Proceedings of the 1st International IEEE EMBS
Conference on Neural Engineering
Capri Island, Italy • March 20-22, 2003

Methods for Estimating Neural Step Sequences in Neural Prosthetic Applications

Gopal Santhanam[1] and Krishna V. Shenoy[1,2]
[1]Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA
[2]Neurosciences Program, Stanford University School of Medicine, Stanford, CA, 94305, USA

Abstract— **The prospect of helping disabled patients by translating neural activity from the brain into control signals for prosthetic devices is currently being realized. Initial proof-of-concept systems have demonstrated the need for faster and more accurate estimation algorithms, without requiring unrealistically many neurons. To address this need, we recently reported the plan-movement maximum likelihood (PMML) algorithm. It combines plan activity, specifying reach end-point, with movement activity, specifying instantaneous direction and speed of the arm movement, to yield more accurate estimates with fewer neurons. This approach could greatly benefit from an improved ability to track the switching of plan activity, which precedes movement onset, so that a more accurate plan estimate can be incorporated into movement decoding. In this paper, we propose a modified point-process filter, employing an adaptive parameter, that is capable of more accurately tracking constant plan periods and step changes than conventional methods. We also suggest that this algorithm is more attractive than an alternate maximum likelihood step tracking scheme. Ultimately, the adaptive algorithm is well-suited for use with the PMML algorithm, or for directly controlling prosthetic devices with plan activity, and should improve neural prosthetic system performance.**

Index Terms— **Neural prosthetic systems, estimation algorithms, decode algorithms, adaptive point-process filters, step tracking.**

I. Introduction

RECENT progress on neural prosthetic systems has demonstrated that monkeys [1], [2] and humans [3] can learn to move a computer icon to various target locations simply by activating neural populations that participate in natural arm movements. The discovery that neurons respond selectively to the direction and speed of arm movements, combined with estimation algorithms for translating this neural activity into computer-icon control signals, has resulted in proof-of-concept neural prosthetic systems. While these systems can perform quite well by recording from many tens to hundreds of neurons simultaneously, there is intense interest in increasing performance while simultaneously decreasing the number of neurons required; decreasing the number neurons needed reduces the number of electrodes that must be surgically implanted. By improving system performance and reducing the number of implanted electrodes it may be possible to deliver prosthetic arm (or icon) systems to disabled patients sooner than would otherwise be possible.

The key idea of our recently reported plan-movement maximum likelihood (PMML) algorithm [4] is that plan activity indicates the targeted end-point of the arm movement and can serve as a powerful probabilistic constraint for decoding neural activity present during movements. The performance of the PMML algorithm, as well as the ability to control prosthetic devices directly with just plan activity, depends on the ability to track neural plan activity with high fidelity. In the case of the PMML algorithm, the quality of the movement estimate relies on an accurate estimate of plan activity just before a movement begins. Since a movement "go" signal may potentially arrive at any time [5], it is critical that estimates of plan activity be accurate and track potentially rapid switches as quickly as possible.

Many standard estimation algorithms (e.g., linear filters [6], maximum likelihood [6], and especially point-process filters [7]) estimate constant or slowly varying neural activity quite well, but to our knowledge an estimation algorithm specifically designed to track constant hold periods as well as rapidly changing periods has not been put forth. Here we propose a point-process filter with an adaptive parameter that is typically set at a value (σ_0) well-suited for estimating constant hold periods, but can be briefly switched to a value (σ_1) well-suited for tracking rapidly changing periods. The switch to the alternate σ_1 parameter is governed by a neural-plan-activity edge detector algorithm running in parallel to the estimation filter. The task consists of planning to locations in a 2-dimensional workspace under some assumed plan switching statistics. The result is that the adaptive point-process filter performs better in simulation than the optimal fixed-parameter point-process filter.

This work was supported in part by the NDSEG Fellowship (G.S.), the NSF Center for Neuromorphic Systems Engineering at Caltech, the Burroughs Wellcome Fund Career Award in the Biomedical Sciences, the Center for Integrated Systems at Stanford University and the Sloan Foundation. Please address correspondences to gopals@stanford.edu.

II. Methodology

A. Neural Signal Model

It has been shown that modeling spike trains of individual neurons as inhomogenous Poisson point-processes captures most of the statistical variation of neural data [8]. In our model, the instantaneous rate encodes the parameter of interest, namely target location. Thus, the distribution of the number of spikes (or action potentials), k_j, observed from a cell indexed by j is given as

$$p_j(k_j|\mathbf{x}(t),T) = \frac{[\lambda_j(\mathbf{x}(t))T]^k}{k!}e^{-\lambda_j(\mathbf{x}(t))T} \quad (1)$$

where the position $\mathbf{x}(t)$ and, consequently, the rate $\lambda_j(\mathbf{x}(t))$ are constant over some T seconds preceeding t.

The relationship between firing rate and spatial position in plan neurons has not been extensively investigated, though it has been shown that the tuning varies with direction and extent of movement [5], [9]. For simplicity, we model the tuning as Gaussian, where the firing rate of the neuron decreases radially from a preferred location. The functional form of mean firing rate is given as

$$\lambda_j(\mathbf{x}(t)) = \lambda_{max}\exp\left(-\frac{\|\mathbf{x}(t)-\mathbf{u}_j\|^2}{2\xi^2}\right) \quad (2)$$

where $\lambda_j(\mathbf{x}(t))$ is the instantaneous rate of the Poisson process at the plan location $\mathbf{x}(t)$, λ_{max} specifies the maximum mean firing rate of the neurons, ξ the standard deviation of the tuning, and $\mathbf{u}_j$ the location of maximal firing. The same values of λ_{max} and ξ are taken for every simulated neuron in our population. The $\mathbf{u}_j$ are randomly chosen to lie within the workspace.

B. Statistical Distributions on Plan Sequences

When attempting to characterize plan dynamics there are three major variables: switching speed of a plan, distribution of plan hold

times, and distribution of plan change distances. A first pass might involve quantifying these variables through observations of the human motor system performing daily tasks. However, the plan and movement subsystems need not have the same characteristics. For example, individuals plan movements to many objects even though only a few movements are ultimately executed. Furthermore, plans of limb movements could have much more rapid switching dynamics than actual movements which involve mass and inertia. These questions are still to be investigated experimentally.

In this study, we choose the limiting case of instantaneous switching speeds. This scenario makes the task of decoding most important since it emphasizes the error between the estimate and the intended plan location around the time of a switch. The hold times are assumed to be uniformly distributed between 100ms and 500ms. The plan activity will switch from one location to another with each location kept constant for a randomly drawn hold time. Finally, the incremental distance between the pre-switch and post-switch plan location is uniformly distributed between $0.25\frac{l}{2}$ and $0.75\frac{l}{2}$, where l is the edge length of the workspace. After first drawing the step distance, a direction is picked uniformly. This angle is redrawn if the increment vector takes the new plan location outside of the workspace.

C. Adaptive Point-Process Filter

The point-process (PP) filter as described in [7] is a promising candidate for continuously estimating the planned end-point. The key similarity between the step tracking problem and the estimation task in [7] lies with the neural response being described as a PP whose mean firing rate is a Gaussian function of position. The filter uses a recursive algorithm, similar to the Kalman time and measurement updates, to incorporate the previous sample estimate with spike data from the current time point. The previous estimate is first modified by the time update, with the upcoming movement increment vector stochastically distributed as a 2-dimensional Gaussian centered at the past estimate. The constraint is known here as the "random walk" parameter since the concept was first used to describe the seemingly random movement statistics of a free foraging rat. The measurement update adjusts the estimate by the latest point-process observations. The new estimate is spatially continuous. A variance is calculated with each estimate, thereby allowing the current estimate to be used to form a prior distribution for the next estimate.

Equations (3)–(5) constitute the one-step prediction (or time update) phase of the PP filter. The measurement update equations for $\hat{\mathbf{x}}(t_k|t_k)$ and posterior variance $W(t_k|t_k)$ equations are not included here. These and further details of the filter derivation can be found in [7].

$$\mathbf{x}(t_k) - \mathbf{x}(t_{k-1}) \sim N(0, W_{\mathbf{x}}(\Delta_k)) \tag{3}$$

$$\hat{\mathbf{x}}(t_k|t_{k-1}) = \hat{\mathbf{x}}(t_{k-1}|t_{k-1}) \tag{4}$$

$$W(t_k|t_{k-1}) = W_{\mathbf{x}}(\Delta_k) + W(t_{k-1}|t_{k-1}) \tag{5}$$

Equation (3) describes the prior on $\mathbf{x}(t_k)$ given $\mathbf{x}(t_{k-1})$. In (4), the vector $\hat{\mathbf{x}}(t_p|t_q)$ is the position estimate at time t_p given all the information until the q^{th} time step. Equation (5) relates $W(t_k|t_{k-1})$, the variance in the position after the time update, to $W(t_{k-1}|t_{k-1})$, the variance of the preceeding estimate $\hat{\mathbf{x}}(t_{k-1}|t_{k-1})$. The Gaussian distribution of the "random walk" in (3) is described by its covariance matrix $W_{\mathbf{x}}(\Delta_k)$; this matrix is constant throughout the operation of the filter. If the diagonal elements of this matrix are small, the prior estimate will be very influential when computing the next estimate. Conversely, the prior estimate will have a smaller effect on the next estimate if the diagonal elements are large. This allows the filter to be nimble when the plan position changes; it will place more importance on the latest vector of spikes at the cost of increasing sensitivity to noise present in the spike train.

The optimal choice for $W_{\mathbf{x}}(\Delta_k)$ is dictated by the statistics of the movement. Consider a simplified version of the random walk covariance where $W_{\mathbf{x}}(\Delta_k) = \sigma^2 I$. If the number of steps per second is reduced (or, equivalently, hold times lengthened) the optimal choice of σ would decrease. Similarly, a distribution that favors larger step sizes would prefer a larger value of σ than a sequence that has smaller step distances on average. The optimal value of σ is termed σ_{opt}.

To achieve better performance, we can adapt σ as follows: use a small random walk parameter (σ_0) during hold periods and use a larger parameter (σ_1) to transition between regions of constant plan activity. In this manner, we are able to exploit the benefits of $\sigma_0 < \sigma_{opt}$ without suffering from its corresponding slow switching rate. On the flip side, $\sigma_1 > \sigma_{opt}$ provides a faster switching rate without the penalty of high noise during the constant hold regions.

Therefore, we run two PP filters in parallel and employ an edge detector. By default, the estimator uses the PP filter with parameter σ_0. When the detector finds an edge, the prior estimate and variance of the σ_0 filter is switched to that of the σ_1 filter. This operation need only be performed for a single time step. It has the effect of reseeding the slower slewing filter to a position closer to the actual plan position. Given that the estimate is coming from another filter with higher σ^2, the reseeded position is naturally noisy. This is not a problem — the prior variance is also reseeded, allowing for large corrections until the variance naturally relaxes with the accumulation of enough post-edge data.

D. Maximum Likelihood Expanding Filter

Alternatively, one intuitive technique for tracking step changes in the plan activity would be to employ an expanding maximum likelihood (ML) filter. A traditional ML approach would first consider, for each cell, only the neural firings over a fixed width, sliding window. At each time step, the estimator picks the position that maximizes the joint probability distribution, across the cell population, given the neural data in the window [6]. A well-known fact is that the variance of the ML estimate of the parameter of a Poisson process varies inversely with the length of the estimation window [10]. Thus, a natural improvement to the standard approach is to expand the window during hold periods rather than slide it at each time step. If the time locations of the edges are known perfectly, one could reset the filter's window after each edge to eliminate any bias from data before the edge. It is important to note, however, that the ML estimate will be rather noisy for a short time after the reset operation as there is very little data in the filter after the edge.

We can write the mathematical description of the ML filter as

$$LL(\mathbf{x}(t)) = \log\left[\prod_{j=1}^{M} p_j(k_j|\mathbf{x}(t), T)\right] \tag{6}$$

$$\hat{\mathbf{x}}(t) = \underset{\mathbf{x}(t)}{\operatorname{argmax}}\left[LL(\mathbf{x}(t))\right] \tag{7}$$

where T is the duration of time since the last edge as reported by an edge detector, k_j is the total number of spikes observed in T, and $\hat{\mathbf{x}}(t)$ is the ML estimate of the plan location.

E. Edge Detection

In order to test our point-process step tracking algorithms, we require an edge detector. The goal should be to implement a strategy that detects edges with a short latency and minimizes false negatives[1]. It is also beneficial to reduce false positives since these errors can introduce excessive noise into the system.

The method used for edge detection is a simple threshold detector. The algorithm is characterized by the parameter tuple $(t_{\sigma_0}, t_{gap}, t_{\sigma_1})$. At any instant in time there is $t_{\sigma_0} + t_{gap} + t_{\sigma_1}$ amount of history in the detection filter. When checking for an edge, the algorithm averages the last t_{σ_0} samples from the σ_0 PP filter and averages the first t_{σ_1} samples from the the σ_1 PP filter. An edge is declared if the average from the faster response filter exceeds the average from the slower filter by a threshold. After the edge detection, the adaptive point-process algorithm acts as previously described. The parameters for the edge detector, including the threshold, were fixed after some hand optimization.

F. Simulation Architecture

The simulations were performed on populations of 100 neurons with preferred locations chosen uniformly randomly in the workspace. The maximal firing rate of each cortical neuron is set to 100 spikes per second. The workspace is a 10 by 10 square of arbitrary units (a.u.) and ξ in (2) is chosen so that the area with $\lambda_j(\mathbf{x}) \geq 0.5\lambda_{max}$ covers approximately 40% of the workspace. For our models, there is no apparent closed-form solution for the maximum-likelihood filter. Thus, we discretized the workspace into a grid to simulate the maximum-likelihood expanding filter. The following results are from a uniformly spaced 400 point grid.

Each trial lasts two seconds in which step sequences are generated as per the described assumptions. The error metric is the average Euclidean distance of the estimate from the true plan position over the entire trial. This is appropriate since we assume that the "go" signal can appear at any time within the trial. Finally, we averaged the trial-by-trial error over 500 iterations to guarantee consistent convergence.

III. Results

A. Non-Adaptive Point-Process Filter

We first ran simulations to understand the limitations of the non-adaptive PP filter. Fig. 1A shows two simulation sweeps of the PP filter. One simulation was conducted without any step sequences (i.e., the plan distance was drawn for only a single reach from the origin and it was held constant throughout the two-second trial). The PP filter's initial position was seeded at the origin. Clearly the noise drops with lower σ parameters; without any steps, there is no penalty for slower slew rates. Furthermore, this curve is a lower bound on the average error. The addition of steps can only add error in the vicinity of each switch time. Next, the inclusion of steps in the plan sequence yields the convex curve in Fig. 1. As expected, very low values of σ incur large error due to the inability to slew quickly to new plan locations while higher values of σ suffer from noise during the hold regions. The optimal point based on the plan sequence statistics is denoted as σ^2_{opt} on the plot.

[1]False negatives can be tolerated as long as the misses are on smaller step distances. Although the filter may be slow to recover to the new hold location, the estimate will not have to slew far.

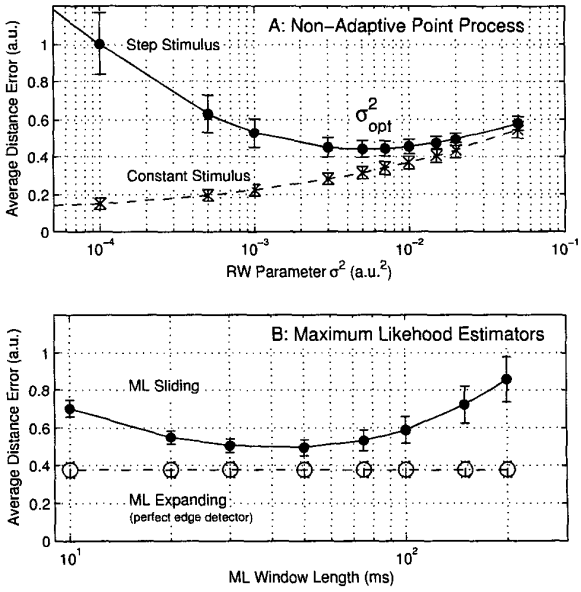

Fig. 1. Comparison of baseline cases. (A) Performance of the PP filter as a function of a single, fixed random walk parameter. (B) Analogous performance curves for ML filters. Note that the filter is not a function of window size; its error is simply repeated across the x-axis for comparison purposes. Error bars indicate standard deviation.

B. Maximum Likelihood Expanding Filter

The traditional ML sliding window approach results are analogous to the non-adaptive PP filter. Fig. 1B presents results from two ML algorithms. The sliding window variant also shows a convex shape. The small integration windows show large error due to their susceptibility to noise. The larger windows perform poorly due to a slow slew rate resulting from the large amount of pre-edge data in many post-edge estimates. The performance of the alternate maximum likelihood expanding filter is also plotted on the same figure. This filter must be reset after each edge and, to simplify the analysis, we did so by using a *perfect* edge detector with 15ms latency. We expect the filter to only do worse with a true edge detection algorithm.

C. Adaptive Point-Process Filter

Lastly, we investigated the adaptive PP filter and the results are shown in Fig. 2. We did not perform an exhaustive search of the high-dimensional space of algorithm and model parameters. However, in the regime we tested, there is a marked improvement of our algorithm over the non-adaptive filter. We chose σ_1 by hand optimization near the point σ_{opt} in Fig. 1. Then, we swept σ_0 and found that the error is lower with smaller values of σ_0 (Fig. 2). The difference in error between very low values of σ_0 is not well-differentiated because, after each edge, the injected variance from filter σ_1 does not decay sufficiently by the time of the next edge. The best value of σ_0 is tightly coupled with the success statistics of the edge detector since the cost of missing an edge will eventaully become more significant to the overall error as σ_0 decreases.

The asymptotic error of 0.37 a.u. at low σ_0 is approximately 16% better than the average error of 0.44 a.u. from the optimal σ_{opt} non-adaptive filter. In a trial-by-trial comparison, 99.2% of all

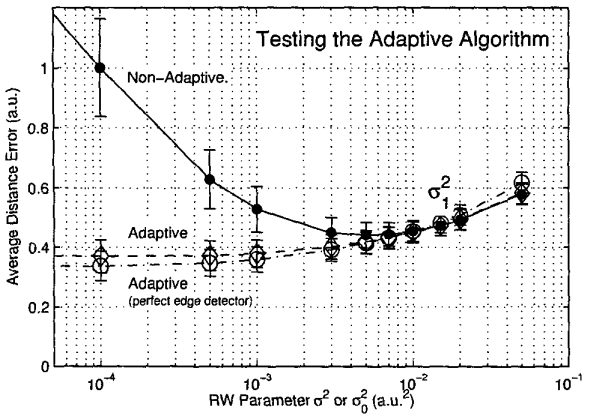

Fig. 2. Response of adaptive PP filter. The highest error curve is repeated from Fig. 1A for comparison. For both adapting filters, σ_1 is fixed to 0.015 a.u.^{2} and σ_0 is swept along the x-axis. The lowest curve corresponds to the adaptive point-process filter with a perfect edge detector that has fixed latency of 15ms. The curve simply labeled "Adaptive" shows simulations that instead use the edge detector algorithm described earlier. The parameters for the edge detection are $(t_{\sigma_0}, t_{gap}, t_{\sigma_1})$ = (50,10,15)ms and the threshold is set at 1.25 a.u.

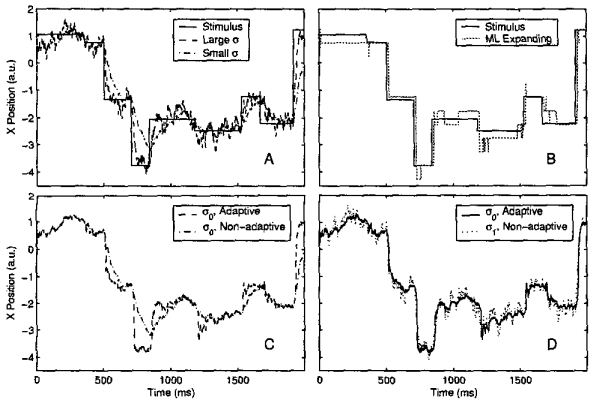

Fig. 3. Single trial responses. (A) Illustration of the trade-off between low noise and fast slew rate across different σ values. (B) The ML expanding filter looks to be performing well but it suffers from snap-to-grid effects. (C) The asymptotically optimal adaptive PP filter performs much better in terms of slew rate than the non-adaptive filter with $\sigma = \sigma_0$. (D) Furthermore, the adaptive filter outperforms a fixed $\sigma = \sigma_1$ counterpart due to less noise in the hold periods.

trials show a performance improvement with the adaptive filter. It is also informative to visually inspect the output estimates of these various filters in a single trial (Fig. 3). These plots conveniently summarize the concepts aforementioned in this paper.

IV. Conclusions

Both the adaptive PP filter and ML expanding filter perform significantly better than their traditional counterparts. These two filters also have similar performance when compared against each other. However, the PP filter is a more attractive alternative for this class of problems. Theoretically, in the limit of an infinitesimal grid spacing, the maximum likelihood expanding filter is an ideal strategy for plan step sequences, but finer grid spacings increase the compute time of the ML filter estimate. In fact, the compute time was seen to be a constraining factor in our study. For example, we would require 1600 grid points with the ML filter to obtain an error equivalent to that of the adaptive PP filter (both using perfect, 15ms latency, edge detection). On a moderately equipped present-day PC, a C++ implementation of this ML filter runs about half as fast as real-time while the equivalent PP filter runs considerably faster than real-time. This performance gap could be improved by rigorously optimizing the algorithm, but ML estimation compute time will degrade (increase linearly) as more neurons are added to the population.

Ultimately, the proposed adaptive PP filter appears to have considerable promise. Its compute time is low enough that it should perform in real-time even as hundreds of neurons are added to the data set. The addition of neurons can only improve the step tracking; the filter will generally be less noisy and the edge detector will be more accurate. Our approach also allows for the prospect of incremental performance improvement. As more sophisticated edge detectors are designed to provide consistent low-latency estimates of switching points, such algorithms can be mated with the adaptive point-process filter to yield more accurate estimates of the plan location. By improving step sequence decoding by approximately 16%, the adaptive point-process algorithm, working in conjunction with PMML or PMML-like algorithms, could substantially improve the performance of neural prosthetic systems.

Acknowledgments

We thank Dr. Maneesh Sahani, Caleb Kemere, Byron Yu, and Prof. Teresa Meng for valuable advice and insights throughout this work.

References

[1] M.D. Serruya, N.G. Hatsopoulos, L. Paninski, M.R. Fellows, and J.P. Donoghue, "Instant neural control of a movement signal," *Nature*, vol. 416, pp. 141–142, March 2002.

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

[3] P.R. Kennedy, R.A.E. Bakay, M.M. Moore, K. Adams, and J. Goldwaithe, "Direct control of a computer from the human central nervous system," *IEEE Trans. on Neural Systems and Rehabilitation Engineering*, vol. 8, pp. 192–202, 2000.

[4] C.T. Kemere, G. Santhanam, B.M. Yu, K.V. Shenoy, and T.H. Meng, "Decoding of plan and peri-movement neural signals in prosthetic systems," *IEEE Workshop on Signal Processing Systems*, pp. 276–283, Oct. 2002.

[5] A.P. Batista and R.A. Andersen, "The parietal reach region codes the next planned movement in a sequential reach task," *Journal of Neurophysiology*, vol. 85, no. 2, pp. 539–544, Feb. 2001.

[6] K. Zhang, I. Ginzburg, B.L. McNaughton, and T.J. Sejnowski, "Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells," *J. Neurophysiology*, vol. 79, no. 2, pp. 1017–1044, Feb. 1998.

[7] E. Brown, L.M. Frank, D. Tang, M.C. Quirk, and M.A. Wilson, "A statistical paradigm for neural spike train decoding applied to position prediction from the ensemble firing patterns of rat hippocampal place cells," *J. Neuroscience*, vol. 18, no. 18, pp. 7411–7425, Sept. 1998.

[8] C. Koch, *Biophysics of Computation*, Oxford University Press, 1999.

[9] J. Messier and J. Kalaska, "Covariation of primate dorsal premotor cell activity with direction and amplitude during a memorized-delay reaching task," *J. Neurophysiology*, vol. 84, no. 1, pp. 152–165, July 2000.

[10] N. Twum-Danso and R. Brockett, "Trajectory estimation from place cell data," *Neural Networks*, vol. 14, no. 6–7, pp. 835–844, July 2001.