30th Annual International IEEE EMBS Conference
Vancouver, British Columbia, Canada, August 20-24, 2008

The Use of a Virtual Integration Environment for the Real-Time Implementation of Neural Decode Algorithms

William Bishop[1], Byron M. Yu[2,5,6], Gopal Santhanam[2], Afsheen Afshar[2,3]
Stephen I. Ryu[2,4], Krishna V. Shenoy[2,5], R. Jacob Vogelstein[1], James Beaty[1], Stuart Harshbarger[1]
[1]The Johns Hopkins University Applied Physics Laboratory, Laurel, MD; and [2]Department of Electrical Engineering, [3]Medical Scientist Training Program, [4]Department of Neurosurgery, and [5]Neurosciences Program, Stanford University, Stanford, California; and [6]Gatsby Computational Neuroscience Unit, University College London, London, United Kingdom

Abstract— **We have developed a virtual integration environment (VIE) for the development of neural prosthetic systems. This paper, the second of two companion articles, describes the use of the VIE as a common platform for the implementation of neural decode algorithms. In this paper, a linear filter decode and a recursive Bayesian algorithm are implemented as separate signal analysis modules of the VIE for the real-time decode of end effector trajectory. The process of implementing each algorithm is described and the real-time behavior as well as computational cost for each algorithm is examined. This is the first report of the real-time implementation of the Mixture of Trajectory Models decode [10]. These real-time algorithms can be easily interfaced with pre-existing modules of the VIE to control simulated and real devices.**

I. INTRODUCTION

Brain computer interfaces (BCI) promise to provide significant rehabilitative value to the severely disabled by using brain derived electrophysiological signals to control artificial devices or re-animate a part of the body. A range of devices, such as robotic manipulators, computer cursors, spelling devices and wheelchairs have been controlled using different invasive and non-invasive recording techniques [1]–[7]. Along with this assortment of recording methods, a variety of algorithms to decode user intent have been implemented and tested.

Despite the basic experimental paradigm being the same in many investigations, it is not always possible to judge the relative value of the signal types and decode strategies employed. Recordings across subjects can be highly variable. Although it is possible to share pre-recorded data to compare algorithms in an off-line manner, there is evidence that the off-line and on-line behavior of neurons differs significantly [5], [8]. Re-implementing previously published neural decode algorithms for real-time comparison against each other is time consuming and can involve a substantial duplication of effort.

This work was supported by the Defense Advanced Research Projects Agency (DARPA) Revolutionizing Prosthetics 2009 (RP2009) project, National Defense Science and Engineering Graduate Fellowships (BMY, GS), National Science Foundation (NSF) Graduate Research Fellowships (BMY, GS), Gatsby Charitable Foundation (BMY), Stanford Bio-X Graduate Fellowship (AA), National Institute of Neurological Disorders and Stroke/Collaborative Research in Computational Neuroscience Grant 5-R01-NS054283-02, Medical Scientist Training Program, Christopher and Dana Reeve Paralysis Foundation (SIR,KVS), Burroughs Wellcome Fund Career Award in the Biomedical Sciences, Stanford Center for Integrated Systems, NSF Center for Neuromorphic Systems Engineering at Caltech, Office of Naval Research, Sloan Foundation, and Whitaker Foundation.
Please address correspondence to william.bishop@jhuapl.edu.

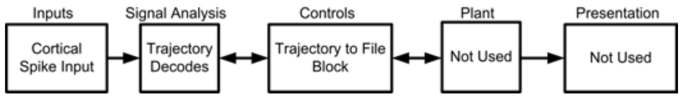

Fig. 1. A diagram of the configuration of the VIE used in the testing of all three algorithms.

We have developed a Virtual Integration Environment (VIE), described in a companion paper, as a common framework for the development of neural prosthetic systems [9]. The VIE modularizes the key components of a neural prosthetic system — inputs, signal analysis, controls, plant and presentation. Complete neural prosthetic systems can be constructed in the VIE and automatically compiled to real-time code. As the VIE implements common interfaces between components, decode algorithms implemented in the signal analysis module of the VIE can be used interchangeably. This makes the sharing of real-time decode algorithms in a "plug-and-play" manner between institutions much more feasible.

In this paper we demonstrate the implementation of a linear filter as well as a recursive Bayesian decode of end effector trajectory in signal analysis modules of the VIE. We examine their real-time behavior and computational cost using pre-recorded data, and we show how they could be used without modification for on-line experiments.

II. METHODS

A linear filter and a recursive Bayesian decode were evaluated in this study. Each was implemented in a separate signal analysis module in the VIE. The VIE implements standard interfaces, and these modules can be interchanged. Fig. 1 displays the configuration of the VIE used for the testing of the real-time decodes. The controls block recorded the output of the decodes for later analysis, and the plant and presentation blocks were unused.

All algorithms estimated arm position in 10 ms intervals using binned spike counts as a feature vector. Bin size was selected individually for each algorithm. As the purpose of the present paper is to describe the real-time implementation of these algorithms in a common framework and not the direct comparison of these algorithms, no formal attempt was made to optimize bin size or binning interval for each algorithm.

Along with the implementation of each algorithm for real-time use, an off-line version of each model was implemented in MATLAB to allow for the direct comparison of off-line and on-line results. The code used to generate the results presented in the original Mixture of Trajectory Models (MTM) paper [10] was used again in the present study, allowing for the direct comparison of the present implementation with previously published work.

These algorithms have all been described elsewhere, and only a brief description, necessary to understand their real-time implementation in the VIE, will be given here.

A. Implementation of Linear Filter Decode in the VIE

The linear filter decode we implemented assumes a linear relationship between the firing rates of a population of units relative to their means and end effector position [11]. This can be represented in the following equation:

$$\hat{\vec{x}} = A(\vec{f_r} - \bar{\vec{f_r}}) + \vec{x_c} \tag{1}$$

In equation 1, $\vec{f_r}$ is a vector of observed firing rates, $\bar{\vec{f_r}}$ is a vector of mean firing rates and $\vec{x_c}$ is a constant offset. Matrix A gives the least squares estimate of end effector position, $\hat{\vec{x}}$, taking into account $\vec{x_c}$, given $\vec{f_r} - \bar{\vec{f_r}}$.

The VIE implementation of the linear filter can be seen in Fig. 2. In our implementation, the binning of spikes is synchronized to an external signal pulse, which allows the decode to be exactly synchronized to external hardware such as a graphics card or motion capture technology, which may run close to but not exactly at a fixed rate. The parameters needed in the real-time decode are currently provided at compile time and remain fixed. However, the model could be easily configured so that most of these were tunable during run time. With the exception of the spike counting block, which made use of an embedded Matlab function block, the entire module was created by simply linking together native Simulink blocks.

B. Implementation of Recursive Bayesian Decode

The MTM algorithm is a recursive Bayesian decode which assumes a subject is reaching to one of M targets [10]. It is capable of incorporating delay neural activity with peri-movement activity to estimate end effector trajectories. The MTM algorithm is made up of two components. First, for each target, a linear Gaussian state model is formed that describes the probability of the arm being in state $\vec{x}_{t+1,m}$ given $\vec{x}_{t,m}$, where *t* is a time index and *m* is an index for each state model. The second component is a single observation model, which describes the probability of a neural observation, $\vec{y}_t$, given $\vec{x}_t$. These two models are combined in a recursive manner for each time step to calculate the state posteriors for each model (referred to as the conditional state posteriors). Finally, the means of the state posteriors are combined in a weighted average to form a single state estimate.

In a real-time system, available calculation time is fixed. The causal, off-line implementation of the MTM decode uses Newton's method to find the peak of each state posterior. Newton's method will not converge for all starting conditions, and time constraints in a real-time system may not permit for multiple attempts. The one-step prediction for each model gives the optimal estimate for the conditional state posterior when new neural observations cannot be incorporated. In our implementation, when a single attempt at Newton's method fails to converge after 10 iterations, the one-step prediction for each model is used as the conditional state posterior.

As noted by Yu et al., the estimation of each conditional state posterior can be carried out in parallel [10]. While our real-time system did not support parallel processing, we exploited this property to easily serialize the calculation of each conditional state posterior into 8 sequential run-time steps. While this introduced a fixed delay of 8 ms into the output of the decode, the increased time for the computation of each conditional state posterior was necessary to avoid overloading the CPU of the real-time PC.

The development of the MTM signal analysis module was greatly facilitated by the use of embedded Matlab functions, which allowed for the reuse of much of the existing MTM code base. These embedded Matlab functions accounted for the bulk of the code in the signal analysis module. A C-coded S-Function was used to apply lags to each channel in the peri-movement spike counting. Native Simulink blocks were used for signal routing and the final conditioning of the decoded output for interface with controls. The details of the real-time MTM implementation are shown in figure 3.

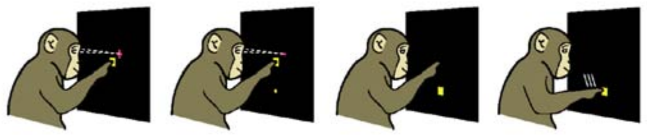

Fig. 4. Delayed center out reach task. A: The task began with a central hold period during which the primate touched a central target and maintained eye fixation on a cross hair. The target (one of eight) was then presented, but the primate was not yet permitted to reach for the target. After a pseudo-randomly chosen delay period, the central target and cross hair disappeared, central fixation was no longer enforced, and the primate reached for the target. Upon reaching and maintaining contact with the target for a fixed period of time, the animal was rewarded.

C. Datasets, Training and Testing

Datasets used in this study were previously reported in Yu et al. [10]. The dataset was from a Rhesus Macaque (Macaca mulatta) performing a delayed center out task, as illustrated in Fig. 4. Recordings from 98 single and multi-units were obtained from the right hemisphere dorsal premotor (PMd) and motor (M1) cortex. Further description of the dataset (monkey G in the original paper) can be found in [10].

The dataset contains 1456 trials, but only 1440 trials were used to keep the number of trials to each target balanced (180 trials to each target). For each target, 18 trials were randomly

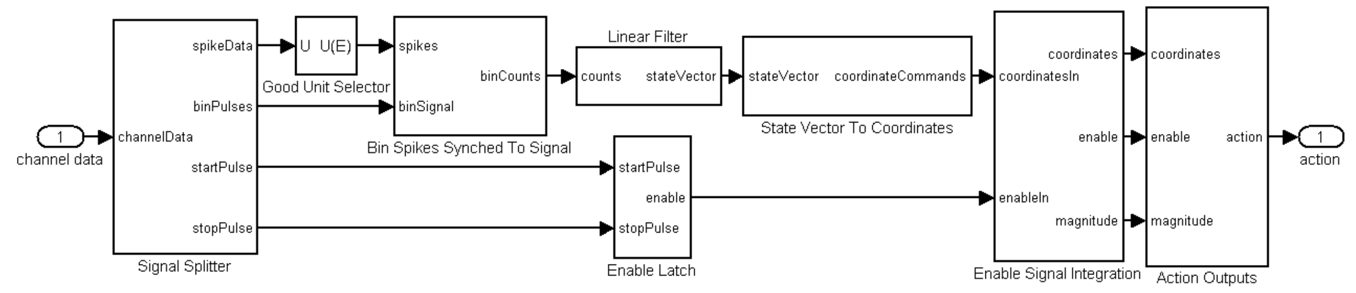

Fig. 2. The core of the linear filter module as it was implemented in a signal analysis module in the VIE. Both spike data and channels carrying synchronization as well as start and stop commands enter the module on the channel data signal bus. Each unit is given its own channel, and spikes are represented as impulses on this channel. All channels carrying spike data are selected from the bus and fed into the good unit selector, which passes only the channels desired for use in the decode to a spike counting block. The spike counting block, Bin Spikes Synched to Signal, implements a circular buffer to record the occurrence of spikes in the last t_b seconds, where t_b is the bin size. The number of spikes for each channel in this buffer is counted every time an external synchronization pulse is received on the binPulses line. The output of the spike counting block is fed to the Linear Filter block which estimates end effector position according to equation 1. The output of the decoded trajectory is passed through a linear transformation in the State Vector To Coordinates block so that the decoded trajectory can be scaled and shifted as desired. Along with pulses to synchronize the decode, start and stop pulses are among the incoming channels. These are fed to the Enable Latch block to produce an enable signal. The enable signal and the scaled and shifted estimate of instantaneous end effector position are fed to the Enable Signal Integration block, which passes through the estimate of end effector position if the enable signal is high. If the enable signal is low, the desired resting position of the end effector is passed. Finally, the output of the Enable Signal Integration block is fed to the Action Outputs block which formats the decoded output appropriately to match the interface to the controls block.

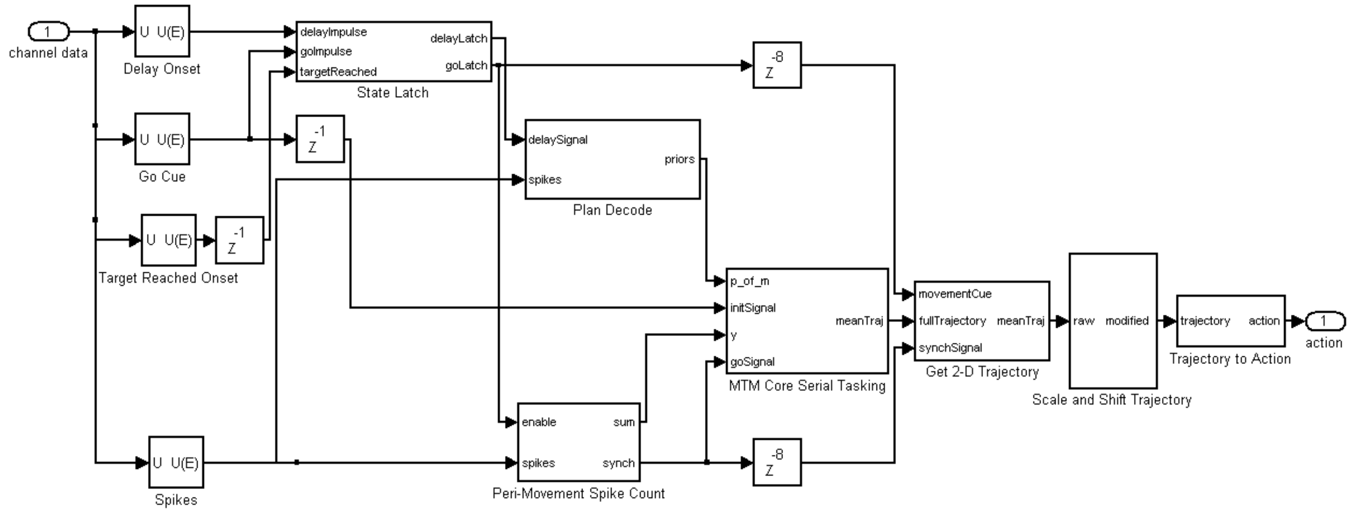

Fig. 3. The core of the mixture of trajectory models algorithm as it was implemented in a signal analysis module in the VIE. The channel data bus contains both spike channels as well as channels for the synchronization of the decode. The synchronization channels (for delay onset, start of peri-movement decode and finish of peri-movement decode) are fed to the State Latch module which produces a signal that is high during the delay period and a second signal that is high during the time of arm movement. The neural data is fed to the plan portion of the decode and the peri-movement portion of the decode. The plan portion of the decode is implemented in the Plan Decode block. This block bins spikes at a given offset from the onset of the delay enable signal and estimates the prior probability of moving to each target as described in the original MTM paper [10]. The neural activity is also fed to the Peri-Movement Spike Count block, which applies fixed time delays (determined as the optimal lags for each unit during training) to each channel and counts the spikes in successive bins. These spike counts, along with the prior probability of reaching to each goal, are fed to the MTM Core Serial Tasking block. This block performs a one step prediction for the state vector of each model and then uses the results of the one step prediction and the latest peri-movement spike count to estimate the mean and covariance matrices of the conditional state posterior probability density functions for each model. The final step in the MTM Core Serial Tasking block is a weighted average of the conditional state posterior mean vectors to form an estimate of the state vector. The state vector contains terms for the components of position, speed and acceleration as well as position and speed magnitudes. The Get 2-D Trajectory block selects only the position components and passes those to the Scale and Shift Trajectory block. After the desired scale and shift has been applied to the decoded trajectory, it is formatted by the Trajectory to Action block to conform to the controls block interface.

selected to form a test set, and the remaining trials were used for training.

Using a custom Matlab function, the datasets were converted from Stanford's unique format to the Common Neural Raw Format (CNRF) [9]. This permitted the development of off-line training functions which were not tied to a lab specific data format but could be reused in the future for any center out task data stored in CNRF. These training functions were written as Matlab functions, and they saved all relevant trained parameters to structures that were used by the corresponding VIE signal analysis modules during testing. For testing in the VIE, the CNRF data was converted to Common Neural Simulink Format (CNSF), a format designed for streaming in simulation. All testing was carried out on a PC with a 3.6 Ghz Pentium 4 processor running xPC Target real-time kernel.

III. RESULTS

The implemented modules were placed in complete VIE models configured as described in section II and real-time code was generated. The on-line trajectories produced by the VIE decode were recorded and compared to those produced off-line in Matlab. The off-line and on-line decoded trajectories for both algorithms (Fig. 5) matched exactly[1]. While no additional constraints were placed on the VIE implementation of the linear filter, the real-time implementation of the MTM algorithm limited the number of iterations of Newton's method used in the estimation of the conditional state posteriors. The perfect correspondence of the two decodes thus implies that the real-time implementation of Newton's method always reached convergence — despite the additional constraints. Further analysis confirms this; convergence was reached on average with 2.9 (+/- .4) iterations of Newton's method and at no point were more than 5 iterations necessary to reach convergence. This demonstrates that despite the significant complexity of the MTM algorithm, it is possible to run this algorithm in a real-time system such as the VIE.

Along with the real-time behavior of the decodes, the computational expense associated with each decode was quantified. To estimate the processing power requirements of each decode, the mean and max percent use of available execution time (ET) were calculated[2]. In considering a real-time system, both measures are important. The max percent use of available ET indicates how much processing power is available for other processes, irrespective of how they are synchronized with the neural decode. The mean is an indicator of the amount of ET available to other processes that are synchronized with the neural decode in a manner that prevents the parallel processing of computationally intense calculations. By both measures, the MTM decode was an order of magnitude more computationally intensive than the linear filter (Fig. 6). For both algorithms, the max percent use of available ET was significantly higher than the mean. The spread between the mean and max for both algorithms results from large periods of relatively low activity (simply binning spikes or waiting between the delay decode and peri-movement decode for the MTM algorithm) interspersed with computationally intensive periods where estimates of end effector position are actually performed.

To assess the relative impact of Newton's method on the real-time computational burden of the VIE MTM algorithm,

[1]A fixed delay was accounted for in the comparison of the results of the real-time MTM decode with the off-line results

[2]The percent use of available execution time (ET) at the ith time step, p_i, was calculated as: $p_i = \frac{d_i - r_i}{a_i - r_i}$. The time required to compute all necessary calculations for the VIE as a whole at each time step is referred to as the Time of Execution (TET). Here, d_i is the TET for the VIE when the signal analysis module is running the implemented algorithm. The variable r_i is the TET for VIE configuration that uses an empty signal analysis module but is otherwise identical to the configuration of the VIE which employed the algorithm. Finally, a_i is the fundamental step size at which the VIE runs (1 ms), which can be considered the total time available to perform all necessary VIE calculations.

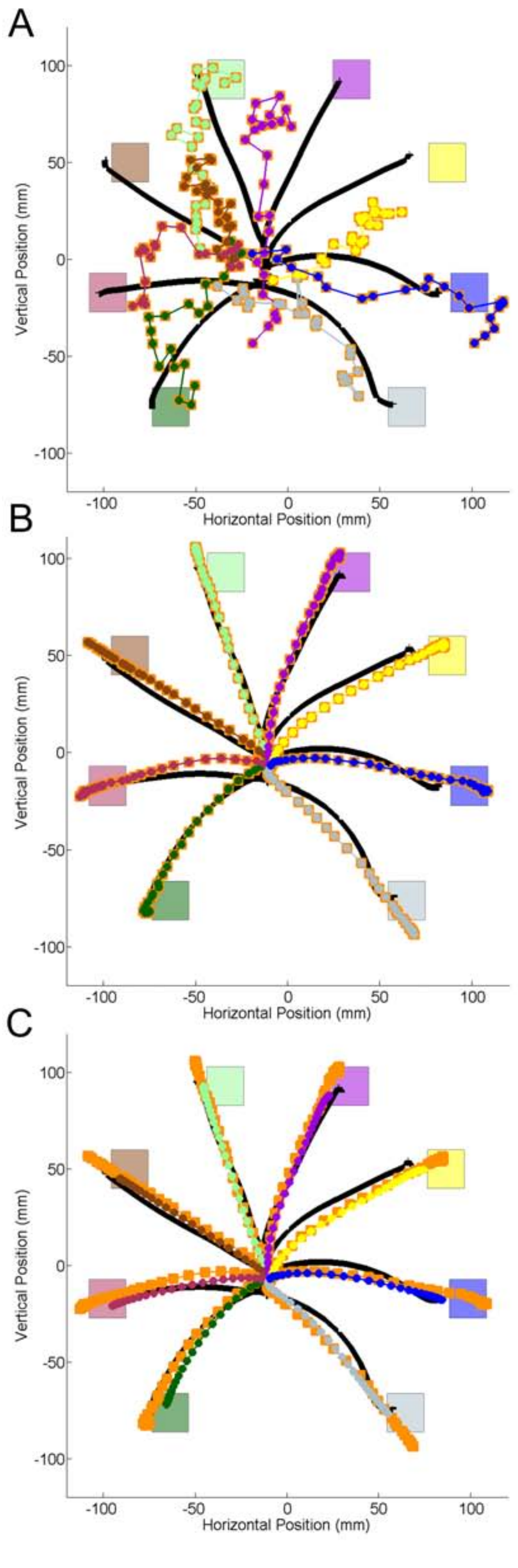

Fig. 5. A representative trial to each target for the linear filter (A) and MTM (B) decodes. The actual trajectory (black) is plotted to each target. The results of the off-line decode performed using Matlab code are shown (orange boxes for all targets). The results of the on-line decode are plotted as dots, connected with thin lines and color coded to the target presented during the trial. Panel C shows the results of the MTM decode when the use of Newton's method is constrained to a max of 2 iterations.

a linear fit between the number of iterations of Newton's method and the percent use of available ET was performed. The calculated R-squared value for the fit was .98. No other processes in the real-time implementation of the VIE are believed to be correlated with the number of iterations in Newton's method, and this is a strong indicator that the number of iterations in Newton's method is the primary driver of max percent use of available ET.

To simulate the behavior of the current VIE implementation of the MTM decode in more resource constrained systems, the maximum number of iterations for Newton's method was set arbitrarily low to 2. This prevented Newton's method from converging at many time steps, and hence, the results of the one-step prediction were used in place of the conditional state posteriors for these time steps. These trajectories were directed toward the correct target but were shortened in comparison to the trajectories decoded off-line (Fig. 5).

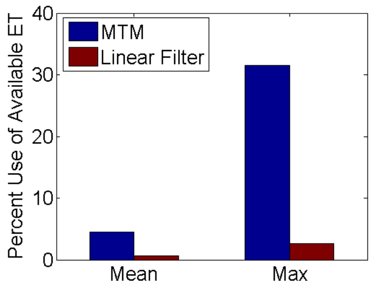

Fig. 6. The mean and max percent use of available execution (ET) time for the MTM and linear filter algorithms. The mean available ET for each measure was nearly identical (999.2 $\pm.9$ μs and 999 ±1 μs the for MTM and the linear filter algorithms, respectively).

IV. DISCUSSION

In this paper, the implementation of a linear filter and recursive Bayesian decode in a signal analysis module of a Virtual Integration Environment (VIE) have been presented. The real-time estimated trajectories produced by these two algorithms matched exactly those produced with off-line Matlab code. While linear filters similar to that presented here have been implemented for real-time use before [12], [13], this is the first real-time implementation of the MTM decode— a sophisticated decode algorithm with computational requirements an order of magnitude greater than those of the linear filter.

A key step in the MTM algorithm is the estimation of the conditional state posteriors, which involves the computationally intense application of Newton's method. The convergence of Newton's method for the particular dataset used in this study by no means guarantees that the real-time decode will converge in 10 iterations or less for all data sets. Additionally, if the MTM decode is to be implemented in a more resource constrained system, the probability of reaching convergence for each conditional state posterior at all time steps will be significantly less. While more study of alternative methods of finding the peak of the state posteriors is needed, this paper presents evidence that using the results of the one-step prediction may be an acceptable method of progressing through time steps when the peak of a conditional state posterior can not be found.

The neural decode algorithms presented here can be easily incorporated into a complete closed-loop experimental system. The input and output of each block conforms to standard interfaces. The input module used in the current work to read from file in real-time could be switched out for one which receives data from a data acquisition system such as Cerebus or Plexon. Similarly, the output of each algorithm could be fed to an appropriate controls block to actuate a desired end effector— whether it be a computer cursor or robotic arm. These decode algorithms could thus be used in a "plug-and-play" fashion in a variety of experimental configurations.

Decode algorithms with computational demands greater than that of the MTM algorithm, which at max only loaded the real-time PC to 30%, can be implemented for real-time use in the VIE. Pertaining to this work directly, in the future, a state decode to autonomously estimate when a subject is planning and making a movement could be implemented in parallel with the MTM algorithm and the real-time performance of this complete system profiled [14], [15].

The use of the VIE among research institutions would increase the feasibility of sharing previously published work with little duplication of effort. An institution using the VIE which desires to use a signal analysis module developed at another institution has only to acquire it in order to begin using it in real-time decodes. Code for off-line training, in preparation for on-line decoding, can also be shared among institutions if the off-line training functions are written to use CNRF data. The direct, real-time, closed-loop comparison of algorithms against each other can be accomplished more readily as the overhead of implementing algorithms for an institution's specific system is removed by the availability of previously verified and validated signal analysis modules that can be used directly in the VIE.

As the field of neural prosthetics continues to grow, the need to directly compare decode algorithms against one another in a real-time, closed-loop manner will only increase. The implementation of algorithms in the VIE provides an efficient path for their real-time implementation and a means of sharing that work with others. Additionally, the modular nature of the VIE, providing standard interfaces to methods of acquiring signals and realizing intent, allows it to serve it a general framework that can be used in a variety of settings. We expect that these properties will render the VIE an enabling piece of technology in the field of systems neuroscience.

REFERENCES

[1] G. Schalk, J. Kubanek, K. J. Miller, N. R. Anderson, E. C. Leuthardt, J. G. Ojemann, D. Limbrick, D. Moran, L. A. Gerhardt, and J. R. Wolpaw, "Decoding two-dimensional movement trajectories using electrocorticographic signals in humans," *Journal of Neural Engineering*, vol. 4, no. 3, pp. 264–275, Sep 2007.

[2] G. Pfurtscheller, C. Neuper, G. R. Muller, B. Obermaier, G. Krausz, A. Schlogl, R. Scherer, B. Graimann, C. Keinrath, D. Skliris, M. Wortz, G. Supp, and C. Schrank, "Graz-BCI: state of the art and clinical applications," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 11, no. 2, pp. 1–4, 2003.

[3] R. Scherer, G. R. Muller, C. Neuper, B. Graimann, and G. Pfurtscheller, "An asynchronously controlled eeg-based virtual keyboard: improvement of the spelling rate," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 979–984, 2004.

[4] K. Tanaka, K. Matsunaga, and H. O. Wang, "Electroencephalogram-based control of an electric wheelchair," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 762–766, 2005.

[5] D. M. Taylor, S. I. H. Tillery, and A. B. Schwartz, "Direct cortical control of 3D neuroprosthetic devices," *Science*, vol. 296, no. 5574, pp. 1829–1832, June 7 2002.

[6] 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*, vol. 442, no. 7099, pp. 164–171, Jul 13 2006.

[7] G. Santhanam, S. I. Ryu, B. M. Yu, A. Afshar, and K. V. Shenoy, "A high-performance brain-computer interface," *Nature*, vol. 442, no. 7099, pp. 195–198, Jul 13 2006.

[8] M. A. Lebedev, J. M. Carmena, J. E. O'Doherty, M. Zacksenhouse, C. S. Henriquez, J. C. Principe, and M. A. Nicolelis, "Cortical ensemble adaptation to represent velocity of an artificial actuator controlled by a brain-machine interface," *The Journal of neuroscience*, vol. 25, no. 19, pp. 4681–4693, May 11 2005.

[9] W. Bishop, R. Armiger, J. Burck, M. Bridges, M. Hauschild, K. Englehart, E. Scheme, R.J. Vogelstein, J. Beaty, S. Harshbarger, "A real-time virtual integration environment for the design and development of neural prosthetic systems," *Proceedings of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vancouver, BC, Canada, 2008.

[10] B. M. Yu, C. Kemere, G. Santhanam, A. Afshar, S. I. Ryu, T. H. Meng, M. Sahani, and K. V. Shenoy, "Mixture of trajectory models for neural decoding of goal-directed movements," *Journal of Neurophysiology*, vol. 97, no. 5, pp. 3763–3780, May 2007.

[11] M. Serruya, N. Hatsopoulos, M. Fellows, L. Paninski, and J. Donoghue, "Robustness of neuroprosthetic decoding algorithms," *Biological Cybernetics*, vol. 88, no. 3, pp. 219–228, Mar 2003.

[12] 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*, vol. 1, no. 2, p. E42, Nov 2003.

[13] 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, no. 6877, pp. 141–142, Mar 14 2002.

[14] N. Achtman, A. Afshar, G. Santhanam, B. M. Yu, S. I. Ryu, and K. V. Shenoy, "Free-paced high-performance brain-computer interfaces," *Journal of Neural Engineering*, vol. 4, no. 3, pp. 336–347, Sep 2007.

[15] C. Kemere, B. M. Yu, G. Santhanam, S. I. Ryu, A. Afshar, T. H. Meng, and K. V. Shenoy, "Hidden markov models for spatial and temporal estimation for prosthetic control." Abstract Viewer / Itinerary Planner. Atlanta, GA: Society for Neuroscience, 2006.

Acknowledgment

We thank Cindy Chestek and Vikash Gilja for their assistance in the translation of the MTM algorithms to the VIE signal analysis module and their review of this paper.