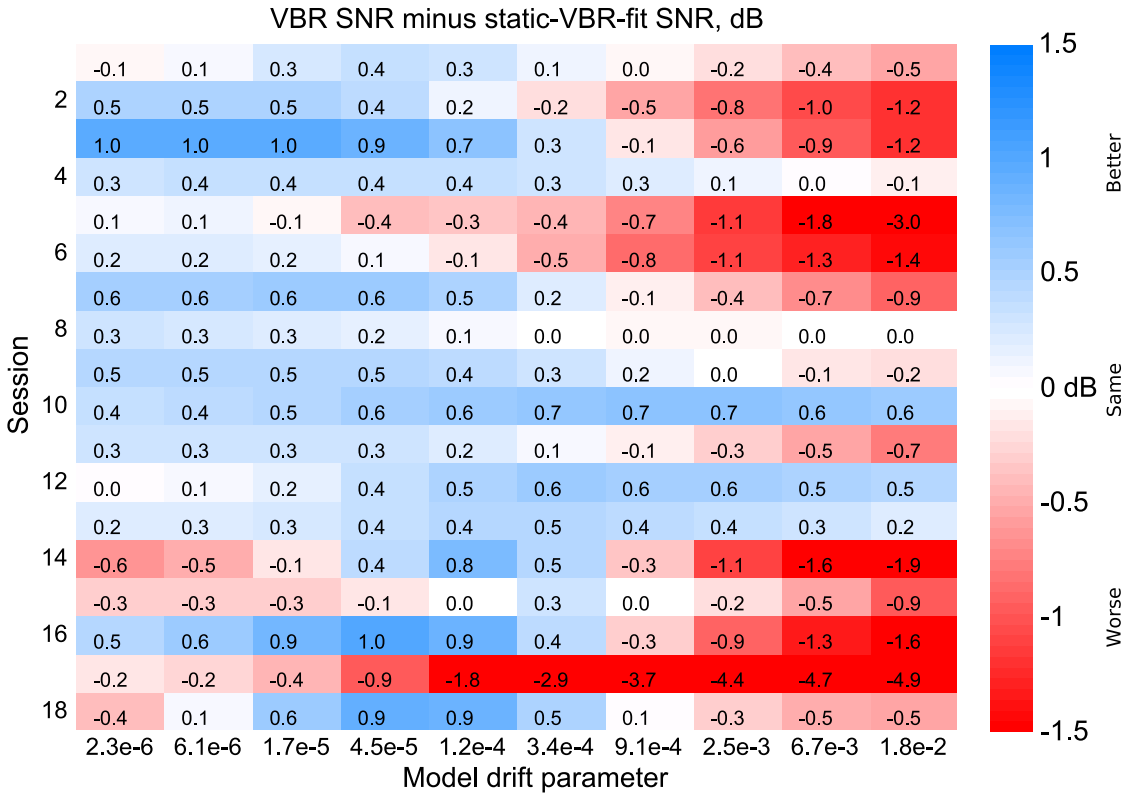

Figure 4.15: VBR reconstruction accuracy improvement versus model drift parameter. Larger cell entries indicate more improvement from using VBR updates. Note, some values saturate the color scale.

computational complexity $O(TN^2 + NTD^2 + N^3 + ND^3)$.

In terms of execution speed, the MATLAB implementation of joint distribution Bayesian regression self-training on an Intel Core i7 class desktop computer used 1.57± 0.02 seconds (mean ± standard deviation) per update. The MATLAB implementation of factorized distribution variational Bayesian regression self-training used 1.97 ± 0.13 seconds per update.

4.3.3 Experiments on Modified Neuronal Data

Off-line reconstructions were performed on a real data session in which several neurons were modified in different ways to demonstrate the adaptive capability of the Bayesian regression self-training method. Three modification were tested. First, Poisson

distributed noise was added to neurons in the training data but not the testing data to simulate neurons which become more tuned. Second, Poisson noise was gradually added to testing data to simulate neurons which become less tuned or are affected by recording degradation. Third, a neuron is gradually replaced by another neuron to simulate a change in tuning.

Simulating Neurons which Become More Tuned

The first experiment tests the ability of Bayesian regression updates to adapt tuning coefficients and tuning noise parameters to neurons which become more tuned. Session 17 (length 72 minutes) described in the previous subsection was used. The first minute of the session was discarded. Two minutes of data after the first minute was used for training data.

Four neurons were chosen from the session based on their comparatively high fano-factor. Homogeneous Poisson noise with mean 100 Hz was added to the four chosen neurons in the training data. No noise was added to the testing data.

Initial tuning parameters were fit with Bayesian regression. The regularization value of $\lambda_h = 8$ was used. Then the 1st order UKF with joint distribution Bayesian regression self-training updates and updates using hand movements were used to reconstruct the remainder of the session. The Bayesian regression updates used model drift parameter of 0 and degrees of freedom cap of 12,000. The update mechanism was not given information about which neurons changed. Instead, all parameters in the tuning model were allowed to change (even though the model drift parameter was 0).

Figure 4.16 on the following page shows the noise standard deviations detected by the updates for each neuron. The solid black curves indicate noise detected by Bayesian regression self-training, and dashed black curves indicate noise detected by Bayesian regression using hand movements. Green horizontal lines indicate the true

values found by using the initial parameter fitting procedure on the testing data. The noise standard deviations converged to the true values slowly, likely due to the large setting of the degrees of freedom cap.

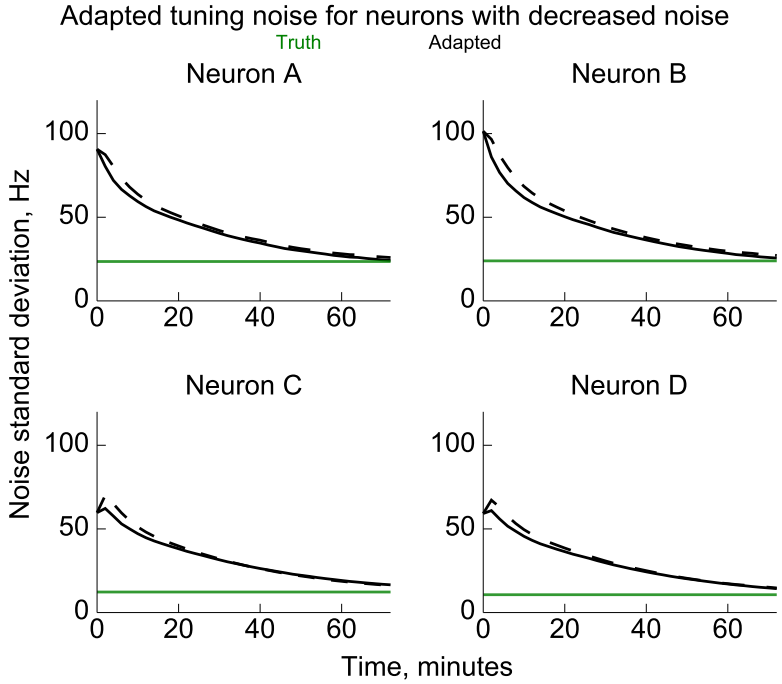

Figure 4.16: Adapted tuning noise for neurons with simulated decrease in noise. Noise was added to neurons in training data but not testing data. Green lines indicate true values of noise standard deviation found by parameter fitting on testing data. Solid black curves indicate values of noise found by Bayesian regression self-training updates. Dashed black curves indicate values found by Bayesian regression updates calculated with hand movements.

Figure 4.17 on page 128 shows the tuning coefficients detected by the updates for each neuron. Kinematic features use these abbreviations: p for position, v for velocity, m for magnitude, and x and y for x- and y-axis. Baseline is the baseline firing rate or bias term in the tuning model. Units for coefficients are standard deviations of binned spike counts divided by standard deviations of kinematic features.

Bayesian regression using hand movements for updates found the true tuning coefficients, as determined by parameter fitting on the testing data. Bayesian regression

using self-training found some of the true coefficients, but did not for vx and vy of neuron A, vx and vy for neuron B, vx for neuron C, and vy for neuron D. These results suggest velocity coefficients are harder to learn than position coefficients.

Simulating Neurons which Become Less Tuned

The second experiment tests the ability of Bayesian regression updates to adapt tuning noise parameters to neurons which become less tuned. The data used is the same as the previous experiment except that homogeneous Poisson noise was added to the four chosen neurons in the testing data. The noise ramped up linearly from 0 to 100 Hz during the time span of the testing data. No noise was added to the training data. Parameter fitting was similar to the previous experiment and $\lambda_h = 8$ was used. Updates used 0 for model drift and 2,400 as the degrees of freedom cap, to improve alertness for changes in noise. All parameters in the tuning model were allowed to change.

Figure 4.18 on page 129 shows the noise standard deviation entries detected by the updates for each neuron. In this experiment, the true values of noise were calculated in twp-minute windows from hand movements using the initial parameter fitting procedure. The Bayesian updates adapted the noise parameter for neurons C and D, but could not keep up with changes to neurons A and B. This may be due to the higher initial values of noise for A and B, thus making the added Poisson noise harder to detect.

Simulating a Neuron which Changes Tuning

The third experiment tests the ability of Bayesian regression updates to adapt tuning coefficients to a neuron which changes tuning. In this experiment, session 17 is used with the first minute discarded, following 6 minutes used as training data, and remainder as testing data. One neuron, neuron C, is gradually replaced with neuron

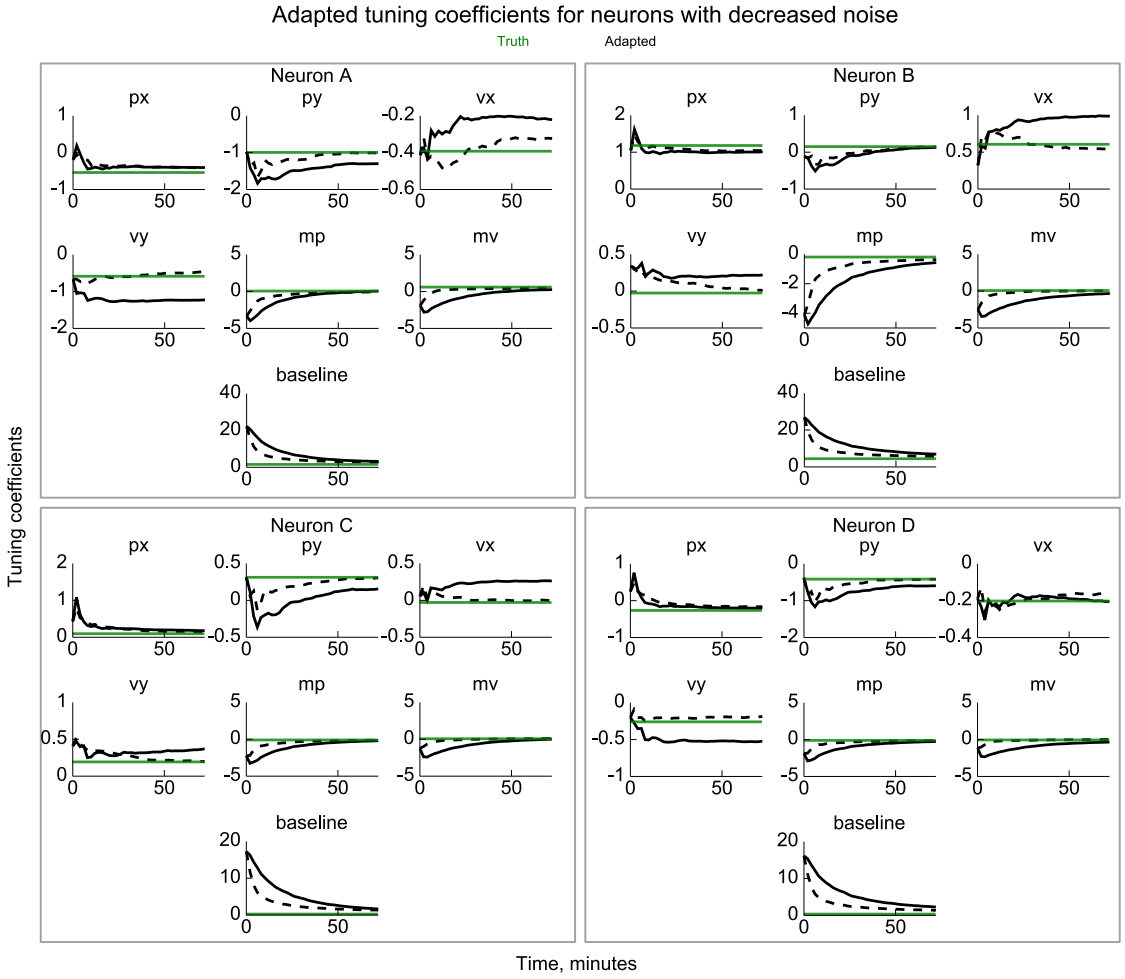

Figure 4.17: Adapted tuning coefficients for neurons with simulated decrease in noise. Noise was added to neurons in training data but not testing data. Green lines indicate true values of tuning coefficients found by parameter fitting on testing data. Solid black curves indicate values of tuning coefficients found by Bayesian regression self-training updates. Dashed black curves indicate values found by Bayesian regression updates calculated with hand movements. Kinematic features use these abbreviations: p for position, v for velocity, m for magnitude, and x and y for x- and y-axis. Units for coefficients are standard deviations of binned spike counts divided by standard deviations of kinematic features.

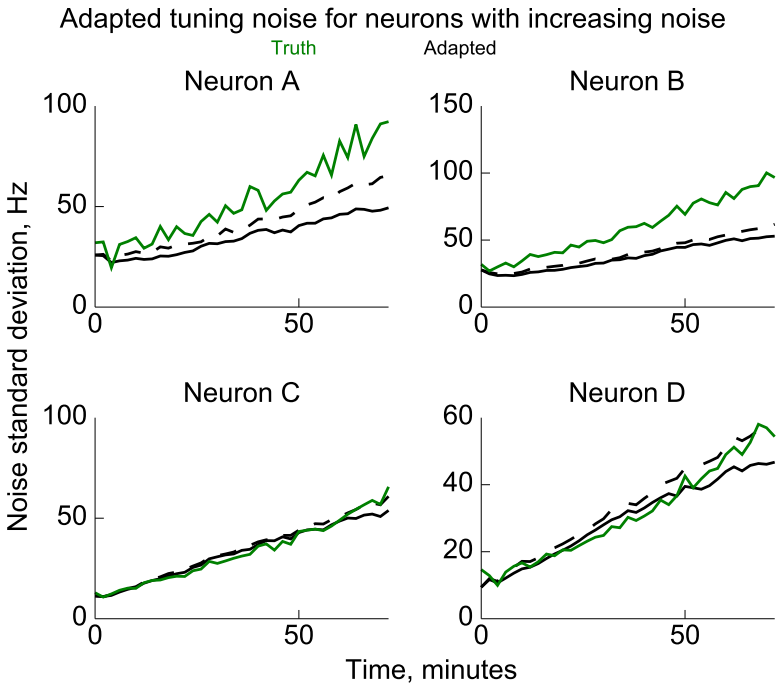

Figure 4.18: Adapted tuning noise for neurons with simulated gradual increase in noise. Noise was gradually added to neurons in testing data but not training data. Green lines indicate true values of noise found by parameter fitting on two-minute windows of testing data. Solid black curves indicate values of noise found by Bayesian regression self-training updates. Dashed black curves indicate values found by Bayesian regression updates calculated with hand movements.

D using a linear interpolation:

$$\hat{y}_C(t) = (1 - a(t)) \cdot y_C(t) + a(t) \cdot y_D(t) \tag{4.11}$$

$y_C(t)$ and $y_D(t)$ are the original binned spike counts of neurons C and D at time t, respectively, and $a(t)$ is a gain which increases linearly from 0 to 1 through the testing data interval. Neuron C was not modified in the training data. Neuron D was removed from the training and testing data. The initial parameter fit was performed with $\lambda_h = 7$, a smaller value since there was more training data. The Bayesian regression updates used a model drift parameter of 10^{-5} and degrees of freedom cap of 12,000. All parameters in the tuning model were allowed to change.

Figure 4.19 on the next page shows the tuning coefficients detected by the updates for neuron C. The true values of the parameters were calculated in two-minute windows from hand movements using the initial parameter fitting procedure. The true values show large variance due to the small size of the window.

Bayesian regression self-training updates were able to follow the changes in position and velocity coefficients, but position magnitude, velocity magnitude, and baseline firing rate coefficients were not adapted correctly. This was likely due to non-exciting data: the relative contributions of position magnitude, velocity magnitude, and baseline firing rate could not be separated with the given testing data. Thus, two of these coefficients were overestimated and one was underestimated in a compromise. Updates using hand movements were able to follow changes in all tuning coefficients.

4.3.4 Closed-Loop Control

Eleven on-line sessions of closed-loop brain-control were conducted over 29 days with one monkey to evaluate the Bayesian regression self-training method. This series of experiments allowed updates over a longer time span than possible with off-line reconstructions on single sessions. The factorized distribution variational Bayesian regression was used because neurons could be flagged inactive and omitted from updates and filtering. Parameters of inactive neurons did not pass through the parameter transition model. Inactive neurons were the result of recording instability, and the inactivity status usually changed only between sessions.

Neuron Inactivity Criteria

Flagging of inactivity was based on two tests. First, if a neuron produced no spikes during the self-training data window, it was flagged inactive. Second, a likelihood-ratio test was employed during filtering. In this test, the product over the previous 10 seconds of the likelihood ratio between two models was calculated and compared

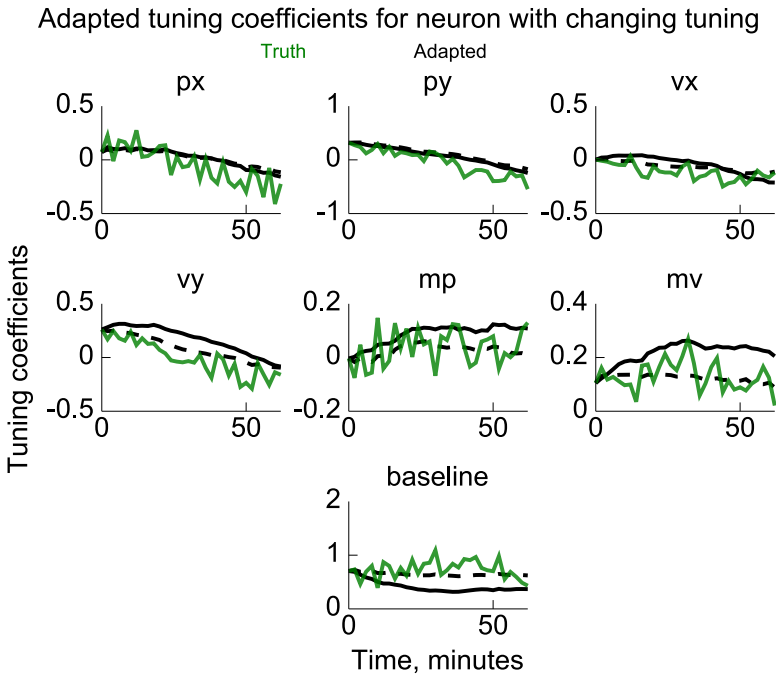

Figure 4.19: Adapted tuning coefficients for neuron with simulated gradual change in tuning. Spike counts of one neuron were gradually replaced by spike counts of another neuron through a linear combination. Green lines indicate true values of tuning coefficients found by parameter fitting on two-minute windows of testing data. Solid black curves indicate values found by Bayesian regression self-training updates. Dashed black curves indicate values found by Bayesian regression updates calculated with hand movements.

to two thresholds. In the numerator of the ratio is the *active model*. In this model, the filter innovation — the observation minus the predicted observation — for neuron i follows a normal distribution with zero mean and variance equal to the (i, i) entry of $\mathbf{P}_{zz,t}$. The second model is the *inactive model*, in which the observation for neuron i follows a normal distribution with zero mean and the same variance as the active model. If the product of the likelihood-ratio over the previous 10 seconds fell below 10^{-10}, the neuron was flagged inactive. If the product of the likelihood-ratio rose above 1, the neuron was flagged active. The thresholds were set by hand to ensure a low false-positive rate for inactivity.

Initial Parameter Fitting

During the first session, 2 minutes of behavior during hand-controlled point-to-point pursuit task was used to perform the initial parameter fit with VBR. The prior variance of the model coefficients used for regularization was the value found to maximize accuracy in off-line reconstructions.

Spike-Sorting

Before the first session, waveform templates were set by a human expert (not the author) using spike-sorting software made by Plexon. The waveform templates were subsequently not changed during the duration of the experiments. All sorted units were used and no effort was taken to select stable units for use by the decoder, in contrast to the approach taken by Ganguly and Carmena [31].

The number of sorted units was 139. Two units did not fire during the 2 minutes of parameter fitting data and thus were flagged inactive from the start. One of these units fired sporadically in other sessions, but with such low frequency that it did not contribute towards decoding. The other initially inactive neuron remained inactive.

Experiment Procedure

During each session, the monkey controlled the cursor in the point-to-point pursuit task using the 5-th order UKF. The joystick was present and hand movements were observed, but joystick recordings were completely ignored. In each session, brain-control was first performed with the UKF using the initial parameters fit on the first session, and VBR self-training was turned off. In other words, the tuning model parameters were always the same during this portion of the experiment. This period of brain-control lasted for 6 to 10 minutes and was used for comparison as the *static* condition.

Then, brain-control was performed for 10 to 50 minutes using the UKF with VBR

self-training updates turned on. The variation in time duration was due to variations in the monkey's motivation to perform the behavioral task. In this condition (VBR), the results of updates were kept between sessions, i.e. the parameters from the last update at the end of a session were used as the beginning parameters and updated further in the VBR condition in the next session. In contrast, the static condition always used the same parameters (from the initial fit), and never used updated parameters.

Self-Training Parameter Settings

The VBR self-training updates occurred after 2 minutes of self-training data were cached. The model drift parameter varied between 10^{-3} and 10^{-5}; larger values were used at the beginning of each session — compensating for the long time period since the previous update — and smaller values were used after a few updates. The degrees of freedom cap was set at 12,000 for all updates.

Updating Baseline Firing Rate Parameters

Previous studies showed that the baseline firing rate of neurons changed significantly over time [12, 31]. To evaluate how well updating only the baseline firing rate parameters maintains brain-control accuracy, during the last 2 session of the experiment, the parameters for baseline firing rate in the static condition were updated by taking an exponential moving average of the observed binned spike counts. The half-life of the exponential moving average varied between 1 and 3 minutes.

Omitted Data

Since the target was larger than the cursor, the amount of juice reward given to the monkey was set to vary by how close the cursor was to the center of the target, in order to promote accurate pursuit. Thus, it is assumed that the monkey was trying to keep the cursor in the center of the target as well as possible given the brain-control

used. However, as the monkey was not head-posted and periodically looked away from the screen and stopped performing the task, periods during which the joystick was not touched or was not moving for longer than 3 seconds were ignored in the evaluation of control accuracy. Caching of self-training data was paused when the monkey was not holding the joystick. The monkey was also observed to hold and move the joystick while not looking at the screen, but this occurred infrequently.

Evaluation Metrics

The ability of the monkey to control the cursor under the two conditions was measured by the signal-to-noise ratio of the position trajectory of the cursor relative to the position trajectory of the continuously moving target (the desired signal). The SNR for the x- and y-axis were averaged.

SNR was calculated over the entire interval of the control condition and in one-minute segments within the interval. The overall SNR and the best SNR among one-minute segments are reported. The latter measure has three desirable qualities. First, it ignores the decrease in monkey motivation over the course of a session. Second, it ignores the increase in accuracy due to practice effects at the beginning of each session. Third, it ignores bad performance caused by monkey inattention that were not caught by the data omission criteria described above. The one-minute windows were long enough such that accurate pursuit was unlikely to occur by chance. However, some of the accuracy fluctuations may be due to poor control in particular portions of the work-space.

Results: Overall Trends

Figure 4.20 on the following page shows the mean and best pursuit accuracy for the 11 sessions over 29 days. Red indicates the static condition and blue indicates the VBR self-training condition. Solid lines indicate the mean pursuit accuracy and dotted

lines indicate the best accuracy among all one-minute windows within the condition. Data for the static condition was not available on day 6 because the control was so poor that the monkey was not willing to perform the task for more than a few seconds.

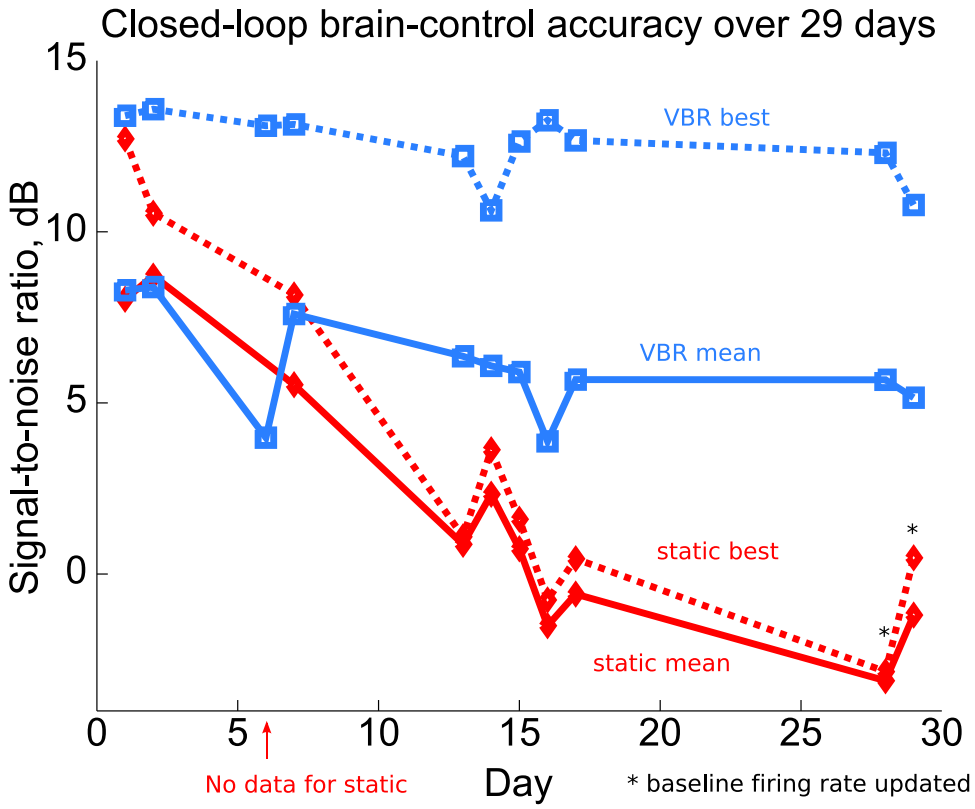

Figure 4.20: Closed-loop brain-control accuracy of UKF with VBR updates. Solid curves indicate mean SNR of each session. Dotted curves indicate best SNR among one-minute windows within each session. During the last two sessions, baseline firing rate parameters of the static UKF were updated using exponential moving average.

VBR self-training reduced performance loss compared to not updating. Correlations between accuracy and time, p-values of correlations, and slopes of best-fit linear trend-lines are shown in Table 4.8 on the next page.

Table 4.8: Closed-loop brain-control accuracy versus time. Correlation coefficient and slope of best-fit line calculated between accuracy for each session and time measured in days. Bold p-values are significant.

Condition	Corr. coef. r	p-value	Slope (dB/day)
VBR mean	-0.539	0.087	-0.088
VBR best	-0.647	**0.032**	-0.070
static mean	-0.913	**<0.001**	-0.399
static best	-0.883	**<0.001**	-0.487

Baseline Firing Rate Updates

During the last 2 days under the static condition, the baseline firing rate parameters of the UKF were updated. Despite this update, control was still poor and substantially worse than the control provided by the VBR updated UKF. This demonstrates that updating the baseline firing rate parameters alone is insufficient to cope with changes in neuronal tuning.

Results: Trends Within Sessions

Figure 4.21 on the following page shows the pursuit accuracy during control with the VBR updated UKF in one-minute windows within each session. Similar to accuracy in off-line reconstructions (Figure 4.13 on page 119), accuracy during closed-loop brain-control fluctuates widely. During several days (6, 7, 17, 28), performance at the beginning of the VBR condition was poor. However, control improved after a few updates, demonstrating the ability of VBR self-training to adapt to changes in tuning which had occurred since the previous session.

Changes in Tuning Coefficients

The changes made to the tuning coefficients over the 29 days by the Bayesian regression self-training updates were examined by comparing the values from the initial fit to

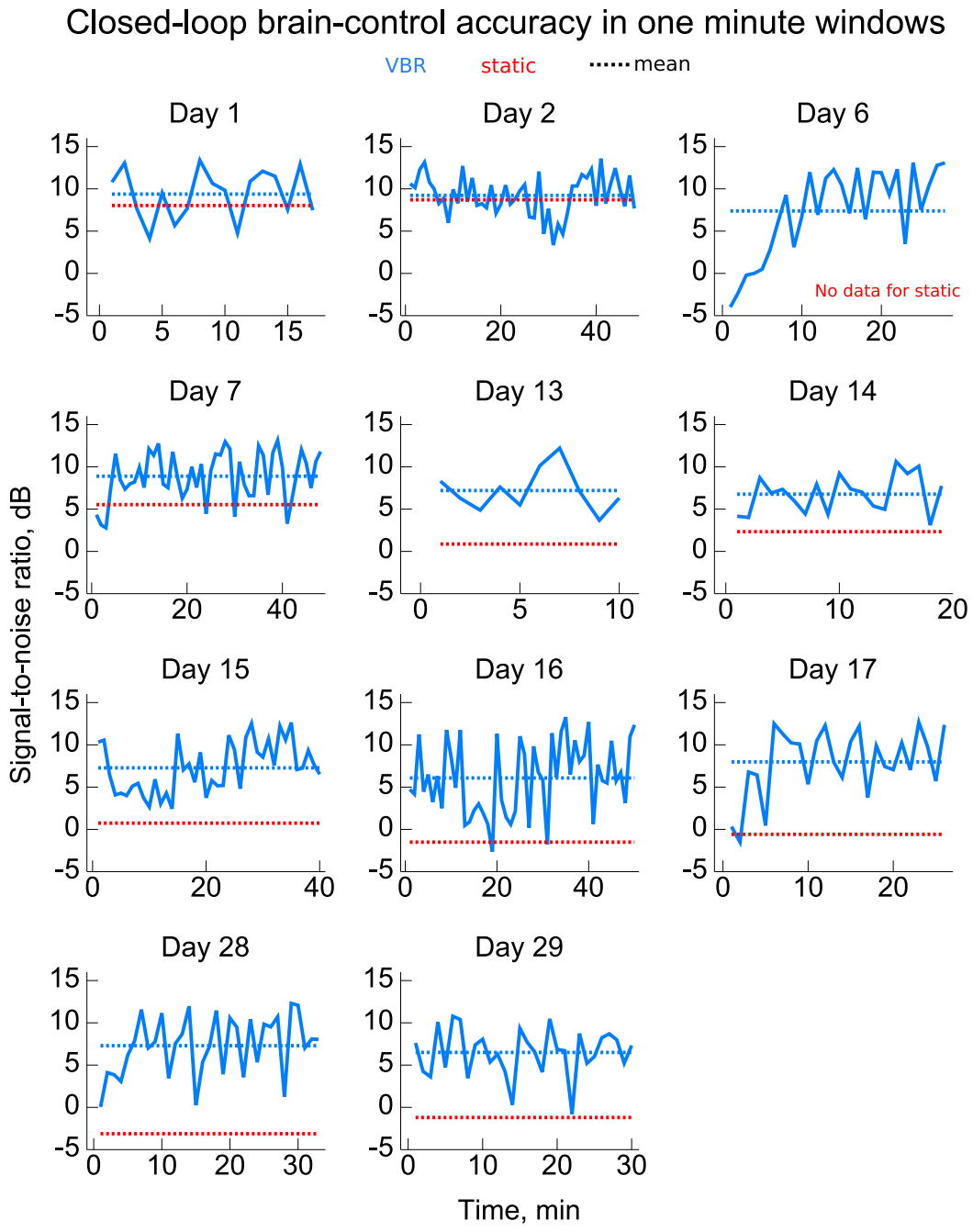

Figure 4.21: Closed-loop brain-control accuracy versus time within sessions. In several sessions control accuracy was poor initially, but improved after a few updates.

the values at the beginning of day 29. The values at the end of day 29 were lost due to a technical difficulty unrelated to the algorithms. All figures show changes in parameters as final values minus initial values.

Figure 4.22 shows the distribution of the changes in the mean of the tuning coefficients (μ_i). Since the kinematic features and binned spike counts were both normalized to unit standard deviation, the units of the x-axis are standard deviations of binned spike counts divided by standard deviations of kinematic features. The distribution has a sharp peak at zero with a long tail, and the mean and median, indicated by the vertical lines, were both very close to zero. Figure 4.23 on the next page shows the distribution of the absolute values of the changes. The median change of approximately 0.1 standard deviations per standard deviation is substantial.

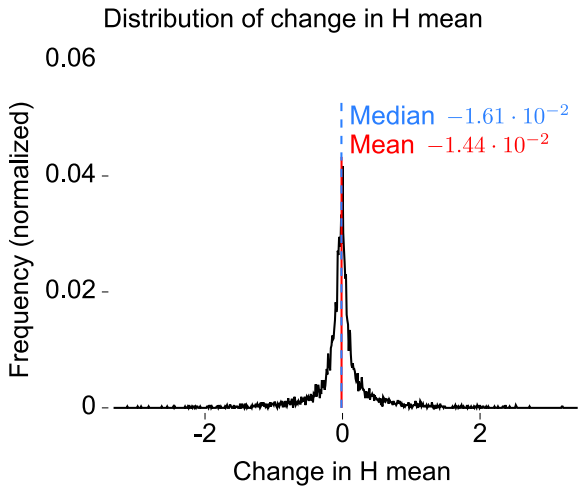

Figure 4.22: Distribution of changes in tuning coefficients. Units of the x-axis are standard deviations of binned spike counts divided by standard deviations of kinematic features.

Figure 4.24 on page 140 shows a pseudo-color table of changes in tuning coefficients for low noise neurons. Low noise neurons are defined to be those with final tuning model noise variance less than 0.5 and non-zero baseline firing rate. Each row shows the tuning coefficients for one neuron. Columns are kinematic features, grouped by

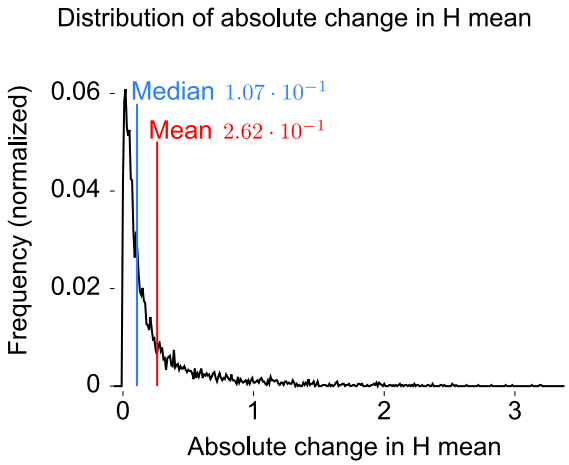

Figure 4.23: Distribution of absolute changes in tuning coefficients. Units of the x-axis are standard deviations of binned spike counts divided by standard deviations of kinematic features.

time tap. The kinematic feature labels use the abbreviations p for position, v for velocity, and m for magnitude. A +100 ms offset indicates the kinematic features at time 100 ms after the end time of the bin for spikes. Color and saturation indicate direction and magnitude of change, with blue indicating increase and red indicating decrease. Some entries saturate the color scale. Units used here are the same as those of the previous two figures.

Several neurons (58, 82, 88, 91, 107, 136) show large changes in tuning coefficients. Most large changes are in position terms.

Figure 4.25 on page 141 shows the distribution of the changes in the diagonal precision entries of the tuning coefficients. Recall that the factorized distribution for the tuning model parameters includes a separate multivariate normal distribution for the tuning coefficients of each neuron. Included in Figure 4.25 are the diagonal entries of the precision matrices ($\mathbf{\Lambda}_i$) for these normal distributions. Units of the x-axis are standard deviations of kinematic features squared divided by standard deviations of binned spike counts squared. Higher values indicate higher certainty. Note, a few values of precision change were larger than the $8 \cdot 10^4$ x-axis limit. Both the

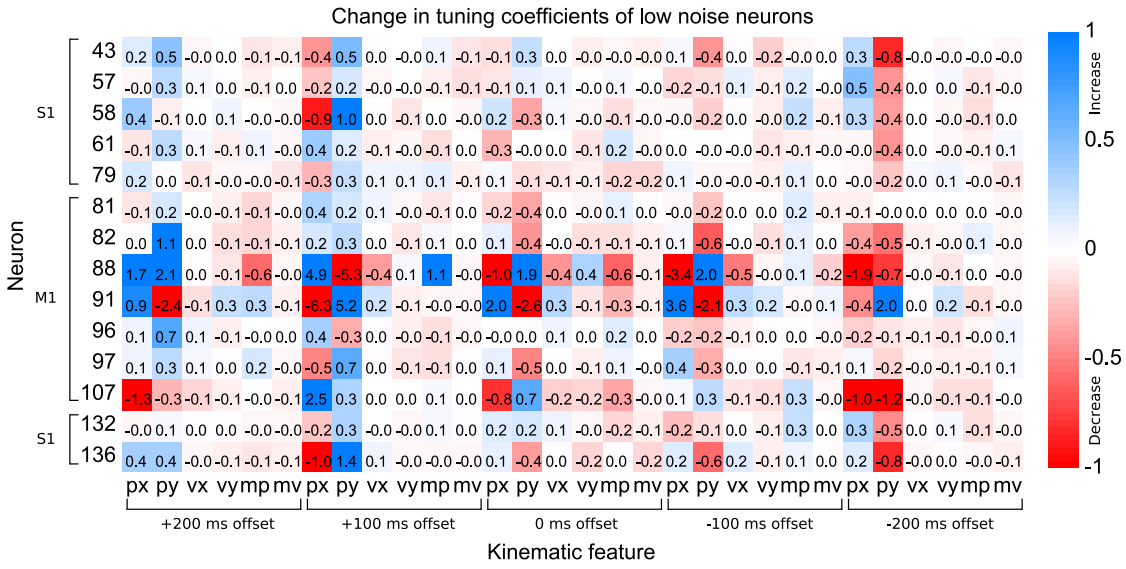

Figure 4.24: Changes in tuning coefficients of low noise neurons. Neurons with tuning error variance less than 0.5 are included. Each neuron's coefficients make up one row. Each tuning feature is a column, and the abbreviations are: p for position, v for velocity, m for magnitude, x for x-axis, and y for y-axis. Features are grouped by time tap. Units in the color scale are standard deviations of binned spike counts divided by standard deviations of kinematic features. Note some values saturate the color scale.

mean and median of the distribution are larger than zero, indicating the uncertainty of the tuning coefficients was generally reduced. However, the uncertainty of some coefficients increased over the course of the on-line experiments.

Changes in Baseline Firing Rate Parameters

Figure 4.26 on page 142 shows the initial and final baseline firing rate parameters for each neuron. Arrows show direction of change, with the triangle centered at the final value and the arrow starting at the initial value. Decreased values are indicated by red lines, and increased values are indicated by blue lines. Values which did not change are indicated by black diamonds. Baseline firing rate parameters generally increased, with a mean increase of 5.66 Hz, median increase of 3.08 Hz, and about 65%

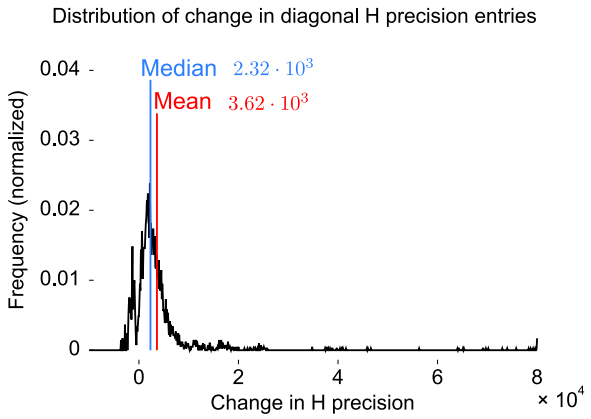

Figure 4.25: Distribution of changes in coefficient precision. Only diagonal entries of the precision matrices for the tuning coefficients of each neuron are included. Units of the x-axis are standard deviations of kinematic features squared divided by standard deviations of binned spike counts squared.

of neurons showing an increase. Several neurons had near zero firing rates. Neurons 14 and 123 had exactly zero baseline firing rates initially, and neurons 8, 80, and 123 had exactly zero baseline firing rates at the end.

Changes in Tuning Model Noise Parameters

Figure 4.27 on page 143 shows a pseudo-color plot of the change in the tuning model noise covariance matrix. The difference is calculated between the mean of the distributions for initial and final noise covariance matrices. The color scale indicates the difference, with blue indicating an increase (higher uncertainty) and red indicating a decrease (lower uncertainty). Note some entries saturate the color scale. Figure 4.27 shows that the Bayesian regression updates changed many off-diagonal covariance entries, which are important for decoding accuracy, as shown in the analysis of the UKF in the previous section.

To examine the changes in the noise variance, i.e. the diagonal entries of the covariance matrix, the initial and final diagonal entries are compared in Figure 4.28.

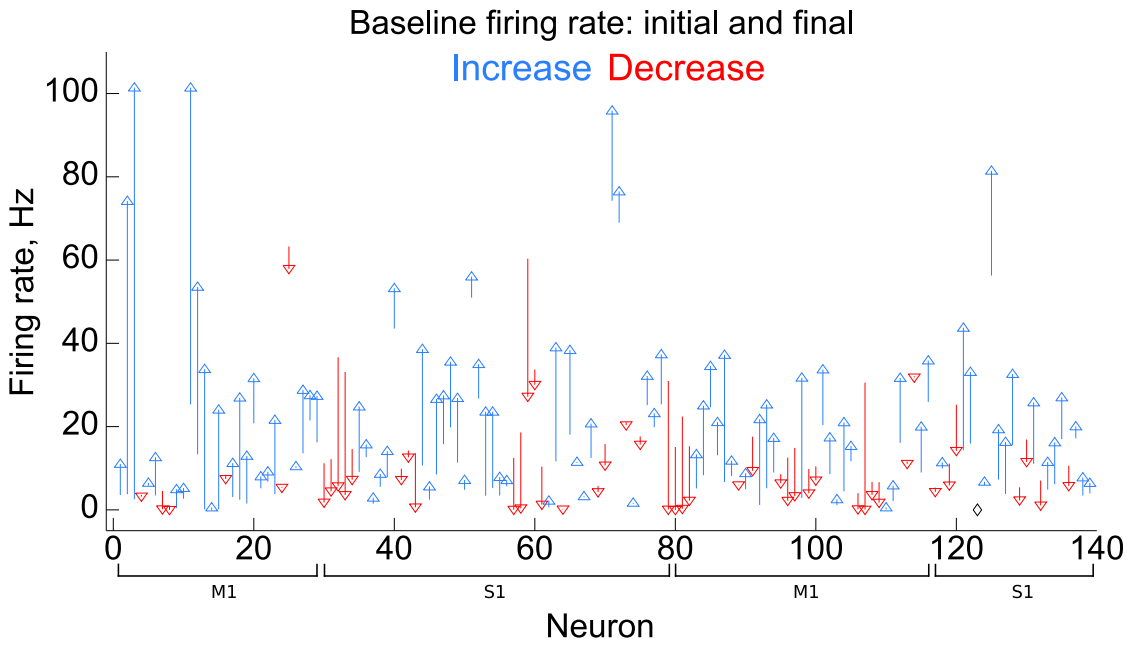

Figure 4.26: Comparison of initial and final baseline firing rate parameters. Arrows show direction of change, with the triangle centered at the final value and the arrow starting at the initial value. Values which did not change are indicated by black diamonds.

Symbols used are similar to those in Figure 4.26. Y-axis units are standard deviations of binned spike counts squared. Some entries saturate the y-axis. Black fillings indicate neurons with zero initial or final baseline firing rate; these neurons were not used by the UKF. About 71% of diagonal entries of R increased, with mean increase 4.13 and median 0.36. The largest increase was to 212. Several entries were very close to zero at the end of the experiments.

Shrinking Observation Noise Variance

During the last few sessions, it was observed that the UKF predictions had near zero variance. Figure 4.28 shows that several highly-tuned neurons had noise variance parameters very close to zero. Observations of these neurons were regarded as highly accurate, which led to low variance in the filter estimates.

This phenomenon is likely caused by the self-training procedure. In a typical

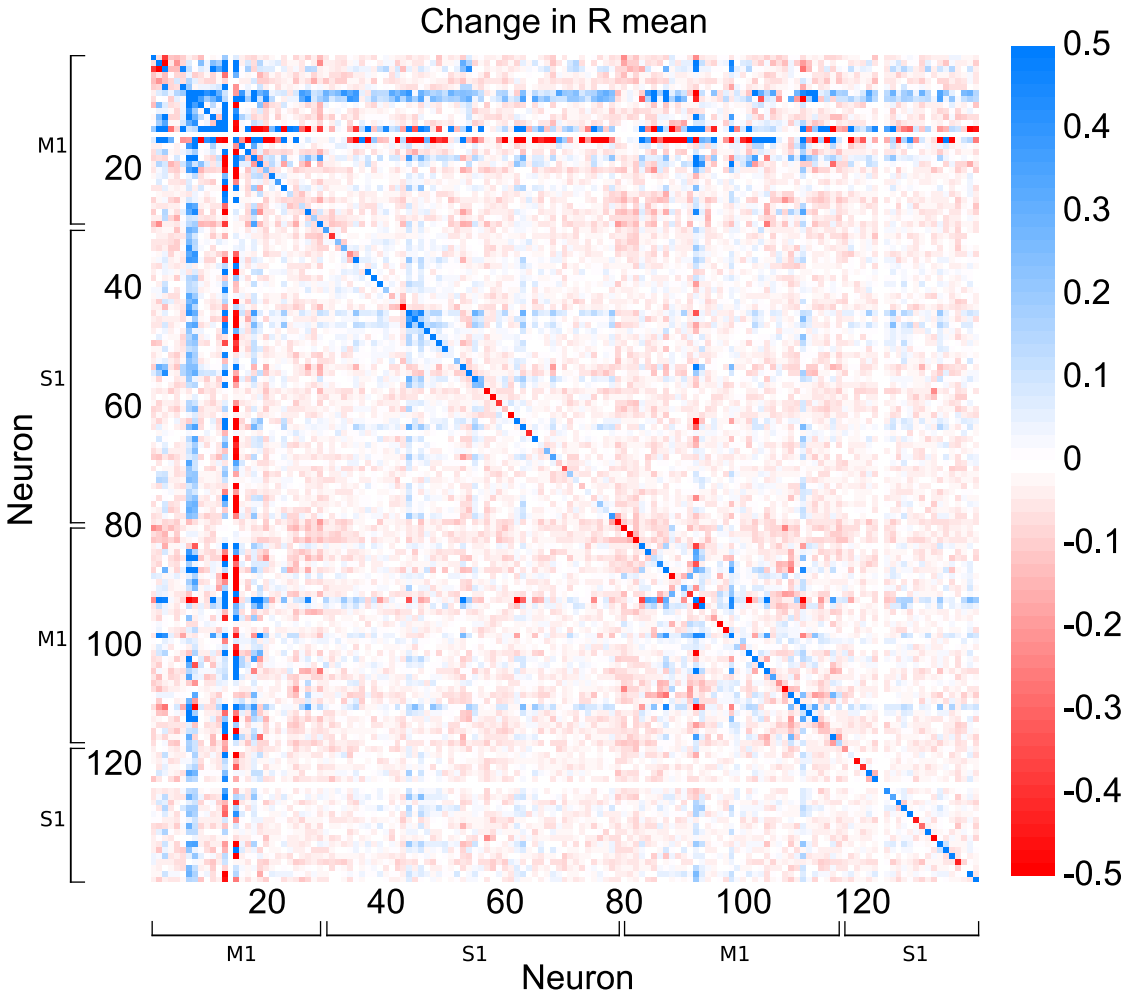

Figure 4.27: Change in tuning noise covariance matrix. Blue indicates increase and red indicates decrease. Note some values saturate the color scale.

population of recorded neurons, some neurons will be relatively more tuned, as indicated by lower observation noise variance in the initial parameter fit. The filter predictions will more closely correlate with the changes in firing rate of these neurons. Since the predictions of the filter are used as training data during self-training updates, the updates will infer low noise for these neurons and decrease their noise variance terms further. This phenomenon continues in a vicious cycle, reducing the observation noise variance of well-tuned neurons and increasing the disparity between well-tuned

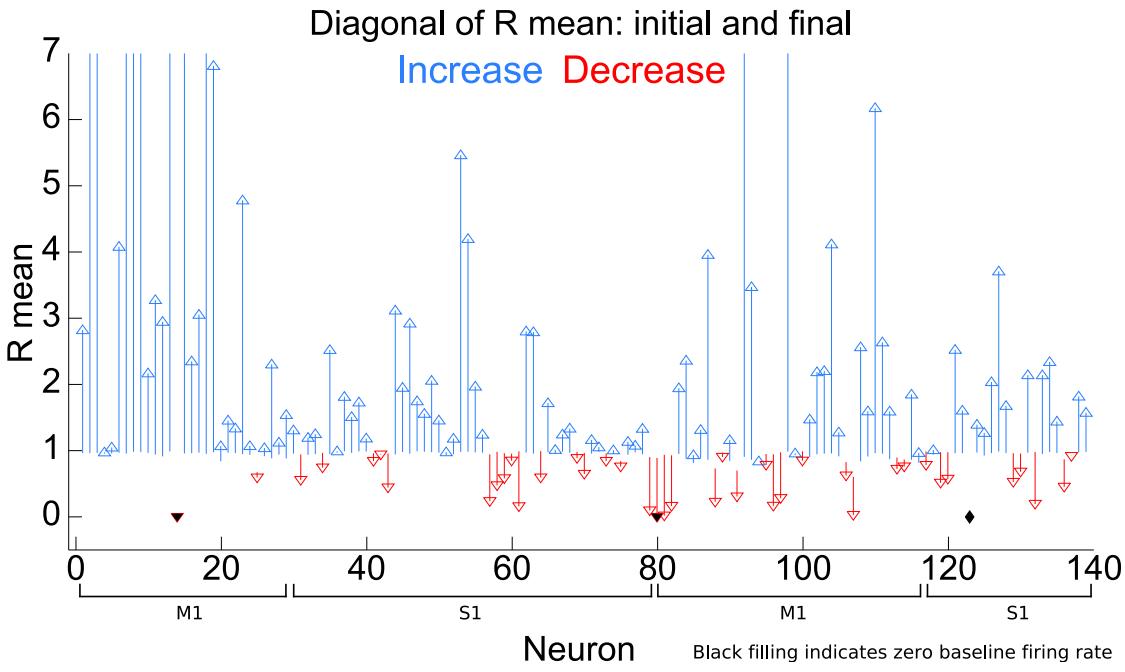

Figure 4.28: Comparison of initial and final tuning noise variance. Arrows show direction of change, with the triangle centered at the final value and the arrow starting at the initial value. Values which did not change are indicated by black diamonds. Y-axis units are standard deviations of binned spike counts squared.

neurons and poorly-tuned neurons. This situation is disadvantageous because relying on a few neurons poses higher risks of performance degradation should any of those neurons be lost [31]. Furthermore, the information bandwidth of the rest of the recorded population is wasted.

Considering the uncertainty of the predictions during the update may reduce this phenomenon. This approach, using error-in-variables regression, is discussed in Section 6.2.

Lower Bound on Observation Noise

Another method to mitigate the above phenomenon is to place lower bounds on the observation noise variance. This stops the filter from "trusting" any neuron completely.

Lower bounds are also useful for avoiding instability. When neurons have very small observation noise variance, the filter produces near zero variance predictions, which lead to singular state covariance matrices that cannot be inverted in the Kalman smoother.

Chapter 5

Adaptive Spike-Sorting

Spike-sorting is an important preprocessing step before decoding, and its accuracy has been shown to affect decoding accuracy [144]. Prior work has reported that spike waveforms slowly change over time [20, 76]. Therefore, to maintain accurate decoding, spike-sorting parameters must adapt to these changes.

This chapter first reviews prior work in updating spike-sorting. Then the variational Bayesian Gaussian mixture clustering approach to updating spike-sorting is presented. The chapter finishes with experiments of the proposed method on synthetic data and on neuronal data collected from a monkey.

5.1 Prior Work

Spike-sorting typically involves an on-line prediction component and an off-line fitting component. The on-line component is the categorization of spike waveforms based on parameters set in the off-line fitting. In some commercial software (e.g. by Plexon), templates for the spike waveforms of each neuron are matched against incoming waveforms using a sum of squared difference metric, because of the simplicity and speed of this operation. Other, more sophisticated, categorization algorithms may also be used.

Templates for template-matching or parameters for categorization algorithms for spike-sorting are fitted from labeled spike waveforms in the off-line component, which typically occurs at the beginning of an experiment involving brain-control. The spike labels are either set by human experts using specialty software or automatically found using some unsupervised clustering technique, using the assumption that each cluster corresponds to a neuron. Once spikes are labeled, templates can be made by calculating the mean waveform for each neuron, and parameters for categorization algorithms can be fitted on the labeled spikes.

Though spike-sorting refers to the entire process of finding and applying templates or parameters, the difficult portion is the off-line labeling of waveforms required to fit the templates or parameters. Thus, in the sequel, "spike-sorting" will usually refer to this off-line labeling problem.

5.1.1 Clustering for Spike-Sorting

Human labeling of waveforms is time-consuming, especially for large populations of neurons. The alternative approach is to cluster spike waveforms and assume each cluster corresponds to a neuron, or *single unit*. This assumption is not always valid, as clusters may correspond to groups of neurons with indistinguishable waveforms, called *multiunits*.

Much research has been done to find practical clustering techniques for spike-sorting (See Lewicki 1998 [67] for a review of early work). A standard approach is to perform maximum-likelihood inference assuming Gaussian mixture models (GMM) using the expectation-maximization (EM) [41, 67, 90, 132] algorithm. Recent approaches include superparamagnetic clustering by Quiroga et al. [91], infinite Gaussian mixture model clustering with Markov Chain Monte Carlo by Wood et al. [145], mean shift clustering by Yang et al. [152], and variational Bayesian Gaussian and t-distribution mixture clustering by Takekawa et al. [119].

5.1.2 Updating Clustering

Several studies have investigated how to update clustering in the face of waveform changes.

Emondi et al. [26] used several metrics on mean waveforms recorded on tetrodes — electrodes with four closely spaced channels — to match clusters between recording sessions. Sorting was performed by hand and two clusters on the same channel in consecutive sessions were determined to be the same neuron if the similarity metric between them passed a threshold. To avoid false positives, the threshold was computed using similarity metrics among different neurons on the same channel in one session. Loss and discovery of neurons were handled using rules based on the similarity metrics.

Bar-Hillel et al. [3] proposed a batch-mode method for tracking changing waveforms. The recording session was first divided into many short time frames. Clustering was performed within each time frame independently using EM, with repeats for different values of the number of clusters. Then a model of how clustering may probabilistically transition between time frames, including gain and loss of clusters, was applied, and the Viterbi path of clusterings among all time frames was found. Then, EM clustering in each time frame was repeated using the Viterbi path values in the neighboring time frames as the initialization. Finally, a new Viterbi path was found from the new time frame clusterings and post-processing was performed.

Wolf and Burdick [142] used a Bayesian initialization for EM clustering for updating spike-sorting. The EM procedure was extended to include a prior term for the cluster locations, while priors for cluster covariances and mixing probabilities were diffuse. Loss and discovery of neurons were handled using similarity-based rules.

Gasthaus et al. [34] presented a generalized Polya urn Dirichlet process mixture model approach to tracking spike waveforms. The proposed method built upon the

infinite Gaussian mixture model spike-sorting of Wood et al. [145] by adding the generalized Polya urn Dirichlet process, a prior for assigning labels to objects. This prior offers a probabilistic model for inferring loss and discovery of neurons. Gasthaus et al. also considered refractory times in their spike-sorting model. Changes in waveforms were handled using a transition model on the cluster parameters. Bayesian inference was performed sequentially, as spikes arrive, using a particle filter.

Gauvain and Lee [35] applied Bayesian clustering updates on Gaussian mixture models for solving the unrelated problem of speaker identification in voice recordings. The Bayesian clustering updates presented here take a similar approach as their work.

5.2 VBGM Clustering Updates

The variational Bayesian Gaussian mixture (VBGM) clustering update method uses variational Bayesian (VB) clustering on Gaussian mixture models [2, 7] with the priors set to the previous clustering parameters and the updated clustering parameters inferred as the posteriors. Additionally, a transition model is applied to the clustering parameters to account for their time-varying nature.

VBGM clustering is a Bayesian extension of expectation-maximization [19] (EM) clustering on GMM which allows priors to be placed on cluster parameters. EM clustering on GMM is equivalent to VBGM clustering with uninformative priors [7]. See Section 3.5.1 on page 67 for a brief review of VB.

5.2.1 Gaussian Mixture Model

In a Gaussian mixture model, each data point arises from one of K mixture components. The probability of the data point coming from each component k is parameterized by the probability π_k, called the *mixing coefficient*. Each mixture component is a multivariate normal distribution with mean $\mathbf{M}_k$ and precision $\mathbf{\Lambda}_k$.

Under this model, the likelihood of a spike s having waveform features $\mathbf{g}(s)$ is:

$$P\left(\mathbf{g}(s)|\pi_{1\ldots K}, \mathbf{M}_{1\ldots K}, \mathbf{\Lambda}_{1\ldots K}\right) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{g}\left(s\right)|\mathbf{M}_k, \mathbf{\Lambda}_k^{-1}\right) \tag{5.1}$$

$\mathbf{g}()$ is a function which generates a column vector of clustering features from the spike. The conditional probability that a spike belongs to a neuron is:

$$P\left(\eta(s) = k|s, \pi_{1\ldots K}, \mathbf{M}_{1\ldots K}, \mathbf{\Lambda}_{1\ldots K}\right) = \frac{\pi_k \mathcal{N}\left(\mathbf{g}\left(s\right)|\mathbf{M}_k, \mathbf{\Lambda}_k^{-1}\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\mathbf{g}\left(s\right)|\mathbf{M}_j, \mathbf{\Lambda}_j^{-1}\right)} \tag{5.2}$$

$\eta()$ is the function which maps a spike to the index of the neuron which generated it.

Features

The feature generating function $\mathbf{g}()$ reduces the dimensionality of the spike waveform from about 40 voltage samples to a small number of features, often 2 for visualization purposes. $\mathbf{g}()$ often combines some preprocessing steps, such as spike alignment, Fourier transform, or wavelet transform, with principal components analysis for dimensionality reduction [67, 152]. The feature generating function used in this work is specified in Section 5.3.

Priors

To enable Bayesian clustering updates, clustering parameters are not stored as point estimates, but as parameterized probability distributions. The mean $\mathbf{M}_k$ and precision $\mathbf{\Lambda}_k$ parameters of each cluster are represented by a normal-Wishart distribution for each cluster and the mixing coefficients π_k are represented in a Dirichlet distribution:

$$\{\mathbf{M}_k, \mathbf{\Lambda}_k\} \sim \mathcal{N}\left(\mathbf{M}_k|\theta_k, (\beta_k \mathbf{\Lambda}_k)^{-1}\right) \mathcal{W}\left(\mathbf{\Lambda}_k|\mathbf{W}_k, \upsilon_k\right) \tag{5.3a}$$

$$\pi_{1\ldots K} \sim \mathcal{D}\left(\alpha_{1\ldots K}\right) \tag{5.3b}$$

θ_k, β_k, $\mathbf{W}_k$, v_k, and α_k are hyper-parameters which describe the distributions for $\mathbf{M}_k$, $\mathbf{\Lambda}_k$, and π_k. The distribution on mean and precision for each cluster is independent of the distributions on means and precisions of all other clusters.

5.2.2 Variational Updates

The model of VB clustering on GMM is shown in graphical form in Figure 5.1 on the next page. The model includes the latent variables ρ_t, which indicate which cluster is responsible for (produced) the data point s_t. Each ρ_t is a 1-of-K binary vector — a vector with K entries of which one entry is 1 and the rest are 0. The ρ_t have conditional distributions:

$$P(\rho_t|\pi_{1\ldots K}) = \prod_{k=1}^{K} \pi_k^{\rho_{t,k}} \tag{5.4}$$

The k subscripts in $\rho_{t,k}$ indicate the kth entry of the vector ρ_t. New intermediary variables, $r_{t,k}$, called *responsibilities*, are defined as the expectations of $\rho_{t,k}$:

$$r_{t,k} = \langle \rho_{t,k} \rangle \tag{5.5}$$

Inference on this model with the above prior is performed using variational Bayes as presented by Bishop [7]. The fixed-point iterative updates can be separated into three groups: updates on responsibilities, updates on clusters (means and precisions), and updates on mixing coefficients. The updates on responsibilities are analogous to the expectation step in EM, and updates on clusters and mixing coefficients are analogous to the maximization step in EM. Responsibility updates are performed after each cluster and mixing coefficient update, i.e. the updates proceed in this order: responsibilities, clusters, responsibilities, mixing coefficients, and repeat. See Section 3.5.3 on page 69 for an overview of the fixed-point procedure.

At the beginning of variational inference, θ_k, β_k, $\mathbf{W}_k$, v_k, and α_k are initialized to their prior values. After inference converges, the posterior values of θ_k, β_k, $\mathbf{W}_k$, v_k,

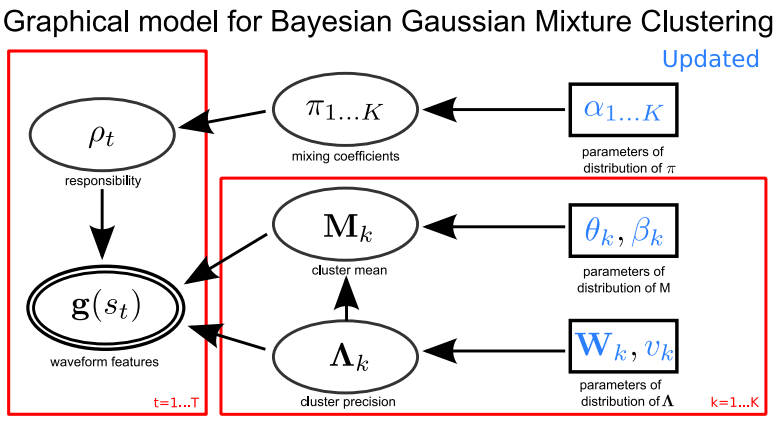

Figure 5.1: Graphical model for Bayesian Gaussian mixture clustering. Random variables are in ellipses. Observed random variables are in double ellipses. Observed point values are in rectangles. An arrow from A to B indicates B is parameterized by A. Red rectangles indicate plates of replicated variables.

and α_k are used for subsequent on-line spike-sorting. If the on-line spike-sorting uses Bayesian classification, then the posterior expectations of $\mathbf{M}_k$, $\mathbf{\Lambda}_k$, and π_k can be used. If the on-line spike-sorting uses template matching, the posterior responsibilities can be used to calculate the mean waveform to set as the new template for each cluster. Alternatively, The posterior responsibilities can be used to label waveforms for fitting the parameters of the on-line spike-sorting method.

Responsibility Update

For $k = 1 \ldots K$ and $t = 1 \ldots T$:

$$r_{k,t} \leftarrow \frac{1}{Z_t} \exp\left\{ \langle \ln \pi_k \rangle + \frac{1}{2} \langle \ln |\mathbf{\Lambda}_k| \rangle - \frac{d}{2} \ln(2\pi) - \frac{1}{2}\left(\frac{d}{\beta_k} + v_k (\mathbf{g}(s_t) - \theta_k)^T \mathbf{W}_k (\mathbf{g}(s_t) - \theta_k) \right) \right\} \tag{5.6}$$

Z_t is the normalizing constant required to make all $r_{k,t}$ for one value of t sum to one. d is the dimension of the features created by $\mathbf{g}()$. The π without subscripts is the

natural constant π. $\langle \ln \pi_k \rangle$ and $\langle \ln |\mathbf{\Lambda}_k| \rangle$ are:

$$\langle \ln \pi_k \rangle = \psi(\alpha_k) - \psi\left(\sum_{j=1}^{K} \alpha_j\right) \tag{5.7}$$

$$\langle \ln |\mathbf{\Lambda}_k| \rangle = d \ln 2 + \ln |\mathbf{W}_k| + \sum_{j=1}^{d} \psi\left(\frac{v_k + 1 - j}{2}\right) \tag{5.8}$$

$\psi()$ is the digamma function.

Cluster Update

For $k = 1 \ldots K$:

$$\beta_k \leftarrow \tilde{\beta}_k + n_k \tag{5.9}$$

$$\theta_k \leftarrow \frac{1}{\beta_k}\left(\tilde{\beta}_k \tilde{\theta}_k + n_k \bar{\mathbf{s}}_k\right) \tag{5.10}$$

$$v_k \leftarrow \tilde{v}_k + n_k \tag{5.11}$$

$$\mathbf{W}_k^{-1} \leftarrow \tilde{\mathbf{W}}_k^{-1} + n_k \mathbf{S}_k + \frac{\tilde{\beta}_k n_k}{\tilde{\beta}_k + n_k}\left(\bar{\mathbf{s}}_k - \tilde{\theta}_k\right)\left(\bar{\mathbf{s}}_k - \tilde{\theta}_k\right)^T \tag{5.12}$$

Tilde signs indicate prior values. n_k, $\bar{\mathbf{s}}_k$, and $\mathbf{S}_k$ are the statistics:

$$n_k = \sum_{t=1}^{T} r_{k,t} \tag{5.13}$$

$$\bar{\mathbf{s}}_k = \frac{1}{n_k} \sum_{t=1}^{T} r_{k,t} \mathbf{g}(s_t) \tag{5.14}$$

$$\mathbf{S}_k = \frac{1}{n_k} \sum_{t=1}^{T} r_{k,t} \left(\mathbf{g}(s_t) - \bar{\mathbf{s}}_k\right)\left(\mathbf{g}(s_t) - \bar{\mathbf{s}}_k\right)^T \tag{5.15}$$

Mixing Coefficient Update

For $k = 1 \ldots K$:

$$\alpha_k \leftarrow \tilde{\alpha}_k + n_k \tag{5.16}$$

5.2.3 Lower Bound Calculation

Hyper-parameter updates are repeated until convergence of the variational lower bound $\mathcal{L}$ (see Section 3.5.3 on page 69), which can be calculated using these equations:

$$\mathcal{L} = \mathcal{L}_s + \mathcal{L}_\rho + \mathcal{L}_\pi + \mathcal{L}_{\mathbf{M},\mathbf{\Lambda}} \tag{5.17}$$

$$\begin{aligned}\mathcal{L}_s =& \frac{1}{2}\sum_{k=1}^{K} n_k \Big\{ \langle \ln |\mathbf{\Lambda}_k| \rangle - \frac{d}{\beta_k} - v_k \operatorname{tr}\left(\mathbf{S}_k \mathbf{W}_k\right) \\ & - v_k \left(\bar{\mathbf{s}}_k - \theta_k\right)^T \mathbf{W}_k \left(\bar{\mathbf{s}}_k - \theta_k\right) - d \ln(2\pi) \Big\}\end{aligned} \tag{5.18}$$

$$\mathcal{L}_\rho = \sum_{t=1}^{T}\sum_{k=1}^{K} r_{k,t} \left(\langle \ln \pi_k \rangle - \ln r_{k,t} \right) \tag{5.19}$$

$$\begin{aligned}\mathcal{L}_\pi =& \ln \Gamma\left(\sum_{k=1}^{K} \tilde{\alpha}_k\right) - \sum_{k=1}^{K} \ln \Gamma(\tilde{\alpha}_k) - \ln \Gamma\left(\sum_{k=1}^{K} \alpha_k\right) + \sum_{k=1}^{K} \ln \Gamma(\alpha_k) \\ & + \sum_{k=1}^{K} (\tilde{\alpha}_k - \alpha_k) \langle \ln \pi_k \rangle\end{aligned} \tag{5.20}$$

$$\begin{aligned}\mathcal{L}_{\mathbf{M},\mathbf{\Lambda}} =& \frac{1}{2}\sum_{k=1}^{K} \Bigg\{ d\left(\ln \tilde{\beta}_k - \ln \beta_k\right) - d\left(\frac{\tilde{\beta}_k - \beta_k}{\beta_k}\right) - \tilde{\beta}_k v_k \left(\theta_k - \tilde{\theta}_k\right)^T \mathbf{W}_k \left(\theta_k - \tilde{\theta}_k\right) \\ & - \tilde{v}_k |\tilde{\mathbf{W}}_k| + v_k |\mathbf{W}_k| - (d \ln 2)(\tilde{v}_k - v_k) - 2\left(\sum_{i=1}^{d} \ln \Gamma\left(\frac{\tilde{v}_k + 1 - i}{2}\right)\right) \\ & + 2\left(\sum_{i=1}^{d} \ln \Gamma\left(\frac{v_k + 1 - i}{2}\right)\right) + (\tilde{v}_k - v_k) \langle \ln |\mathbf{\Lambda}_k| \rangle \\ & - v_k \left(\operatorname{tr}\left(\tilde{\mathbf{W}}_k^{-1} \mathbf{W}_k\right) - d \right) \Bigg\}\end{aligned} \tag{5.21}$$

tr() is the trace operator and $\Gamma()$ is the Gamma function.

5.2.4 Initial Clustering

The initial spike-sorting can be performed using VBGM clustering with uninformative or diffuse priors. Note that the priors for clusters should not be completely the same, or the algorithm may be stuck in a degenerate solution. Small random values for the prior means are used in this work.

For all experiments in Section 5.3, prior values during initial spike-sorting were, for each cluster $k = 1 \dots K$:

$$\tilde{\theta}_k \sim \mathcal{N}(0, 25 \cdot \mathbf{I}) \tag{5.22}$$

$$\tilde{\beta}_k = 10 \tag{5.23}$$

$$\tilde{\mathbf{W}}_k = 10 \cdot \mathbf{I} \tag{5.24}$$

$$\tilde{v}_k = d + 2 \tag{5.25}$$

$$\tilde{\alpha}_k = 1 \tag{5.26}$$

$\mathbf{I}$ is the d by d identity matrix.

5.2.5 Transition Model for Parameters

Since $\mathbf{M}_k$, $\mathbf{\Lambda}_k$, and π_k are assumed to change over time, transition models are needed for them. $\mathbf{M}_k$ used the identity transition model with normally-distributed noise. The transition for $\mathbf{\Lambda}_k$ and π_k used a multiplier to increase variance. The transitions were applied using the following equations for each cluster $k = 1 \dots K$:

$$\beta_k^{-1} \leftarrow \beta_k^{-1} + \delta \tag{5.27}$$

$$v_k \leftarrow \gamma v_k \tag{5.28}$$

$$\mathbf{W}_k \leftarrow \gamma \mathbf{W}_k \tag{5.29}$$

$$\alpha_k \leftarrow \zeta \alpha_k \tag{5.30}$$

$0 \leq \delta$ is the noise variance added to the variance for $\mathbf{M}_k$. $0 < \gamma \leq 1$ is the parameter for the variance increase for $\mathbf{\Lambda}_k$, and the multiplication to $\mathbf{W}_k$ is applied per element. $0 < \zeta \leq 1$ is the parameter for the variance increase for π_k.

5.3 Experiments

Variational Bayesian Gaussian mixture (VBGM) clustering updates were applied to synthetic data and neuronal data recorded from a monkey. The synthetic experiments allowed clustering update results to be compared to the true mixture components. The demonstrations on neuronal data show the method can operate on real-world data.

5.3.1 Synthetic Experiments

Two-dimensional synthetic data were generated from time-varying Gaussian mixtures to test the ability of VBGM clustering updates to adapt to changing clusters. Synthetic data were meant to represent waveform features; however, no simulated waveforms were generated for these experiments.

Experiment 1

The mixture model in experiment 1 had three components with static mixing coefficients $\pi_1 = 0.3$, $\pi_2 = 0.3$, and $\pi_3 = 0.4$. The static component covariances were:

$$\begin{aligned} \mathbf{\Lambda}_1^{-1} &= \mathbf{I} \\ \mathbf{\Lambda}_2^{-1} &= 2 \cdot \mathbf{I} \\ \mathbf{\Lambda}_3^{-1} &= 0.5 \cdot \mathbf{I} \end{aligned} \tag{5.31}$$

$\mathbf{\Lambda}_i^{-1}$ is the covariance matrix of the ith component, and $\mathbf{I}$ is the identity matrix. Component centers varied with time:

$$\begin{aligned}
\mathbf{M}_{1,t} &= \left(2 - \frac{2t}{10000}\ ,\ 4 - \frac{2t}{10000}\right) \\
\mathbf{M}_{2,t} &= \left(3 + \frac{2t}{10000}\ ,\ -1 + \frac{2t}{10000}\right) \\
\mathbf{M}_{3,t} &= \left(-0.5 + \frac{3t}{10000}\ ,\ 2 - \frac{2t}{10000}\right)
\end{aligned} \tag{5.32}$$

t is the iteration number. The component means change roughly 2 standard deviations in each axis over the course of 10,000 iterations. One thousand data points were drawn from the mixture model with $t = 0$ for initial clustering ("training data"). Ten thousand data points were drawn while t incremented from 1 to 10,000, with one data point per iteration, to create a time-varying test set. The test set is plotted in Figure 5.2 on the following page. Each of the first ten panels shows 1,000 data points with feature dimension 1 on the x-axis and feature dimension 2 on the y-axis. The last two panels show feature dimension versus time plots of all data points. Data points which belong to the first, second, and third components are colored red, green, and blue, respectively. Coloring is for visualization only; the true cluster labels were not provided to the VBGM clustering procedure. Magenta Xs indicate the true component means (at the middle of the iteration interval) in the first ten panels, and magenta lines indicate true component means in the last two panels.

Procedure

Initial clustering was performed on the static training data using the prior values described in the previous section and the true number of clusters provided to the algorithm. Then, VBGM clustering updates were executed sequentially on the test set in batches of 1,000 data points. For the transition model on the clustering parameters, $\delta = 10^{-2}$ and $\gamma = \zeta = 1$.

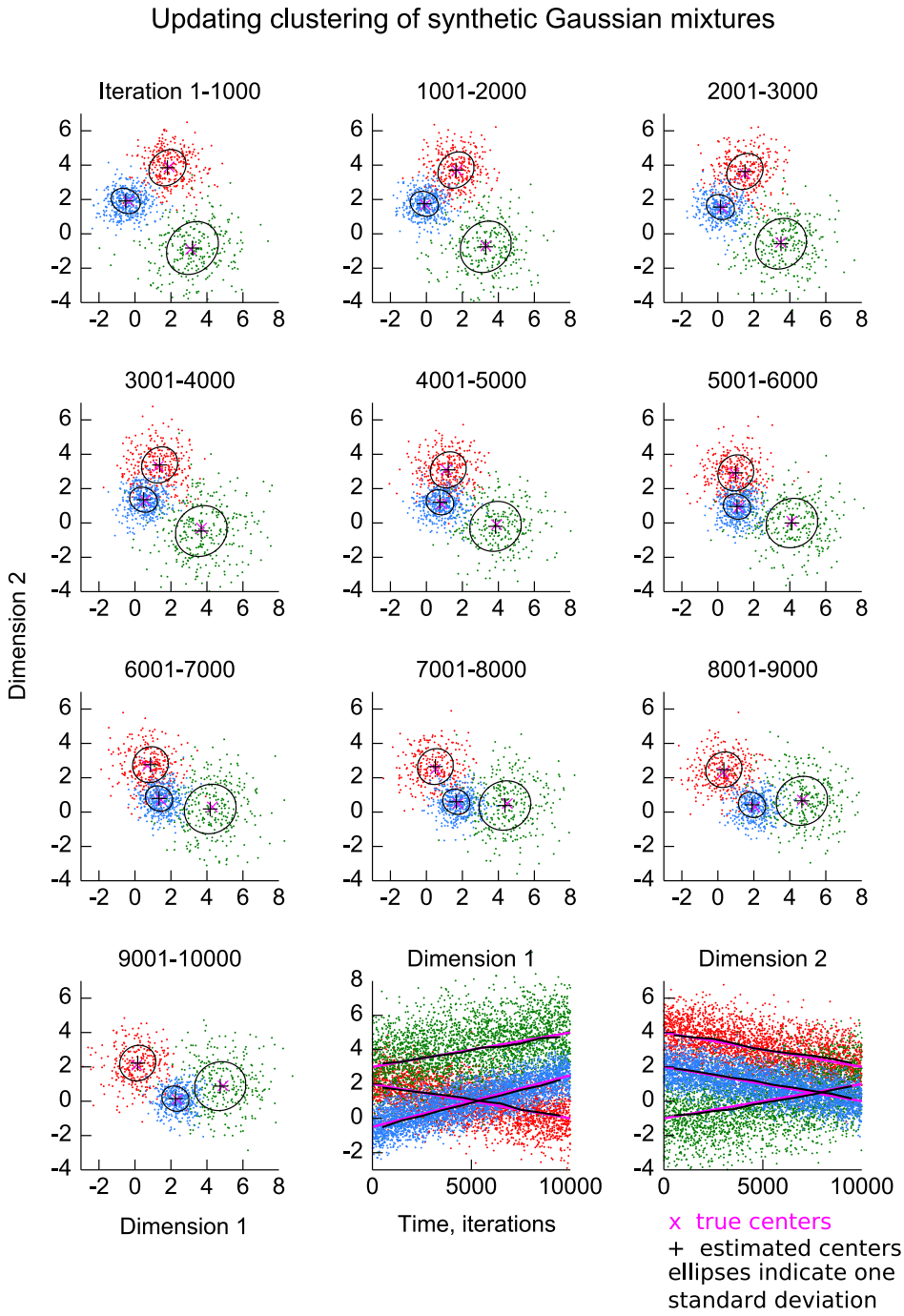

Figure 5.2: Clustering updates on synthetic data, experiment 1. Only component means changed. One data point was generated each iteration according to a time-varying mixture model. Lower right two graphs plot each feature dimension versus time.

Results

The clusters found by each update are plotted in Figure 5.2 on the previous page. Black plus signs and ellipses indicate inferred cluster means and standard deviations, respectively. In the lower right two plots, the black curves show cluster means over time. The VBGM clustering updates were able to accurately track the moving clusters.

Experiment 2

The mixture model for experiment 2 was similar to the one in experiment 1, except that the component centers were:

$$\begin{aligned}
\mathbf{M}_{1,t} &= \left(2 + \frac{3t}{10000}\ ,\ 4 - \frac{3t}{10000}\right) \\
\mathbf{M}_{2,t} &= \left(3 - \frac{2t}{10000}\ ,\ -1 + \frac{3t}{10000}\right) \\
\mathbf{M}_{3,t} &= \left(-0.5\ ,\ 2 - \frac{4t}{10000}\right)
\end{aligned} \tag{5.33}$$

The components in experiment 2 move faster and closer to each other. The component means change roughly 3 standard deviations in each axis over the course of 10,000 iterations. The experiment procedure and transition model for parameters in experiment 2 are similar to those in experiment 1.

Results

The clusters found by each update are plotted in Figure 5.3 on the following page. The VBGM clustering updates were able to track the moving clusters, but less accurately than in experiment 1. Particularly, tracking of the green cluster was adversely affected by the proximity of the red cluster.

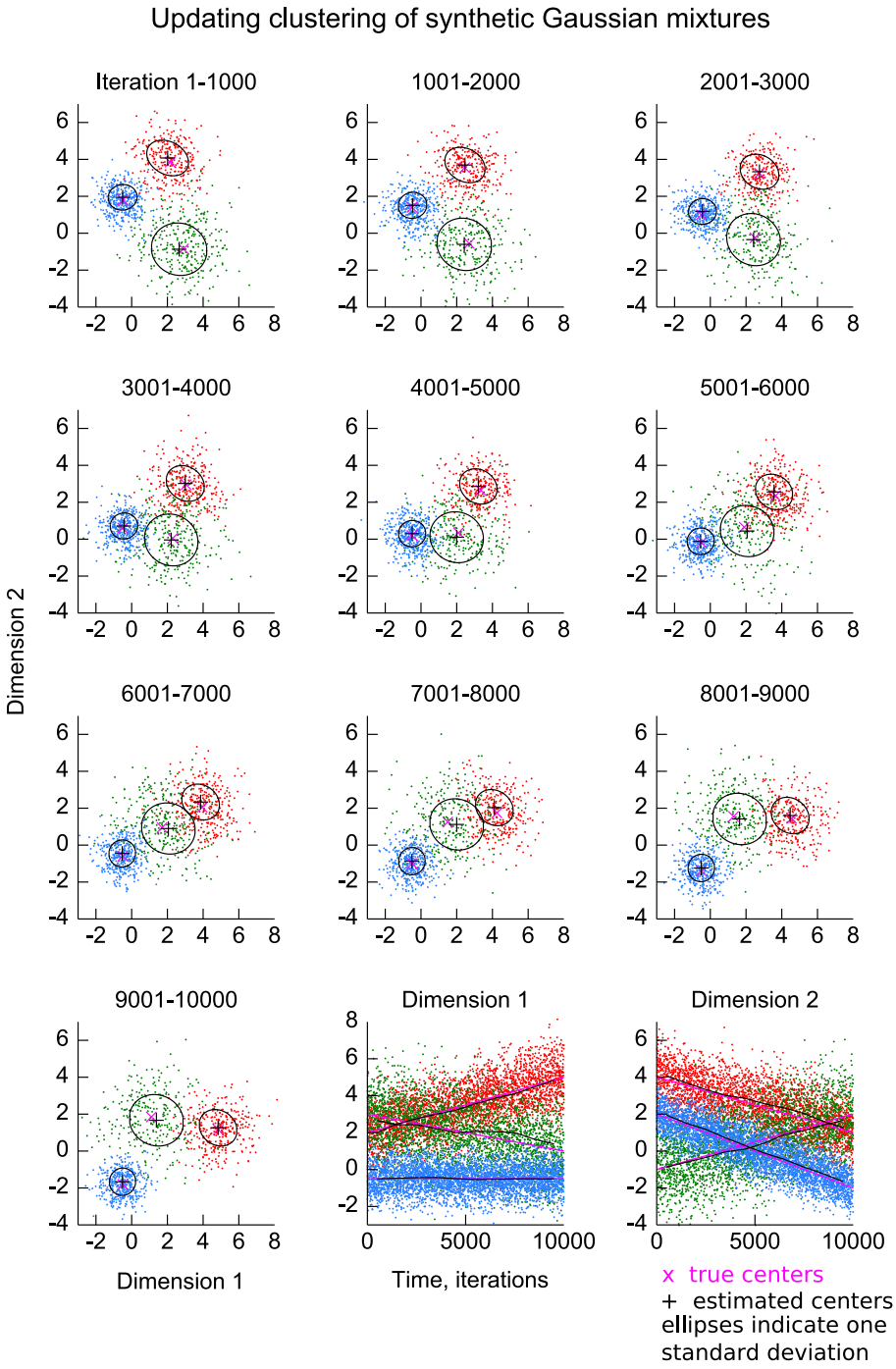

Figure 5.3: Clustering updates on synthetic data, experiment 2. Component means changed more quickly than in experiment 1. One data point was generated each iteration according to a time-varying mixture model. Lower right two graphs plot each feature dimension versus time.

Experiment 3

The mixture model in experiment 3 had time-varying component means, covariances, and mixing coefficients. Since tracking all parameters was more difficult, more data points (50,000) were generated and the changes were slower. The component means were:

$$\begin{aligned}
\mathbf{M}_{1,t} &= \left(2 + \frac{3t}{50000}\ ,\ 4 - \frac{3t}{50000}\right) \\
\mathbf{M}_{2,t} &= \left(3 - \frac{2t}{50000}\ ,\ -1 + \frac{3t}{50000}\right) \\
\mathbf{M}_{3,t} &= \left(-0.5\ ,\ 2 - \frac{4t}{50000}\right)
\end{aligned} \tag{5.34}$$

The component covariances were:

$$\begin{aligned}
\mathbf{\Lambda}_{1,t}^{-1} &= \begin{bmatrix} 1 & -\frac{0.7t}{50000} \\ -\frac{0.7t}{50000} & 1 \end{bmatrix} \\
\mathbf{\Lambda}_{2,t}^{-1} &= \begin{bmatrix} 2 + \frac{t}{50000} & 0 \\ 0 & 2 + \frac{t}{50000} \end{bmatrix} \\
\mathbf{\Lambda}_{3,t}^{-1} &= \begin{bmatrix} 0.5 & \frac{0.3t}{50000} \\ \frac{0.3t}{50000} & 0.5 \end{bmatrix}
\end{aligned} \tag{5.35}$$

The mixing coefficients were:

$$\begin{aligned}
\pi_{1,t} &= 0.3 + \frac{0.2t}{50000} \\
\pi_{2,t} &= 0.3 - \frac{0.1t}{50000} \\
\pi_{3,t} &= 0.4 - \frac{0.1t}{50000}
\end{aligned} \tag{5.36}$$

The component means change roughly 3 standard deviations. Fifty thousand data points were drawn while t incremented from 1 to 50,000. The test set is plotted in

Figure 5.4 on the next page. Each of the first 10 panels shows 5,000 data points. Magenta Xs and ellipses indicate the true component centers and standard deviations, respectively. VBGM clustering updates were applied sequentially to the test set in batches of 5,000 data points. For the transition model on the clustering parameters, $\delta = 10^{-2}$, $\gamma = 0.99$, and $\zeta = 0.5$.

Results

The clusters found by the updates are plotted in Figure 5.4 on the following page. The VBGM clustering updates were able to track the changing cluster means and, to a lesser extent, covariances. Again, tracking of the green cluster was poor compared to the other clusters. The final covariances found by the updates were less elongated than the true covariances. The initial and final coefficients found by VBGM clustering were: (initial:) $\pi_1 = 0.2906$, $\pi_2 = 0.3021$, $\pi_3 = 0.4073$, (final:) $\pi_1 = 0.4640$, $\pi_2 = 0.2132$, and $\pi_3 = 0.3228$.

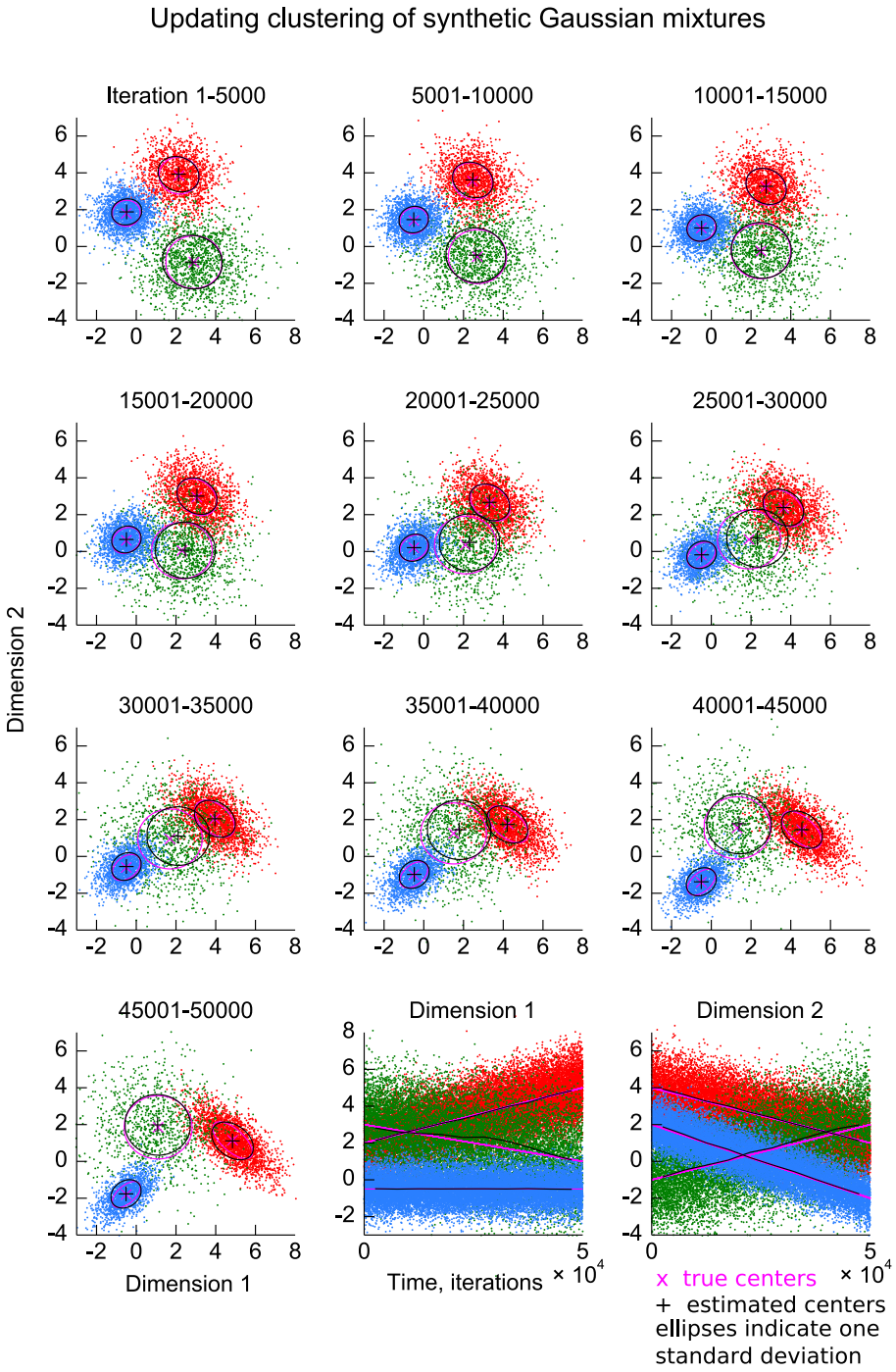

Figure 5.4: Clustering updates on synthetic data, experiment 3. Component means, covariances, and mixing coefficients changed. One data point was generated each iteration according to a time-varying mixture model. Lower right two graphs plot each feature dimension versus time.

5.3.2 Tracking Neuronal Waveforms

Neuronal waveform data from monkey C (see Section 4.1 on page 78) were used to demonstrate spike-sorting updates. In each demonstration, spikes at the beginning and end of sessions were discarded to remove transient noise. Noise waveforms among the remaining spikes were culled by template matching against a database of common noise shapes. Waveforms which saturated the analog to digital converter for more than 10% of samples were discarded. Remaining spikes were separated into 6 blocks with an equal number of spikes in each.

Waveform Features

Features were generated from spike waveforms by first computing the time-derivative of each waveform. Then principal components analysis was performed on the results from the first block, and the largest two principal components were used to reduce dimensionality to two on all blocks. Two was chosen as the feature dimensionality for easy visualization. The features were scaled to have standard deviations of 2, to roughly match the scale of the synthetic data.

Procedure

Initial clustering was performed on the first block using the prior values described in the previous section. The number of clusters observed by the author was provided to the algorithm. Then, VBGM clustering updates were applied sequentially to the remaining 5 blocks. For the transition model on the clustering parameters, $\delta = 10^{-2}$ and $\gamma = \zeta = 1$.

Results

Clustering updates from 3 example electrodes are shown in Figures 5.5, 5.6, and 5.7. Spikes in Figures 5.5, 5.6, and 5.7 were recorded over the span of 23, 24, and

35 minutes, respectively. Black plus signs and ellipses indicate means and standard deviations of inferred clusters. In the sixth panel of each figure, initial clusters are shown in red for comparison. The bottom plots show mean initial and final waveforms from each cluster. The means were weighed by posterior responsibilities. Visual inspection suggests that VBGM clustering updates are able to track the slowly changing clusters. Figure 5.7 shows a substantial change in waveform shape over the 35 minutes of spikes shown.

Computational Complexity and Speed

With a moderately-optimized MATLAB implementation of VBGM clustering, execution speed was very fast. Initial clustering of 15,000 data points took less than 2 seconds on an Intel Core i7 class desktop computer. Clustering updates were even faster, converging in a few iterations and taking less than 0.25 seconds when operating on batches of 15,000 data points. The fast updates are due to the highly informative prior and initialization.

Empirically, the responsibility calculation is the slowest part of the algorithm. The slowest steps in each iteration in terms of computational complexity are in Equations 5.6 on page 152 and 5.15 on page 153, with complexity $O(TKd^2)$, where T is the amount of data, K is the number of clusters, and d is the feature dimensionality. Overall, one iteration of VBGM clustering has complexity $O(TKd^2 + Kd^3)$, where the cubic term comes from the inverse in Equation 5.12 on page 153.

Spike-Sorting Update After Each Spike

The linear dependence on T of the complexity and the observation that the amortized execution time per data point was small suggest that real-time updates after receiving each spike is possible with proper optimizations. The update procedure should converge quickly when fed single data points, because of the relative strength of the

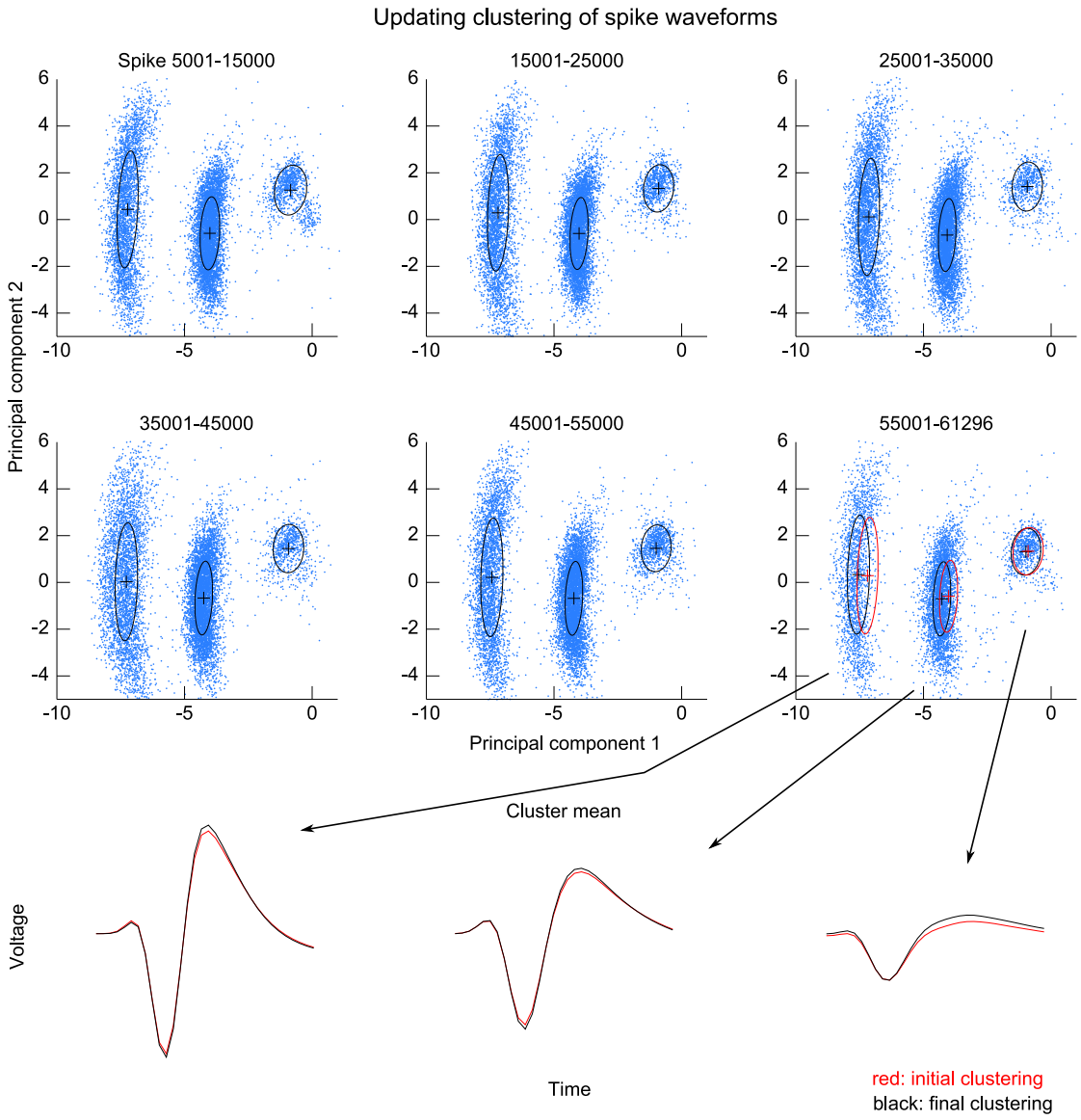

Figure 5.5: Clustering updates on neuronal data, example 1. Black plus signs and ellipses indicate means and standard deviations of clusters. Red symbols in last clustering panel show initial clusters for comparison. Waveform plots at bottom show comparison of initial and final mean waveforms per cluster. Data shown spans 23 minutes.

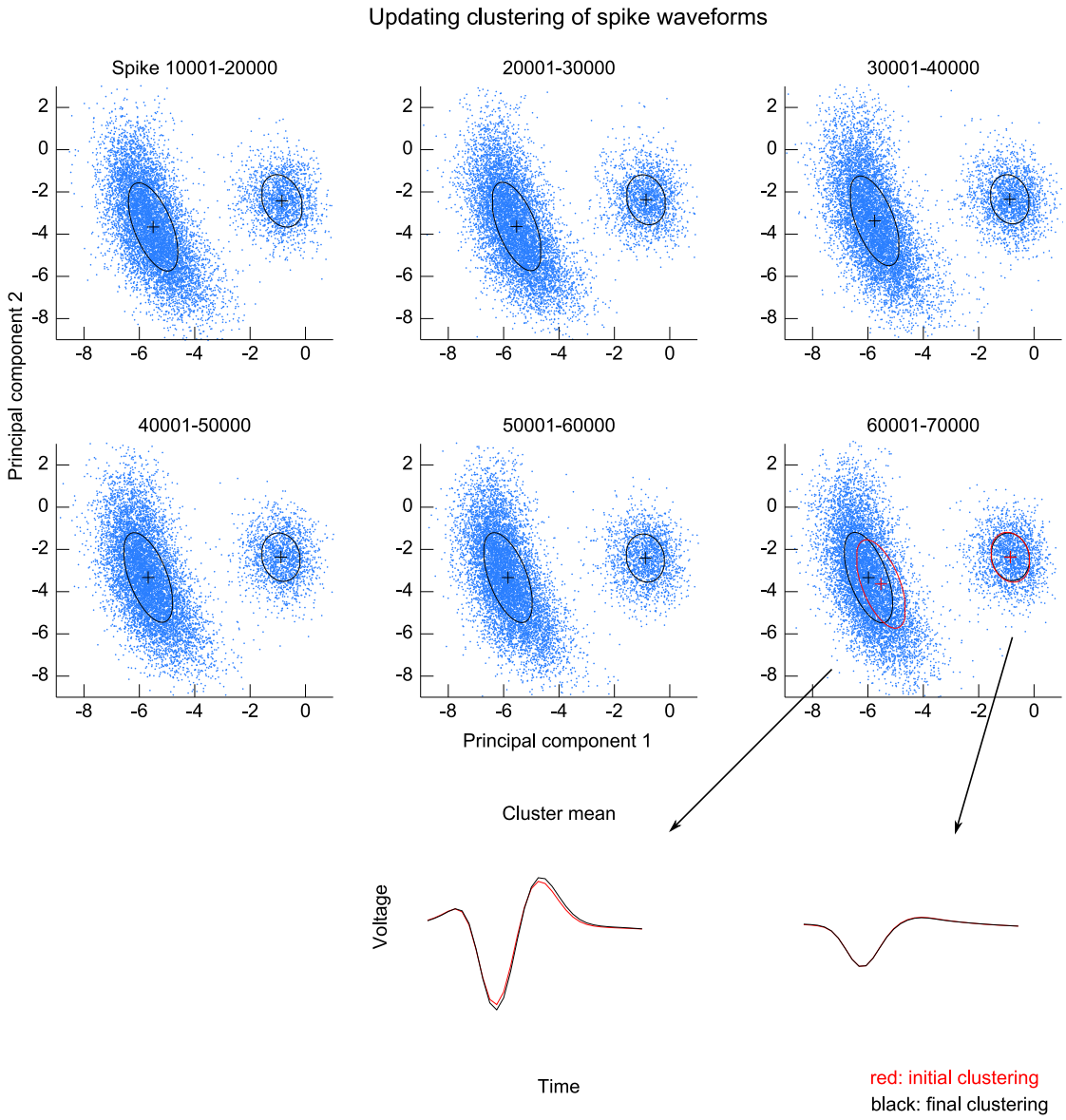

Figure 5.6: Clustering updates on neuronal data, example 2. Black plus signs and ellipses indicate means and standard deviations of clusters. Red symbols in last clustering panel show initial clusters for comparison. Waveform plots at bottom show comparison of initial and final mean waveforms per cluster. Data shown spans 24 minutes.

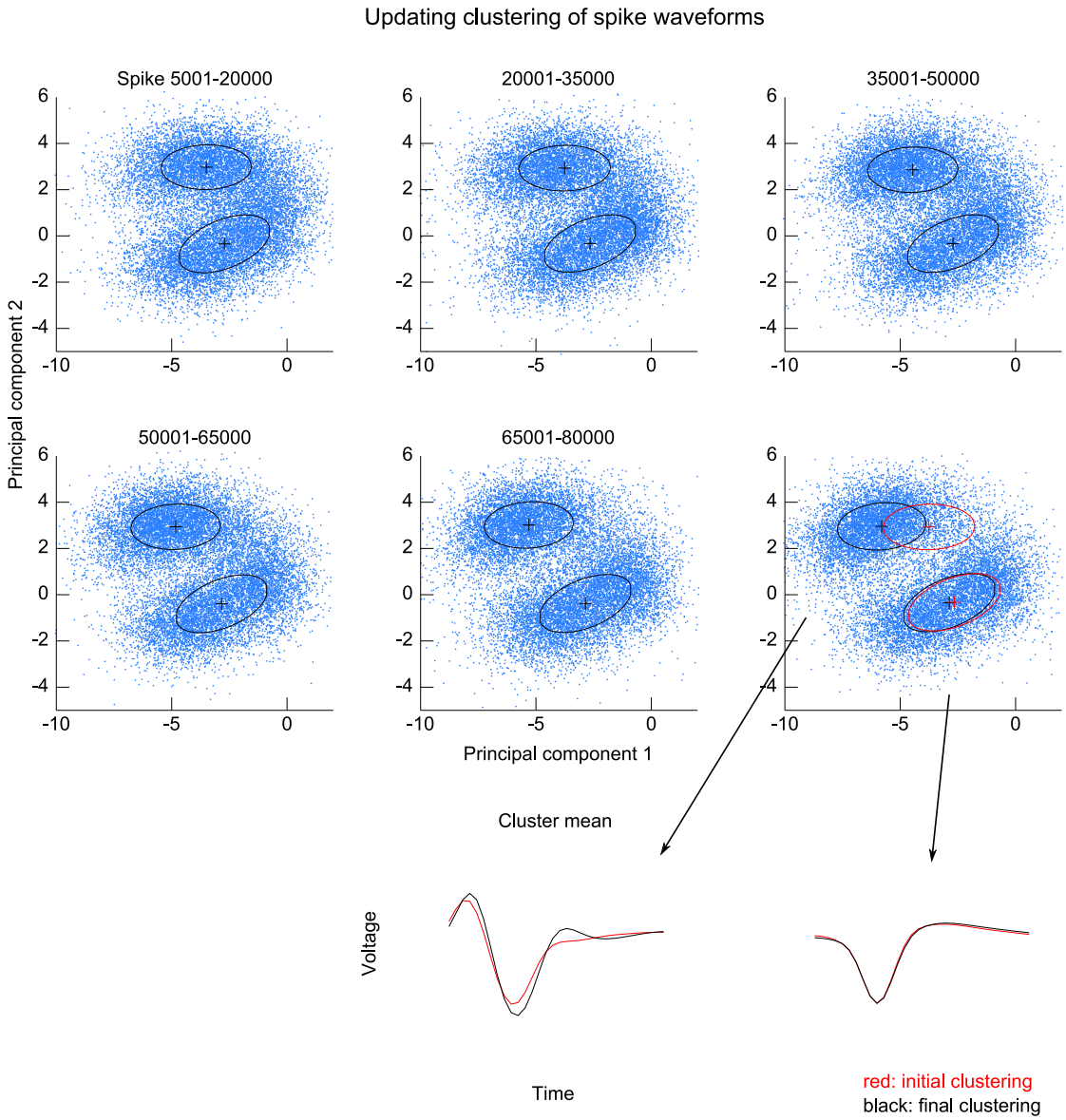

Figure 5.7: Clustering updates on neuronal data, example 3. Black plus signs and ellipses indicate means and standard deviations of clusters. Red symbols in last clustering panel show initial clusters for comparison. Waveform plots at bottom show comparison of initial and final mean waveforms per cluster. Data shown spans 35 minutes.

prior compared to the data. Even if execution time runs out before convergence, the result will only be biased towards the prior.

Spike-sorting updates on individual spikes were tested to evaluate computational speed. The average execution rate was approximately 220 Hz over 10,000 spikes using 2 clusters and 2 feature dimensions. The rate should be substantially higher for an efficient implementation in C/C++. Real-time VBGM spike-sorting updates on single or small batches of spikes should be possible with an efficient implementation.

Chapter 6

Conclusion

This chapter discusses results and summarizes this dissertation. Possible extensions and avenues for future work are also discussed.

6.1 Discussion

6.1.1 n-th Order Unscented Kalman Filter

The off-line and on-line experiments in Chapter 4 demonstrate the improved accuracy of the n-th order UKF versus commonly-used linear decoders. This improved accuracy is due to two main factors.

Quadratic Model of Tuning

The quadratic model of tuning predicts binned spike counts better than previous linear models (mean 0.02 dB improvement per neuron). This more accurate model translates to improved reconstruction accuracy of the 1st order UKF versus the Kalman filter with a linear model (mean 0.39 dB improvement) and the Wiener filter (mean 0.25 dB improvement). The use of the unscented Kalman filter allows inference under a non-linear model without the need for computationally expensive Monte Carlo simulation.

Taps in State Space

The n taps in the state of the n-th order UKF allows the tuning model to capture relationships between neuronal activity and desired movements at multiple time offsets simultaneously. In previous work with the Kalman filter, the best time offset for each neuron had to be found by optimization [149]. By including multiple time offsets, the best subset can be found via regularization instead of combinatorial search, and more than one offset may be used. Additionally, the n taps in the state mean that the movement model is an order n autoregressive process. This allows specification of richer models of movement which are more able to capture stereotypical patterns of movements.

The benefit of using multiple taps in the state is demonstrated in the better accuracy of the $n = 10$ tap UKF versus the $n = 1$ tap UKF (mean 0.85 dB improvement). The benefit from modeling tuning relationships across multiple time offsets is shown in the comparison of two constant-velocity movement model UKFs, which do not benefit from the fitted autoregressive order n movement model. The $n = 10$ constant-velocity movement model UKF was 2.3 dB more accurate than the $n = 1$ constant-velocity movement model UKF. By comparing the 10th order UKF with a 10th order constant-velocity movement model UKF, the contribution of the fitted autoregressive order 10 movement model was found to be $6.25 - 3.88 = 2.37$ dB. Even though the movement model contributed much to the accuracy in reconstructing monkey behavior on the pursuit task, it was unable to make reconstructions without the neuronal recordings, as shown in Figure 4.7.

Setting the Number of Taps

The off-line reconstructions suggested the optimal number of taps n depends on the behavioral task. For the pursuit task with Lissajous curve trajectories, reconstruction accuracy increased with the number of future taps and increased slightly with the

number of past taps. For the center-out task, a small number of future taps resulted in the best accuracy, while the accuracy was largely insensitive to the number of past taps. Also, the improvement of the 10th order UKF versus the 1st order UKF was larger for the pursuit task than for the center-out task (1.23 dB versus 0.48 dB).

These findings suggest that the movements in the pursuit task were autocorrelated over longer time spans. The center-out task movements were not limited in speed, and thus the monkey performed them quickly to increase juice reward per unit time. Center-out reaches were relatively short, on the order of 1 second. Also, center-out reaches were uncorrelated trial-to-trial, which limited the usefulness of a high order autoregressive movement model. In the pursuit task, movement speeds were dictated by the speed of the target, which forced the monkey to move slower. Also, trajectories followed a predictable pattern with a period on the order of half a minute. Thus, the large n autoregressive model was beneficial in modeling the movement.

Accuracy of UKF Approximations

The comparison of the n-th order UKF to the particle filter with the same model shows that the unscented transform approximation is quite accurate. For the $n = 1$ setting, the particle filter, with a fairly numerous 50,000 particles, was significantly more accurate, but the difference was not large (0.07 dB). However, at the $n = 10$ setting, with a large number of dimensions (40) in the state space, the particle filter was no more accurate than the UKF. Using the unscented transform thus sacrifices some accuracy to convert the exponential blow-up of the curse of dimensionality into a 3rd order polynomial increase.

Importance of Covariance Modeling

Off-line reconstructions with randomly chosen subsets of neurons of different cardinalities (Figure 4.8) show the importance of modeling the noise covariance among

neurons. The population vector decoding method ignores this covariance, which leads to decreased accuracy as the number of neurons increases. The slow rise in performance of the Kalman filter with off-diagonal covariance matrix entries set to zero also demonstrates this importance. These findings highlight one disadvantage of early point-process methods [4, 9, 10, 24, 25, 112, 124, 125, 137, 138], which do not model neuron noise covariance, because they use Poisson noise models. However, the point-process framework can be extended to model non-tuning covariance among neurons as part of the state [126].

6.1.2 Bayesian Regression Self-Training

Experiments on synthetic data and neuronal data with noise added or neurons switched show that Bayesian regression self-training can be used to track time-varying observation models. Off-line reconstructions demonstrate the ability of Bayesian regression self-training to improve the reconstruction accuracy of the n-th order UKF. Improvements were found in sessions as short as 10 minutes. Neural tuning may not have changed substantially in that short time span; rather, the self-training update method may have improved upon the initial parameter fit to produce better reconstructions. If this was the case, the improved accuracy demonstrates the ability of the self-training updates to refine the tuning model autonomously. This autonomous learning ability would allow the method to take advantage of newly discovered neurons or neurons which become more tuned through practice [31].

Comparison to Wu and Hatsopoulos

Wu and Hatsopoulos [150] reported improvements of 0.0022 cm^2 and 0.0053 cm^2 per trial reduction in mean squared-error with the Kalman filter. Each trial lasted 4 to 5 seconds. Using their reported static-decoder accuracy of 7.9 and 8.2 cm^2 MSE as baselines and assuming each trial lasted 4.5 seconds, their improvement

per minute was 0.016 dB/min and 0.038 dB/min. Bayesian regression self-training was able to achieve similar improvement rates of 0.036 dB/min (joint model) and 0.033 dB/min (factorized model) without using hand movement data for training, as Wu and Hatsopoulos did [150]. When hand movements were used for updates, the improvements from using Bayesian regression updates were even larger (0.078 dB/min for joint and 0.82 dB/min for factorized). However, Wu and Hatsopoulos used 7–9 minutes of training data, while the experiments in this work used 2 minutes of training data. The smaller amount of data used in the initial fit may reduce the accuracy of the non-updated UKF and increase the benefit from autonomous learning.

Long-term Adaptation

On-line, closed-loop experiments with brain-control over a period of nearly a month showed that Bayesian regression self-training allowed the 5th order UKF to maintain control accuracy much better than the non-updated 5th order UKF, which suffered substantial degradation in control accuracy. Even if the baseline firing rate parameters of the static 5th order UKF were updated, control accuracy was still much worse than the updated 5th order UKF. This demonstrates that updating baseline firing rate parameters alone is insufficient for maintaining control accuracy.

Comparison to Ganguly and Carmena

In the work of Ganguly and Carmena [31], performance as measured by percentage correct and time-to-target in a center-out task started low, increased quickly, and plateaued at about 10 days. After 10 days, performance stayed relatively constant for the remainder of the 19 day experiment interval. However, the performance metrics may have been saturated or the monkeys may not have been motivated to perform any better. In the experiments reported here, no effort was made to pick putatively stable neurons, which was done by Ganguly and Carmena. Control

accuracy was initially high but decreased slightly over the 29 day duration while using the self-training updates. Accuracy using the non-updated decoder was much worse, markedly different from the results reported by Ganguly and Carmena. This difference may be due to the use of all sorted units in this study, instead of only the putatively stable units. The loss of units or increase of noise can have a large impact on the accuracy of a static decoder.

Comparison to Similar Adaptive Filtering Methods

The Bayesian regression self-training method is similar to dual filtering with a low frequency parameter filter. One advantage of this approach is that the batch-mode operation allows Kalman smoothing to be used. This allows information from observations to propagate backwards in time through the state transition model.

The batch-mode update of Bayesian regression self-training is similar to the batch-mode system identification approaches proposed by Ghahramani and Roweis [38] and Beal [6]. However, Bayesian regression self-training does not repeat filtering and smoothing after the parameter updates, i.e. it only performs one "iteration" of EM for system identification [111]. Performing more than one iteration may improve accuracy. This extension would require filtering and smoothing to account for the uncertainty in the model parameters and Bayesian regression to account for the uncertainty in the smoothing output.

However, performing more than one iteration may slow down the algorithm for little gain. Unlike the batch-mode system identification approaches mentioned above, Bayesian regression self-training is intended to update model parameters, not discover them for the first time. The changes it finds are expected to be small, and the prior information should weigh heavily compared to the information from new data. The fast convergence of VBGM clustering updates suggests that an iterative Bayesian regression self-training update would likely converge in a few iterations as well.

6.1.3 VBGM Clustering Update

Demonstrations on synthetic and neuronal data show the ability of variational Bayesian Gaussian mixture clustering updates to track slowly changing Gaussian mixtures and spike waveform clusters. The updates are computationally light; real-time spike-sorting updates on single or small batches of spikes should be possible with an efficient implementation.

Comparison to Wolf and Burdick

Wolf and Burdick [142] also used priors for initialization of clustering in a Bayesian extension of expectation-maximization. However, they only placed strong priors on the cluster centers, leaving the cluster covariances and mixing coefficients to use diffuse priors. Wolf and Burdick used a non-probabilistic transition model for the cluster parameters which required searching for cluster correspondences using a similarity metric. In comparison, the VBGM clustering update uses a fully Bayesian approach with priors on all clustering parameters. Having priors on all parameters speeds up convergence and enforces a transition model on all parameters. Wolf and Burdick include provision for handling loss and discovery of neurons, as well as noise outliers. These features are avenues for future work with respect to VBGM clustering updates.

Comparison to Gasthaus et al.

The method proposed by Gasthaus et al. [34] includes several useful features and provides an elegant approach to the neuron loss and discovery problem. However, their probabilistic model requires Monte Carlo estimation, which comes with a large computational burden, especially as the number of neurons recorded increases into the hundreds or thousands. In comparison, the VBGM clustering update is simpler and lacks the ability to handle neuron loss and discovery. However, the VBGM clustering

update is computationally far lighter.

6.2 Extensions and Future Work

Parameter Optimization

The parameters choices used in the n-th order UKF and Bayesian regression self-training can be further optimized. The dependency of decoding accuracy on the κ parameter of the UKF can be explored. The effects of various settings of the degrees of freedom cap parameter can be explored. Also, the merit of the multiplier method for adjusting the certainty of the inverse-Wishart distribution for the observation noise covariance can be explored in longer duration experiments. More sophisticated hierarchical Bayesian inference may be able to estimate the best model drift and degrees of freedom cap or multiplier directly from the data, similar to the work of Sykacek et al. [117, 118].

Adaptation of Movement Models

Adaptive filtering can be applied to movement models as well. There is a rich literature on transition model selection and learning (for a survey, see [69–71]). Learning and updating a library of stereotypical movements would enable a prosthetic to have "muscle memory", similar to how humans master repetitive movements. This would allow off-loading of a portion of control to the prosthetic system.

Error-In-Variables Regression

The linear regression in self-training is more accurately described with an error-in-variables regression formulation, because the smoothed filter outputs have uncertainty. In its present form, Bayesian regression ignores this uncertainty, similar to the dual filtering approach, which also considers the state filter outputs as point estimates. Using a Bayesian error-in-variables regression formulation would account for the

uncertainty in the state estimates similar to joint filtering. Furthermore, once parameters are updated, filtering and smoothing may be repeated with the new parameters to refine the predicted movements. Then the model parameter updates can be improved further and the process repeated, as discussed in the previous section. A variational Bayesian implementation of this idea for linear state-space models was given by Beal [6]. An extension for non-linear state-space models may be possible using the unscented transform. However, this process may be too computationally intensive for the BMI application, and the refined trajectory estimates are not useful for real-time control.

Neuron Loss and Discovery

The VBGM clustering update can be extended to handle neuron loss and discovery using automatic determination of the number of clusters [7]. Determining neuron loss with this method is straightforward. One approach for determining neuron discovery is to always posit one or more new clusters at each update, assign diffuse priors for these new clusters, and keep them if they are determined to be real clusters after inference. Alternatively, neuron loss and discovery can be handled using a Dirichlet process mixture model similar to that used by Gasthaus et al. [34] and using variational approximations [153] for fast inference.

Joint Update of Spike-Sorting and Tuning Model

Ventura has shown that spike-sorting using tuning information improves accuracy [132, 133]. The tuning model parameter identification process should also be aware of the uncertainty in spike-sorting to properly estimate uncertainties in tuning parameters. Thus, updates of spike-sorting parameters and tuning model parameters should occur together in a joint inference.

6.3 Summary

This dissertation presented a non-linear adaptive Bayesian filter for BMI decoding and a method for adaptive spike-sorting, enabling a fully-adaptive brain-machine interface. The methods are the n-th order unscented Kalman filter with Bayesian regression self-training updates and the variational Bayesian Gaussian mixture clustering update.

n-th Order UKF

The n-th order UKF allows the use of non-linear models of neuronal tuning with Kalman filtering by using the unscented transform. A quadratic model of tuning was developed to take advantage of this ability. The n-th order UKF also incorporates multiple time taps of kinematic variables in its state. This allows tuning models to capture relationships between neuronal activity and movement at multiple time offsets simultaneously, and makes the movement model of the filter an autoregressive order n process.

Bayesian Regression Self-Training

The n-th order UKF was made adaptive by updating the tuning model parameters using Bayesian regression self-training. In this method, filter outputs and neuronal recordings are cached and used to perform updates periodically. Filter outputs are first smoothed with a Kalman smoother to increase their accuracy. Then, Bayesian regression is applied to update the tuning model. The Bayesian regression uses the previous tuning model parameters as priors and computes the updated parameters using the smoothed filter output and neuronal recordings. Two Bayesian regression variants were presented. The joint distribution variant can be computed analytically. The factorized distribution variant is more flexible but requires iterative approximation using variational Bayes.

VBGM Clustering Update

The VBGM clustering update was used to update the clustering of spike waveforms, which enables adaptive spike-sorting. The VBGM clustering update is a Bayesian extension of the expectation-maximization clustering algorithm for Gaussian mixture models. Previous clustering parameters are used as priors for the Bayesian clustering update, facilitating tracking of clusters over time and fast convergence.

Experiments

The proposed methods were tested in off-line and on-line experiments with Rhesus monkeys implanted in the cortex with micro-wire electrode arrays. Off-line analysis on 16 sessions from 2 monkeys demonstrates the increased predictive ability of the quadratic model of tuning versus the commonly-used linear model. Off-line reconstructions using the 1st and 10th order UKF demonstrate their improved accuracy over commonly-used linear decoders: the Wiener filter, the Kalman filter, and the population vector method. On-line, closed-loop experiments with 2 monkeys demonstrate that brain-control is more accurate when using the 10th order UKF than when using the Wiener filter and Kalman filter.

Adaptive filtering demonstrations on synthetic data suggest the Bayesian regression self-training method can be applied to tracking problems where the observation model is time-varying. Adaptive filtering on neuronal data with noise added or neurons switched shows that the method can adapt to changing tuning models.

Off-line reconstructions on 18 sessions from 3 monkeys with the 5th order UKF updated by Bayesian regression self-training were more accurate than reconstructions with the 5th order UKF alone. When actual hand movements were used by the updates, the accuracy improvement was even larger. In on-line, closed-loop experiments conducted over 29 days with one monkey, the 5th order UKF with Bayesian regression self-training was able to maintain control accuracy much better than the non-updated

5th order UKF.

Clustering update demonstrations on synthetic mixtures and neuronal spike waveforms show that the VBGM clustering updates can track slow changes in clusters and detect changes in waveforms.

These methods together comprise a non-linear, adaptive BMI which has improved control accuracy and robustness to tuning and recording change. These characteristics make BMI-controlled prosthetics more practical and bring the technology closer to clinical use.

Appendix A

Variational Updates for Bayesian Regression

This appendix derives the variational updates for factorized distribution variational Bayesian regression.

H Update

The update for the distributions of the tuning coefficients $\mathbf{H}_{(i)}$ from Equation 3.17 on page 70 is copied below:

$$Q(\mathbf{H}_{(i)}) = \frac{1}{Z_i} \exp\langle \ln P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y}) \rangle_{\sim Q(\mathbf{H}_{(i)})} \tag{A.1}$$

The index i specifies which neuron the tuning coefficients describe. The following operations simplify this equation into terms involving only $\mathbf{H}_{(i)}$:

$$\begin{aligned}
\ln Q(\mathbf{H}_{(i)}) &= \langle \ln P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y}) \rangle_{\sim Q(\mathbf{H}_{(i)})} + \text{constant} \\
&= \left\langle \sum_{t=1}^{T} \ln P\left(\mathbf{y}_t | \mathbf{H}, \mathbf{R}\right) + \ln P\left(\mathbf{H}_{(i)} | \tilde{\mu}_i, \tilde{\mathbf{\Lambda}}_i\right) \right\rangle_{\sim Q(\mathbf{H}_{(i)})} + \text{constant} \\
&= \left\langle \sum_{t=1}^{T} \ln \mathcal{N}\left(y_{i,t} | \mathbf{H}_{(i)} \mathbf{x}_t, \mathbf{R}\right) + \ln \mathcal{N}\left(\mathbf{H}_{(i)} | \tilde{\mu}_i, \tilde{\mathbf{\Lambda}}_i^{-1}\right) \right\rangle_{\sim Q(\mathbf{H}_{(i)})} + \text{constant}
\end{aligned} \tag{A.2}$$

$\mathbf{H}_{(i)}$ is a 1 by D row vector and μ_i is a D by 1 column vector. $y_{i,t}$ is the binned spike count of neuron i at time t. The log joint probability is factorized into a sum of log conditional probabilities of variables given their priors, using the graphical model of factorized Bayesian regression (Figure 3.3 on page 68). The terms which do not depend on $\mathbf{H}_{(i)}$ are constant under the expectation and are put into the constant term.

Next, the likelihood term is rewritten:

$$\begin{aligned}
&\sum_{t=1}^{T} \ln \mathcal{N}\left(y_{i,t} | \mathbf{H}_{(i)} \mathbf{x}_t, \mathbf{R}\right) \\
&\quad = -\frac{T}{2}\ln(2\pi) - \frac{T}{2}\ln|\mathbf{R}_{(i,i)}| - \frac{1}{2}\operatorname{tr}\left((\mathbf{Y}_{(i)} - \mathbf{H}_{(i)}\mathbf{X})^T \mathbf{R}_{(i,i)}^{-1}(\mathbf{Y}_{(i)} - \mathbf{H}_{(i)}\mathbf{X})\right) \\
&\quad = -\frac{T}{2}\ln(2\pi) - \frac{T}{2}\ln|\mathbf{R}_{(i,i)}| - \frac{1}{2}\operatorname{tr}\left(\mathbf{Y}_{(i)}^T \mathbf{R}_{(i,i)}^{-1}\mathbf{Y}_{(i)}\right) + \operatorname{tr}\left(\mathbf{Y}_{(i)}^T \mathbf{R}_{(i,i)}^{-1}\mathbf{H}_{(i)}\mathbf{X}\right) \\
&\qquad - \frac{1}{2}\operatorname{tr}\left((\mathbf{H}_{(i)}\mathbf{X})^T \mathbf{R}_{(i,i)}^{-1}(\mathbf{H}_{(i)}\mathbf{X})\right) \\
&\quad = \begin{bmatrix} \mathbf{R}_{(i,i)}^{-1}\mathbf{X}\mathbf{Y}_{(i)}^T \\ -\frac{1}{2}\mathbf{R}_{(i,i)}^{-1}\operatorname{vec}(\mathbf{X}\mathbf{X}^T) \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{(i)}^T \\ \operatorname{vec}(\mathbf{H}_{(i)}^T\mathbf{H}_{(i)}) \end{bmatrix} - \frac{T}{2}\ln(2\pi) - \frac{T}{2}\ln|\mathbf{R}_{(i,i)}| \\
&\qquad - \frac{1}{2}\operatorname{tr}\left(\mathbf{Y}_{(i)}^T \mathbf{R}_{(i,i)}^{-1}\mathbf{Y}_{(i)}\right)
\end{aligned} \tag{A.3}$$

$\mathbf{Y}_{(i)}$ is the ith row of $\mathbf{Y}$ corresponding to the binned spike counts of neuron i in the data.

Then the prior term is rewritten:

$$\begin{aligned}
&\ln \mathcal{N}\left(\mathbf{H}_{(i)}|\tilde{\mu}_i, \tilde{\mathbf{\Lambda}}_i^{-1}\right) \\
&\quad = -\frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\tilde{\mathbf{\Lambda}}_i^{-1}| - \frac{1}{2}\left((\mathbf{H}_{(i)}^T - \tilde{\mu}_i)^T \tilde{\mathbf{\Lambda}}_i (\mathbf{H}_{(i)}^T - \tilde{\mu}_i)\right) \\
&\quad = -\frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\tilde{\mathbf{\Lambda}}_i^{-1}| - \frac{1}{2}\left(\mathbf{H}_{(i)}\tilde{\mathbf{\Lambda}}_i\mathbf{H}_{(i)}^T\right) + \mathbf{H}_{(i)}\tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i - \frac{1}{2}\left(\tilde{\mu}_i^T\tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i\right) \\
&\quad = \begin{bmatrix} \tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i \\ -\frac{1}{2}\operatorname{vec}(\tilde{\mathbf{\Lambda}}_i) \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{(i)}^T \\ \operatorname{vec}(\mathbf{H}_{(i)}^T\mathbf{H}_{(i)}) \end{bmatrix} - \frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\tilde{\mathbf{\Lambda}}_i^{-1}| - \frac{1}{2}\left(\tilde{\mu}_i^T\tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i\right)
\end{aligned} \tag{A.4}$$

Due to conjugacy, $Q(\mathbf{H}_{(i)})$ has a multivariate normal distribution. The log of $Q(\mathbf{H}_{(i)})$ can be rewritten using the above equations:

$$\begin{aligned}
\ln Q(\mathbf{H}_{(i)}) &= \ln \mathcal{N}\left(\mathbf{H}_{(i)}|\mu_i, \mathbf{\Lambda}_i^{-1}\right) \\
&= \begin{bmatrix} \mathbf{\Lambda}_i\mu_i \\ -\frac{1}{2}\operatorname{vec}(\mathbf{\Lambda}_i) \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{(i)}^T \\ \operatorname{vec}(\mathbf{H}_{(i)}^T\mathbf{H}_{(i)}) \end{bmatrix} + \text{normalization terms} \\
&= \begin{bmatrix} \tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i + \langle\mathbf{R}_{(i,i)}^{-1}\rangle\mathbf{X}\mathbf{Y}_{(i)}^T \\ -\frac{1}{2}\operatorname{vec}(\tilde{\mathbf{\Lambda}}_i) - \frac{1}{2}\langle\mathbf{R}_{(i,i)}^{-1}\rangle\operatorname{vec}(\mathbf{X}\mathbf{X}^T) \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_{(i)}^T \\ \operatorname{vec}(\mathbf{H}_{(i)}^T\mathbf{H}_{(i)}) \end{bmatrix} + \text{constant}
\end{aligned} \tag{A.5}$$

In the third line, Equations A.3 and A.4 are substituted into Equation A.2 and like terms are combined. The terms which do not multiply by $\mathbf{H}_{(i)}$ are either constants or become constants under the expectation and are placed into the constant term.

The values in the left vector in the second and third lines are equal [141]. With some manipulation, $\mathbf{\Lambda}_i$ and μ_i can be written in terms of their priors and the data:

$$\mathbf{\Lambda}_i = \tilde{\mathbf{\Lambda}}_i + \mathbf{X}\mathbf{X}^T\langle 1/\mathbf{R}_{(i,i)}\rangle \tag{A.6}$$

$$\mu_i = \mathbf{\Lambda}_i^{-1}(\tilde{\mathbf{\Lambda}}_i\tilde{\mu}_i + \mathbf{X}\mathbf{Y}_{(i)}^T\langle 1/\mathbf{R}_{(i,i)}\rangle) \tag{A.7}$$

R Update

The update for the distribution of the tuning noise covariance $\mathbf{R}$ from Equation 3.18 on page 70 is copied below:

$$Q(\mathbf{R}) = \frac{1}{Z_R} \exp\langle \ln P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y}) \rangle_{\sim Q(\mathbf{R})} \tag{A.8}$$

Factorization using the graphical model (Figure 3.3 on page 68) is applied and terms which are constant under the expectation are moved into the constant term:

$$\begin{aligned}
\ln Q(\mathbf{R}) &= \langle \ln P(\mathbf{H}, \mathbf{R}, \mathbf{X}, \mathbf{Y}) \rangle_{\sim Q(\mathbf{R})} + \text{constant} \\
&= \left\langle \sum_{t=1}^{T} \ln P\left(\mathbf{y}_t | \mathbf{H}, \mathbf{R}\right) + \ln P\left(\mathbf{R} | \tilde{\mathbf{\Psi}}, \tilde{m}\right) \right\rangle_{\sim Q(\mathbf{R})} + \text{constant} \\
&= \left\langle \sum_{t=1}^{T} \ln \mathcal{N}\left(\mathbf{y}_t | \mathbf{H}\mathbf{x}_t, \mathbf{R}\right) + \ln \mathcal{W}^{-1}\left(\mathbf{R} | \tilde{\mathbf{\Psi}}, \tilde{m}\right) \right\rangle_{\sim Q(\mathbf{R})} + \text{constant}
\end{aligned} \tag{A.9}$$

Next, the likelihood term is rewritten:

$$\begin{aligned}
&\sum_{t=1}^{T} \ln \mathcal{N}\left(\mathbf{y}_t | \mathbf{H}\mathbf{x}_t, \mathbf{R}\right) \\
&\quad = -\frac{TN}{2} \ln(2\pi) - \frac{T}{2} \ln |\mathbf{R}| - \frac{1}{2} \operatorname{tr}\left((\mathbf{Y} - \mathbf{H}\mathbf{X})^T \mathbf{R}^{-1} (\mathbf{Y} - \mathbf{H}\mathbf{X})\right) \\
&\quad = \begin{bmatrix} -\frac{1}{2} \operatorname{vec}\left((\mathbf{Y} - \mathbf{H}\mathbf{X})(\mathbf{Y} - \mathbf{H}\mathbf{X})^T\right) \\ -\frac{T}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}(\mathbf{R}^{-1}) \\ \ln |\mathbf{R}| \end{bmatrix} - \frac{TN}{2} \ln(2\pi)
\end{aligned} \tag{A.10}$$

Then the prior term is rewritten:

$$\begin{aligned}
&\ln \mathcal{W}^{-1}\left(\mathbf{R} | \tilde{\mathbf{\Psi}}, \tilde{m}\right) \\
&\quad = \frac{\tilde{m}}{2} \ln |\tilde{\mathbf{\Psi}}| - \frac{\tilde{m} + N + 1}{2} \ln |\mathbf{R}| - \frac{1}{2} \operatorname{tr}\left(\tilde{\mathbf{\Psi}} \mathbf{R}^{-1}\right) - \frac{\tilde{m}N}{2} \ln(2) - \ln \Gamma_N\left(\frac{\tilde{m}}{2}\right) \\
&\quad = \begin{bmatrix} -\frac{1}{2} \operatorname{vec}(\tilde{\mathbf{\Psi}}) \\ -\frac{\tilde{m}+N+1}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}(\mathbf{R}^{-1}) \\ \ln |\mathbf{R}| \end{bmatrix} + \frac{\tilde{m}}{2} \ln |\tilde{\mathbf{\Psi}}| - \frac{\tilde{m}N}{2} \ln(2) - \ln \Gamma_N\left(\frac{\tilde{m}}{2}\right)
\end{aligned} \tag{A.11}$$

$\Gamma_N()$ is the multivariate Gamma function.

Due to conjugacy, $Q(\mathbf{R})$ has an inverse-Wishart distribution. The log of $Q(\mathbf{R})$ can be rewritten using the above equations:

$$\begin{aligned}
\ln Q(\mathbf{R}) &= \ln \mathcal{W}^{-1}(\mathbf{R}|\mathbf{\Psi}, m) \\
&= \begin{bmatrix} -\frac{1}{2}\operatorname{vec}(\mathbf{\Psi}) \\ -\frac{m+N+1}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}(\mathbf{R}^{-1}) \\ \ln|\mathbf{R}| \end{bmatrix} + \text{normalization terms} \\
&= \begin{bmatrix} -\frac{1}{2}\operatorname{vec}(\tilde{\mathbf{\Psi}}) - \frac{1}{2}\operatorname{vec}\left(\left\langle (\mathbf{Y}-\mathbf{HX})(\mathbf{Y}-\mathbf{HX})^T \right\rangle\right) \\ -\frac{\tilde{m}+N+1}{2} - \frac{T}{2} \end{bmatrix}^T \begin{bmatrix} \operatorname{vec}(\mathbf{R}^{-1}) \\ \ln|\mathbf{R}| \end{bmatrix} + \text{constant}
\end{aligned} \tag{A.12}$$

In the third line, Equations A.10 and A.11 are substituted into Equation A.9 and like terms are combined. The terms which do not multiply by $\mathbf{R}$ are constants and are placed into the constant term.

The values in the left vector in the second and third lines are equal [141]. With some manipulation, $\mathbf{\Psi}$ and m can be written in terms of their priors and the data:

$$\mathbf{\Psi} = \tilde{\mathbf{\Psi}} + \langle (\mathbf{Y}-\mathbf{HX})(\mathbf{Y}-\mathbf{HX})^T \rangle \tag{A.13}$$

$$m = \tilde{m} + T \tag{A.14}$$

The expectation in Equation A.13 has a quadratic term and requires accounting for the covariance of the tuning coefficients for each neuron:

$$\langle (\mathbf{Y}-\mathbf{HX})(\mathbf{Y}-\mathbf{HX})^T \rangle = (\mathbf{Y}-\langle \mathbf{H} \rangle \mathbf{X})(\mathbf{Y}-\langle \mathbf{H} \rangle \mathbf{X})^T + \mathbf{W} \tag{A.15}$$

The N by N matrix $\mathbf{W}$ holds the contribution from the covariance of the tuning coefficients. Since tuning coefficient distributions are factorized across neurons, covariance of tuning coefficients only contributes to the diagonal of $\mathbf{\Psi}$. $\mathbf{W}$ is computed by:

$$\mathbf{W}_{(i,j)} = \begin{cases} \operatorname{tr}(\mathbf{X}^T \mathbf{\Lambda}_i^{-1} \mathbf{X}), & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{A.16}$$

This comes from the multivariate generalization of the identity:

$$\langle (ab)^2 \rangle = (\langle a \rangle b)^2 + b^2 \,\mathrm{VAR}(a) \tag{A.17}$$

Here, a is a scalar random variable, b is a scalar constant, and $\mathrm{VAR}(a)$ is the variance of a.

References

[1] P. Artemiadis, G. Shakhnarovich, C. Vargas-Irwin, M. J. Black, and J. P. Donoghue. Decoding grasp aperture from motor-cortical population activity. In *3rd International IEEE EMBS Conference on Neural Engineering*, pages 518–521, 2007. (cited on p. 19)

[2] H. Attias. A variational Bayesian framework for graphical models. In *Advances in Neural Information Processing Systems 12*, pages 209–215. MIT Press, 2000. (cited on p. 66, 67, 149)

[3] A. Bar-Hillel, A. Spiro, and E. Stark. Spike sorting: Bayesian clustering of non-stationary data. *Journal of Neuroscience Methods*, 157(2):303–316, 2006. (cited on p. 148)

[4] R. Barbieri, L. M. Frank, D. P. Nguyen, M. C. Quirk, V. Solo, M. A. Wilson, and E. N. Brown. A Bayesian decoding algorithm for analysis of information encoding in neural ensembles. In *Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE*, volume 2, pages 4483–4486, 2004. (cited on p. 15, 17, 18, 173)

[5] A. Bashashati, M. Fatourechi, R. K. Ward, and G. E. Birch. A survey of signal processing algorithms in brain-computer interfaces based on electrical brain signals. *Journal of Neural Engineering*, 4:R32–57, 2007. (cited on p. 13)

[6] M. J. Beal. *Variational Algorithms for Approximate Bayesian Inference*. Dissertation, University College London, 2003. (cited on p. 46, 66, 67, 175, 178)

[7] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006. (cited on p. 58, 64, 66, 67, 70, 149, 151, 178)

[8] M. J. Black, E. Bienenstock, J. P. Donoghue, M. D. Serruya, W. Wu, and Y. Gao. Connecting brains with machines: the neural control of 2D cursor movement. In *First International IEEE EMBS Conference on Neural Engineering*, pages 580–583, 2003. (cited on p. 14, 17, 19)

[9] A. E. Brockwell, R. E. Kass, and A. B. Schwartz. Statistical signal processing and the motor cortex. In *Proceedings of the IEEE*, volume 95, pages 881–898, 2007. (cited on p. 12, 15, 17, 18, 173)

[10] A. E. Brockwell, A. L. Rojas, and R. E. Kass. Recursive Bayesian decoding of motor cortical signals by particle filtering. *Journal of Neurophysiology*, 91:1899–1907, 2004. (cited on p. 12, 15, 17, 18, 173)

[11] J. M. Carmena, M. A. Lebedev, R. E. Crist, J. E. O'Doherty, D. M. Santucci, D. F. Dimitrov, P. G. Patil, C. S. Henriquez, and M. A. Nicolelis. Learning to control a brain-machine interface for reaching and grasping by primates. *PLoS Biology*, 1(E42), 2003. (cited on p. 5, 11, 14)

[12] C. A. Chestek, A. P. Batista, G. Santhanam, B. M. Yu, A. Afshar, J. P. Cunningham, V. Gilja, S. I. Ryu, M. M. Churchland, and K. V. Shenoy. Single-neuron stability during repeated reaching in macaque premotor cortex. *Journal of Neuroscience*, 27:10742–10750, 2007. (cited on p. 6, 41, 42, 43, 75, 133)

[13] K. Choi, H. Hirose, Y. Sakurai, T. Iijima, and Y. Koike. Prediction of arm trajectory from the neural activities of the primary motor cortex with modular connectionist architecture. *Neural Networks*, 22(9):1214–1223, 2009. (cited on p. 16)

[14] Y. Chu, Z. Li, Y. Su, and Z. Pizlo. Mental representation in problem solving: the role of direction and insight. *Journal of Problem Solving*, 2010. Accepted for publication. (cited on p. 204)

[15] T. Cowan and D. Taylor. Predicting reach goal in a continuous workspace for command of a brain-controlled upper-limb neuroprosthesis. In *2nd International IEEE EMBS Conference on Neural Engineering*, pages 74–77, 2005. (cited on p. 12)

[16] H. Cox. On the estimation of state variables and parameters for noisy dynamic systems. *IEEE Transactions on Automatic Control*, 9:5–12, 1964. (cited on p. 44)

[17] S. Darmanjian, S. P. Kim, M. C. Nechyba, S. Morrison, J. C. Principe, J. Wessberg, and M. A. L. Nicolelis. Bimodal brain-machine interface for motor control of robotic prosthetic. In *2003 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, volume 4, pages 3612–3617, 2003. (cited on p. 16)

[18] S. Darmanjian and J. Principe. Spatial-temporal clustering of neural data using linked-mixtures of hidden Markov models. *EURASIP Journal on Advances in Signal Processing*, 2009. (cited on p. 47)

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. (cited on p. 46, 67, 81, 149)

[20] A. S. Dickey, A. Suminski, Y. Amit, and N. G. Hatsopoulos. Single-unit stability using chronically implanted multielectrode arrays. *Journal of Neurophysiology*, 102(2):1331–1339, 2009. (cited on p. 6, 146)

[21] J. DiGiovanna, B. Mahmoudi, J. Fortes, J. C. Principe, and J. C. Sanchez. Coadaptive brain-machine interface via reinforcement learning. *IEEE Transactions on Biomedical Engineering*, 56(1):54–64, 2009. (cited on p. 16, 47)

[22] J. P. Donoghue, A. Nurmikko, M. Black, and L. R. Hochberg. Assistive technology and robotic control using motor cortex ensemble-based neural interface systems in humans with tetraplegia. *Journal of Physiology*, 579:603–611, 2007. (cited on p. 74)

[23] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001. (cited on p. 100)

[24] U. T. Eden, L. M. Frank, R. Barbieri, V. Solo, and E. N. Brown. Dynamic analysis of neural encoding by point process adaptive filtering. *Neural Computation*, 16:971–998, 2004. (cited on p. 15, 17, 18, 49, 173)

[25] U. T. Eden, W. Truccolo, M. Fellows, J. P. Donoghue, and E. N. Brown. Reconstruction of hand movement trajectories from a dynamic ensemble of spiking motor cortical neurons. In *Engineering in Medicine and Biology Society, IEMBS '04. 26th Annual International Conference of the IEEE*, volume 2, pages 4017–4020, 2004. (cited on p. 15, 17, 18, 173)

[26] A. A. Emondi, S. P. Rebrik, A. V. Kurgansky, and K. D. Miller. Tracking neurons recorded from tetrodes across time. *Journal of Neuroscience Methods*, 135(1-2):95 – 105, 2004. (cited on p. 148)

[27] J. Fisher and M. Black. Motor cortical decoding using an autoregressive moving average model. In *Engineering in Medicine and Biology Society, IEMBS 2005. 27th Annual International Conference of the IEEE*, pages 2130–2133, 2005. (cited on p. 16)

[28] G. W. Fraser, S. M. Chase, A. Whitford, and A. B. Schwartz. Control of a brain-computer interface without spike sorting. *Journal of Neural Engineering*, 6:055004, 2009. (cited on p. 13, 15, 17)

[29] Q. G. Fu, D. Flament, J. D. Coltz, and T. J. Ebner. Temporal encoding of movement kinematics in the discharge of primate primary motor and premotor neurons. *Journal of Neurophysiology*, 73:836–854, 1995. (cited on p. 12)

[30] G. J. Gage, K. A. Ludwig, K. J. Otto, E. L. Ionides, and D. R. Kipke. Naive coadaptive cortical control. *Journal of Neural Engineering*, 2:52–63, 2005. (cited on p. 47)

[31] K. Ganguly and J. M. Carmena. Emergence of a stable cortical map for neuroprosthetic control. *PLoS Biol*, 7:e1000153, 2009. (cited on p. 6, 42, 43, 44, 75, 132, 133, 144, 173, 174)

[32] Y. Gao, E. Bienenstock, S. Shoham, and J. P. Donoghue. Probabilistic inference of hand motion from neural activity in motor cortex. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002. (cited on p. 15, 17)

[33] Y. Gao, M. Black, E. Bienenstock, W. Wu, and J. P. Donoghue. A quantitative comparison of linear and non-linear models of motor cortical activity for the encoding and decoding of arm motions. In *First International IEEE EMBS Conference on Neural Engineering*, pages 189–192, 2003. (cited on p. 13, 15, 17)

[34] J. Gasthaus, F. Wood, D. Grr, and Y. W. Teh. Dependent Dirichlet process spike sorting. In *Advances in Neural Information Processing Systems 22*. MIT Press, 2009. (cited on p. 148, 176, 178)

[35] J. L. Gauvain and C. H. Lee. Bayesian learning of Gaussian mixture densities for hidden Markov models. In *DARPA Speech and Natural Language Workshop*, pages 272–277, 1991. (cited on p. 149)

[36] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey. On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *Journal of Neuroscience*, 2:1527–1537, 1982. (cited on p. 11)

[37] A. P. Georgopoulos, A. B. Schwartz, and R. E. Kettner. Neuronal population coding of movement direction. *Science*, 233:1416–1419, 1986. (cited on p. 5, 13)

[38] Z. Ghahramani and S. T. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems 11*. MIT Press, 1999. (cited on p. 46, 175)

[39] V. Gilja, P. Nuyujukian, C. A. Chestek, J. P. Cunningham, B. M. Yu, S. J. Ryu, and K. V. Shenoy. Addressing the impact of instability on closed loop

neural prosthetic control. Poster presented at the 2009 Society for Neuroscience Annual Meeting, Chicago, IL, 2009. (cited on p. 49)

[40] B. D. Grant, Z. Li, T. L. Hanson, J. E. O'Doherty, M. A. Lebedev, and M. A. Nicolelis. Automated spike sorting of multiunit data for brain-machine interface applications. Poster presented at the 2007 Society for Neuroscience Annual Meeting, San Diego, CA, 2007. (cited on p. 81)

[41] K. D. Harris, D. A. Henze, J. Csicsvari, H. Hirase, and G. Buzsaki. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *Journal of Neurophysiology*, 84(1):401–414, 2000. (cited on p. 147)

[42] N. G. Hatsopoulos, J. Joshi, and J. G. O'Leary. Decoding continuous and discrete motor behaviors using motor and premotor cortical ensembles. *Journal of Neurophysiology*, 92:1165–1174, 2004. (cited on p. 5, 14, 16)

[43] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, Upper Saddle River, NJ, 1996. (cited on p. 14, 26)

[44] L. R. Hochberg, M. D. Serruya, G. M. Friehs, J. A. Mukand, M. Saleh, A. H. Caplan, A. Branner, D. Chen, R. D. Penn, and J. P. Donoghue. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442:164–171, 2006. (cited on p. 5, 11, 14, 47)

[45] J. Hu, J. Si, B. P. Olson, R. S. Clement, and J. He. Decoding motor cortical spike trains for brain machine interface applications. In *Engineering in Medicine and Biology Society, 2003. 25th Annual International Conference of the IEEE*, pages 2071–2074, 2003. (cited on p. 16)

[46] J. Hu, J. Si, B. P. Olson, and J. He. Feature detection in motor cortical spikes by principal component analysis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 13(3):256–262, 2005. (cited on p. 16)

[47] R. E. Isaacs, D. J. Weber, and A. B. Schwartz. Work toward real-time control of a cortical neural prothesis. *IEEE Transactions on Rehabilitation Engineering*, 8:196–198, 2000. (cited on p. 12, 16)

[48] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999. (cited on p. 66, 67)

[49] S. Julier, J. Uhlmann, and H. F. Durrant-Whyte. A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, 45:477–482, 2000. (cited on p. 28)

[50] S. J. Julier. The scaled unscented transformation. In *Proceedings of the American Control Conference*, volume 6, pages 4555–4559, 2002. (cited on p. 27)

[51] S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of the American Control Conference*, volume 3, pages 1628–1632, 1995. (cited on p. 27, 28, 30, 33)

[52] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960. (cited on p. 17, 18)

[53] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME–Journal of Basic Engineering*, 83:95–108, 1961. (cited on p. 17)

[54] C. Kemere, K. V. Shenoy, and T. H. Meng. Model-based neural decoding of reaching movements: A maximum likelihood approach. *IEEE Transactions on Biomedical Engineering*, 51(6):925–932, 2004. (cited on p. 15)

[55] P. R. Kennedy and R. A. Bakay. Restoration of neural output from a paralyzed patient by a direct brain connection. *Neuroreport*, 9(8):1707–1711, 1998. (cited on p. 13)

[56] P. R. Kennedy, R. A. Bakay, M. M. Moore, K. Adams, and J. Goldwaithe. Direct control of a computer from the human central nervous system. *IEEE Transactions on Rehabilitation Engineering*, 8(2):198–202, 2000. (cited on p. 13)

[57] H. K. Kim, J. M. Carmena, S. J. Biggs, T. L. Hanson, M. A. L. Nicolelis, and M. A. Srinivasan. The muscle activation method: an approach to impedance control of brain-machine interfaces through a musculoskeletal model of the arm. *IEEE Transactions on Biomedical Engineering*, 54(8):1520 – 1529, 2007. (cited on p. 4, 16)

[58] S. P. Kim, J. M. Carmena, M. A. Nicolelis, and J. C. Principe. Multiresolution representations and data mining of neural spikes for brain-machine interfaces. In *2nd International IEEE EMBS Conference on Neural Engineering*, pages 221 –224, 2005. (cited on p. 17)

[59] S. P. Kim, Y. N. Rao, D. Erdogmus, J. C. Sanchez, M. A. L. Nicolelis, and J. C. Principe. Determining patterns in neural activity for reaching movements using nonnegative matrix factorization. *EURASIP Journal on Applied Signal Processing*, 19:3113–3121, 2005. (cited on p. 17)

[60] S. P. Kim, J. C. Sanchez, D. Erdogmus, Y. N. Rao, J. C. Principe, and M. A. Nicolelis. Modeling the relation from motor cortical neuronal firing to hand movements using competitive linear filters and a MLP. In *International Joint Conference on Neural Networks*, volume 1, pages 66–70, 2003. (cited on p. 12, 16)

[61] S. P. Kim, J. C. Sanchez, Y. N. Rao, D. Erdogmus, J. M. Carmena, M. A. Lebedev, M. A. Nicolelis, and J. C. Principe. A comparison of optimal MIMO linear and nonlinear models for brain-machine interfaces. *Journal of Neural Engineering*, 3:145–161, 2006. (cited on p. 87)

[62] S. P. Kim, J. D. Simeral, L. R. Hochberg, J. P. Donoghue, G. M. Friehs, and M. J. Black. Multi-state decoding of point-and-click control signals from motor cortical activity in a human with tetraplegia. In *Third International IEEE EMBS Conference on Neural Engineering*, pages 486–489, 2007. (cited on p. 16)

[63] S. P. Kim, F. Wood, M. Fellows, J. P. Donoghue, and M. J. Black. Statistical analysis of the non-stationarity of neural population codes. In *First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 811–816, 2006. (cited on p. 6, 41, 42)

[64] R. E. Kopp and R. J. Orford. Linear regression applied to system identication for adaptive control systems. *American Institute of Aeronautics and Astronautics Journal*, 1:2300–06, 1963. (cited on p. 44)

[65] S. Koyama, S. M. Chase, A. S. Whitford, M. Velliste, A. B. Schwartz, and R. E. Kass. Comparison of brain-computer interface decoding algorithms in open-loop and closed-loop control. *Journal of Computational Neuroscience*, 2009. (cited on p. 13)

[66] M. A. Lebedev, J. M. Carmena, J. E. O'Doherty, M. Zacksenhouse, C. S. Henriquez, J. C. Principe, and M. A. L. Nicolelis. Cortical ensemble adaptation to represent velocity of an artificial actuator controlled by a brain-machine interface. *Journal of Neuroscience*, 25(19):4681–4693, 2005. (cited on p. 42, 90)

[67] M. S. Lewicki. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):53–78, 1998. (cited on p. 147, 150)

[68] C. S. Li, C. Padoa-Schioppa, and E. Bizzi. Neuronal correlates of motor performance and motor learning in the primary motor cortex of monkeys adapting to an external force field. *Neuron*, 30(2):593–607, 2001. (cited on p. 6, 41, 42)

[69] X. R. Li and V. P. Jilkov. Survey of maneuvering target tracking-part IV: decision-based methods. In *Proceedings of the 2002 SPIE Conference on Signal and Data Processing of Small Targets*, volume 4728-60, 2002. (cited on p. 177)

[70] X. R. Li and V. P. Jilkov. Survey of maneuvering target tracking-part I: dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1333–1364, 2003. (cited on p. 177)

[71] X. R. Li and V. P. Jilkov. Survey of maneuvering target tracking-part V: multiple-model methods. *IEEE Transactions on Aerospace and Electronic Systems*, 41(4):1255–1321, 2005. (cited on p. 177)

[72] Z. Li, J. E. O'Doherty, T. L. Hanson, M. A. Lebedev, C. S. Henriquez, and M. A. L. Nicolelis. N-th order Kalman filter improves the performance of a brain machine interface for reaching. Poster presented at the 2007 Society for Neuroscience Annual Meeting, San Diego, CA, 2007. (cited on p. 14, 17, 19)

[73] Z. Li, J. E. O'Doherty, T. L. Hanson, M. A. Lebedev, C. S. Henriquez, and M. A. L. Nicolelis. Unscented Kalman filter for brain-machine interfaces. *PLoS ONE*, 4(7):e6243, 2009. (cited on p. 6, 204)

[74] X. Liao, J. Fischer, T. Milekovic, J. Rickert, A. Aertsen, and C. Mehring. Co-adaptivity during learning to control a myoelectric interface. Poster presented at the 2009 Society for Neuroscience Annual Meeting, Chicago, IL, 2009. (cited on p. 49)

[75] S. Lin, J. Si, and A. B. Schwartz. Self-organization of firing activities in monkey's motor cortex: trajectory computation from spike signals. *Neural Computation*, 9:607–621, 1997. (cited on p. 12)

[76] M. D. Linderman, V. Gilja, G. Santhanam, A. Afshar, S. Ryu, T. H. Meng, and K. V. Shenoy. Neural recording stability of chronic electrode arrays in freely behaving primates. In *Engineering in Medicine and Biology Society, EMBS '06. 28th Annual International Conference of the IEEE*, pages 4387–4391, 2006. (cited on p. 6, 146)

[77] L. Ljung. Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, 24(1):36–50, 1979. (cited on p. 45)

[78] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification*. MIT Press, Cambridge, MA, 1987. (cited on p. 49, 50, 55)

[79] D. W. Moran and A. B. Schwartz. Motor cortical representation of speed and direction during reaching. *Journal of Neurophysiology*, 82:2676–2692, 1999. (cited on p. 11, 25, 92)

[80] G. H. Mulliken, S. Musallam, and R. A. Andersen. Decoding trajectories from posterior parietal cortex ensembles. *Journal of Neuroscience*, 28:12913–12926, 2008. (cited on p. 16)

[81] S. Musallam, B. D. Corneil, B. Greger, H. Scherberger, and R. A. Andersen. Cognitive control signals for neural prosthetics. *Science*, 305:258–262, 2004. (cited on p. 16, 47)

[82] N. S. Narayanan, E. Y. Kimchi, and M. Laubach. Redundancy and synergy of neuronal ensembles in motor cortex. *Journal of Neuroscience*, 25(17):4207–4216, 2005. (cited on p. 53)

[83] A. Navot, L. Shpigelman, N. Tishby, and E. Vaadia. Nearest neighbor based feature selection for regression and its application to neural acitivty. In *Advances in Neural Information Processing Systems 18*. MIT Press, 2005. (cited on p. 13)

[84] A. T. Nelson. *Nonlinear Estimation and Modeling of Noisy Time-Series by Dual Kalman Filtering Methods*. Dissertation, Oregon Graduate Institute of Science and Technology, 2000. (cited on p. 44, 45)

[85] L. W. Nelson and E. Stear. The simultaneous on-line estimation of parameters and states in linear systems. *IEEE Transactions on Automatic Control*, 21:94–98, 1976. (cited on p. 45)

[86] B. P. Olson, J. Si, J. Hu, and J. He. Closed-loop cortical control of direction using support vector machines. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 13(1):72 – 80, 2005. (cited on p. 16)

[87] C. Padoa-Schioppa, C. S. Li, and E. Bizzi. Neuronal activity in the supplementary motor area of monkeys adapting to a new dynamic environment. *Journal of Neurophysiology*, 91:449–473, 2004. (cited on p. 6, 41, 42)

[88] Z. Pizlo and Z. Li. Solving combinatorial problems: 15-puzzle. *Memory & Cognition*, 33(6):1069–1084, 2005. (cited on p. 204)

[89] Z. Pizlo, E. Stefanov, J. Saalweachter, Z. Li, Y. Haxhimusa, and W. G. Kropatsch. Traveling salesman problem: a foveating pyramid model. *Journal of Problem Solving*, 1(1):83–101, 2006. (cited on p. 204)

[90] C. Pouzat, O. Mazor, and G. Laurent. Using noise signature to optimize spike-sorting and to assess neuronal classification quality. *Journal of Neuroscience Methods*, 122(1):43 – 57, 2002. (cited on p. 147)

[91] R. Q. Quiroga, Z. Nadasdy, and Y. Ben-Shaul. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Computation*, 16(8):1661–1687, 2004. (cited on p. 147)

[92] H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965. (cited on p. 57)

[93] A. C. Rencher and G. B. Schaalje. *Linear Models in Statistics*. Wiley-Interscience, Hoboken, NJ, 2nd edition, 2008. (cited on p. 59)

[94] J. Rickert, D. Braun, and C. Mehring. Unsupervised adaptive Kalman filter for decoding non-stationary brain signals. Poster presented at the 2007 Society for Neuroscience Annual Meeting, San Diego, CA, 2007. (cited on p. 49)

[95] C. P. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, New York, 2nd edition, 2004. (cited on p. 66)

[96] U. Rokni, A. G. Richardson, E. Bizzi, and H. S. Seung. Motor learning with unstable neural representations. *Neuron*, 54:653–666, 2007. (cited on p. 6, 41, 42, 43)

[97] D. Rotermund, U. A. Ernst, and K. R. Pawelzik. Towards on-line adaptation of neuro-prostheses with neuronal evaluation signals. *Biological Cybernetics*, 95(3):243–257, 2006. (cited on p. 48)

[98] J. C. Sanchez, D. Erdogmus, Y. N. Rao, S. P. Kim, M. A. L. Nicolelis, J. Wessberg, and J. C. Principe. Interpreting neural activity through linear and nonlinear models for brain machine interfaces. In *Engineering in Medicine and Biology Society, 2003. 25th Annual International Conference of the IEEE*, volume 3, pages 2160–2163, 2003. (cited on p. 16)

[99] J. C. Sanchez, S. P. Kim, D. Erdogmus, Y. N. Rao, J. C. Principe, J. Wessberg, and M. A. L. Nicolelis. Input-output mapping performance of linear and nonlinear models for estimating hand trajectories from cortical neuronal firing

patterns. In *12th IEEE Workshop on Neural Networks for Signal Processing*, pages 139–148, 2002. (cited on p. 12, 16, 87)

[100] J. C. Sanchez, J. C. Principe, J. M. Carmena, M. A. Lebedev, and M. A. L. Nicolelis. Simultaneus prediction of four kinematic variables for a brain-machine interface using a single recurrent neural network. In *Engineering in Medicine and Biology Society, IEMBS '04. 26th Annual International Conference of the IEEE*, pages 5321–5324, 2004. (cited on p. 16)

[101] G. Santhanam, S. I. Ryu, B. M. Yu, A. Afshar, and K. V. Shenoy. A high-performance brain-computer interface. *Nature*, 442:195–198, 2006. (cited on p. 16)

[102] G. Santhanam, B. M. Yu, V. Gilja, S. I. Ryu, A. Afshar, M. Sahani, and K. V. Shenoy. Factor-analysis methods for higher-performance neural prostheses. *Journal of Neurophysiology*, 2009. (cited on p. 16, 17)

[103] G. Schalk, J. R. Wolpaw, D. J. McFarland, and G. Pfurtscheller. EEG-based communication: presence of an error potential. *Clinical Neurophysiology*, 111:2138–2144(7), 2000. (cited on p. 48)

[104] M. D. Serruya, N. G. Hatsopoulos, L. Paninski, M. R. Fellows, and J. P. Donoghue. Instant neural control of a movement signal. *Nature*, 416(141-142), 2002. (cited on p. 5, 11, 14)

[105] G. Shakhnarovich, S. P. Kim, and M. J. Black. Nonlinear physically-based models for decoding motor-cortical population activity. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2006. (cited on p. 4, 16)

[106] S. Shoham, L. Paninski, M. Fellows, N. G. Hatsopoulos, J. P. Donoghue, and R. A. Normann. Statistical encoding model for a primary motor cortical brain-machine interface. *IEEE Transactions on Biomedical Engineering*, 52(7):1312–1322, 2005. (cited on p. 13, 15, 17)

[107] L. Shpigelman, H. Lalazar, and E. Vaadia. Kernel-ARMA for hand tracking and brain-machine interfacing during 3D motor control. In *Advances in Neural Information Processing Systems 21*. MIT Press, 2009. (cited on p. 16, 48)

[108] L. Shpigelman, R. Paz, E. Vaadia, and Y. Singer. A temporal kernel-based model for tracking hand movements from neural activities. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004. (cited on p. 13, 16)

[109] L. Shpigelman, Y. Singer, R. Paz, and E. Vaadia. Spikernels: embedding spiking neurons in inner-product spaces. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. (cited on p. 13, 16)

[110] L. Shpigelman, Y. Singer, R. Paz, and E. Vaadia. Spikernels: predicting arm movements by embedding population spike rate patterns in inner-product spaces. *Neural Computation*, 17:671–690, 2005. (cited on p. 13, 16)

[111] R. H. Shumway and D. S. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 3(4):253–264, 1982. (cited on p. 46, 175)

[112] L. Srinivasan, U. T. Eden, S. K. Mitter, and E. N. Brown. General-purpose filter design for neural prosthetic devices. *Journal of Neurophysiology*, 98:2456–2475, 2007. (cited on p. 15, 17, 18, 49, 173)

[113] L. Srinivasan, U. T. Eden, A. S. Willsky, and E. N. Brown. Goal-directed state equation for tracking reaching movements using neural signals. In *2nd International IEEE EMBS Conference on Neural Engineering*, pages 352–355, 2005. (cited on p. 15)

[114] L. Srinivasan, U. T. Eden, A. S. Willsky, and E. N. Brown. A state-space analysis for reconstruction of goal-directed movements using neural signals. *Neural Computation*, 18(2465-2494), 2006. (cited on p. 15)

[115] R. L. Stratonovich. Optimum nonlinear systems which bring about a separation of a signal with constant parameters from noise. *Radiofizika*, 2(6):892–901, 1959. (cited on p. 17)

[116] R. L. Stratonovich. Application of the Markov processes theory to optimal filtering. *Radio Engineering and Electronic Physics*, 5(11):1–19, 1960. (cited on p. 17)

[117] P. Sykacek and S. Roberts. Adaptive classification by variational Kalman filtering. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2002. (cited on p. 48, 56, 177)

[118] P. Sykacek, S. J. Roberts, and M. Stokes. Adaptive BCI based on variational Bayesian Kalman filtering: an empirical evaluation. *IEEE Transactions on Biomedical Engineering*, 51(5):719–727, 2004. (cited on p. 48, 56, 177)

[119] T. Takekawa, Y. Isomura, and T. Fukai. Accurate spike sorting for multi-unit recordings. *European Journal of Neuroscience*, 31(2):263–272, 2010. (cited on p. 147)

[120] D. M. Taylor, S. I. Tillery, and A. B. Schwartz. Direct cortical control of 3D neuroprosthetic devices. *Science*, 296:1829–1832, 2002. (cited on p. 6, 11, 13, 41, 42, 44, 47, 74, 92, 93)

[121] J. A. Ting, A. D'Souza, K. Yamamoto, T. Yoshioka, D. Hoffman, S. Kakei, L. Sergio, J. Kalaska, M. Kawato, P. Strick, and S. Schaal. Predicting EMG data from M1 neurons with variational Bayesian least squares. In *Advances in Neural Information Processing Systems 18*. MIT Press, 2005. (cited on p. 17)

[122] J. A. Ting, A. D'Souza, K. Yamamoto, T. Yoshioka, D. Hoffman, S. Kakei, L. Sergio, J. Kalaska, M. Kawato, P. Strick, and S. Schaal. Variational Bayesian last squares: An application to brain-machine interface data. *Neural Networks*, 21(8):1112–1131, 2008. (cited on p. 17)

[123] W. Truccolo and J. P. Donoghue. Nonparametric modeling of neural point processes via stochastic gradient boosting regression. *Neural Computation*, 19:672–705, 2007. (cited on p. 13)

[124] W. Truccolo, U. T. Eden, M. Fellows, J. P. Donoghue, and E. N. Brown. A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects. *Journal of Neurophysiology*, 93:1074–1089, 2005. (cited on p. 15, 17, 18, 173)

[125] W. Truccolo, G. M. Friehs, J. P. Donoghue, and L. R. Hochberg. Primary motor cortex tuning to intended movement kinematics in humans with tetraplegia. *Journal of Neuroscience*, 28(5):1163–1178, 2008. (cited on p. 6, 12, 15, 17, 18, 41, 42, 173)

[126] W. Truccolo, L. R. Hochberg, and J. P. Donoghue. Collective dynamics in human and monkey sensorimotor cortex: predicting single neuron spikes. *Nature Neuroscience*, 13(1):105–111, 2009. (cited on p. 13, 173)

[127] R. S. Turner and M. E. Andersen. Pallidal discharge related to the kinematics of reaching movements in two dimensions. *Journal of Neurophysiology*, 77:1051–1074, 1997. (cited on p. 12)

[128] R. van der Merwe. *Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models*. Dissertation, Oregon Health and Science University, 2004. (cited on p. 27, 28, 30, 33)

[129] C. E. Vargas-Irwin, P. K. Artemiadis, G. Shakhnarovich, M. J. Black, and J. P. Donoghue. Neural correlates of grip aperture in monkey primary motor cortex.

Poster presented at the 2007 Society for Neuroscience Annual Meeting, San Diego, CA, 2007. (cited on p. 19)

[130] M. Velliste, S. Perel, M. C. Spalding, A. S. Whitford, and A. B. Schwartz. Cortical control of a prosthetic arm for self-feeding. *Nature*, 453:1098–1101, 2008. (cited on p. 74)

[131] V. Ventura. Spike train decoding without spike sorting. *Neural Computation*, 20:923–963, 2008. (cited on p. 16)

[132] V. Ventura. Automatic spike sorting using tuning information. *Neural Computation*, 21:2466–2501, 2009. (cited on p. 147, 178)

[133] V. Ventura. Traditional waveform based spike sorting yields biased rate code estimates. *Proceedings of the National Academy of Sciences*, 106(17):6921–6926, 2009. (cited on p. 178)

[134] R. Wahnoun, J. He, and S. I. Helms Tillery. Selection and parameterization of cortical neurons for neuroprosthetic control. *Journal of Neural Engineering*, 3(2):162–171, 2006. (cited on p. 13, 47, 74)

[135] R. Wahnoun, S. I. Helms Tillery, and J. He. Neuron selection and visual training for population vector based cortical control. In *Engineering in Medicine and Biology Society, IEMBS '04. 26th Annual International Conference of the IEEE*, volume 2, pages 4607–4610, 2004. (cited on p. 13, 47, 74)

[136] E. A. Wan, R. van der Merwe, and A. T. Nelson. Dual estimation and the unscented transform. In *Advances in Neural Information Processing Systems 12*. MIT Press, 2000. (cited on p. 45)

[137] Y. Wang, A. R. C. Paiva, and J. C. Principe. A Monte Carlo sequential estimation for point process optimum filtering. In *IJCNN '06. International Joint Conference on Neural Networks*, pages 1846–1850, 2006. (cited on p. 15, 17, 18, 173)

[138] Y. Wang, A. R. C. Paiva, J. C. Principe, and J. C. Sanchez. Sequential Monte Carlo point-process estimation of kinematics from neural spiking activity for brain-machine interfaces. *Neural Computation*, 21:2894–2930, 2009. (cited on p. 15, 17, 18, 173)

[139] J. Wessberg, C. R. Stambaugh, J. D. Kralik, P. D. Beck, M. Laubach, J. K. Chapin, J. Kim, S. J. Biggs, M. A. Srinivasan, and M. A. Nicolelis. Real-time

prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, 408:361–365, 2000. (cited on p. 5, 12, 14, 16)

[140] A. S. Willsky. A survey of design methods for failure detection in dynamic systems. *Automatica*, 12:601–611, 1976. (cited on p. 66)

[141] J. Winn and C. M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6:661–694, 2005. (cited on p. 66, 67, 70, 72, 184, 186)

[142] M. T. Wolf and J. W. Burdick. Bayesian clustering and tracking of neuronal signals for autonomous neural interfaces. In *CDC 2008. 47th IEEE Conference on Decision and Control*, pages 1992–1999, 2008. (cited on p. 148, 176)

[143] J. R. Wolpaw and D. 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, 2004. (cited on p. 47)

[144] D. S. Won. *An Information-Theoretic Analysis of Spike Processing in a Neuroprosthetic Model*. Dissertation, Duke University, 2007. (cited on p. 146)

[145] F. Wood, S. Goldwater, and M. J. Black. A non-parametric Bayesian approach to spike sorting. In *Engineering in Medicine and Biology Society, IEMBS '06. 28th Annual International Conference of the IEEE*, volume 1, pages 1165–1168, 2006. (cited on p. 147, 149)

[146] F. Wood, S. Roth, and M. J. Black. Modeling neural population spiking activity with Gibbs distributions. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2005. (cited on p. 13)

[147] W. Wu, M. J. Black, Y. Gao, E. Bienenstock, M. D. Serruya, A. Shaikhouni, and J. P. Donoghue. Neural deocoding of cursor motion using a Kalman filter. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003. (cited on p. 11, 14, 17, 19)

[148] W. Wu, M. J. Black, D. Mumford, Y. Gao, E. Bienenstock, and J. P. Donoghue. Modeling and decoding motor cortical activity using a switching Kalman filter. *IEEE Transactions on Biomedical Engineering*, 51(6):933–942, 2004. (cited on p. 14, 17, 19, 90)

[149] W. Wu, Y. Gao, E. Bienenstock, J. P. Donoghue, and M. Black. Bayesian population decoding of motor cortical activity using a Kalman filter. *Neural Computation*, 18(80-118), 2006. (cited on p. 14, 17, 19, 102, 171)

[150] W. Wu and N. G. Hatsopoulos. Real-time decoding of nonstationary neural activity in motor cortex. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 16(3):213–222, 2008. (cited on p. 48, 118, 173, 174)

[151] W. Wu, A. Shaikhouni, J. P. Donoghue, and M. Black. Closed-loop neural control of cursor motion using a Kalman filter. In *Engineering in Medicine and Biology Society, IEMBS '04. 26th Annual International Conference of the IEEE*, volume 2, pages 4126–4129, 2004. (cited on p. 13, 14, 17, 19)

[152] Z. Yang, Q. Zhao, and W. Liu. Improving spike separation using waveform derivatives. *Journal of Neural Engineering*, 6(4):046006, 2009. (cited on p. 147, 150)

[153] O. Zobay. Mean field inference for the Dirichlet process mixture model. *Electronic Journal of Statistics*, 3:507–545, 2009. (cited on p. 178)

Biography

Zheng Li (Mandarin Chinese: 李征) was born January 7, 1983 in Guangzhou, China. He received a B.S. with honors and highest distinction in Computer Science and Mathematics from Purdue University in 2004 and a Ph.D. in Computer Science from Duke University in 2010. He was the recipient of a James B. Duke Fellowship and a Clinical Faculty Arts and Sciences Fellowship from the Duke University Graduate School. He has co-authored articles on *Unscented Kalman filter for brain-machine interfaces* [73], *Mental representation in problem solving: the role of direction and insight* [14], *Traveling salesman problem: a foveating pyramid model* [89], and *Solving combinatorial problems: 15-puzzle* [88].