PROGRAMMABLE NEURAL PROCESSING FRAMEWORK FOR IMPLANTABLE WIRELESS BRAIN-COMPUTER INTERFACES

by

Shimeng Huang

B.S., Tsinghua University, China, 2008

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

M.S. in Electrical Engineering

University of Pittsburgh

2010

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Shimeng Huang

It was defended on

April 6th, 2010

and approved by

Allen C. Cheng, Assistant Professor, Electrical and Computer Engineering Department

Zhi-Hong Mao, Assistant Professor, Electrical and Computer Engineering Department

Mingui Sun, Professor, Neurological Surgery

Thesis Advisor: Allen C. Cheng, Assistant Professor, Electrical and Computer Engineering Department

PROGRAMMABLE NEURAL PROCESSING FRAMEWORK FOR IMPLANTABLE WIRELESS BRAIN-COMPUTER INTERFACES

Shimeng Huang, M.S.

University of Pittsburgh, 2010

Brain-computer interfaces (BCIs) are able to translate cerebral cortex neural activity into control signals for computer cursors or prosthetic limbs. Such neural prosthetics offer tremendous potential for improving the quality of life for disabled individuals. Despite the success of laboratory-based neural prosthetic systems, there is a long way to go before it makes a clinically viable device. The major obstacles include lack of portability due to large physical footprint and performance-power inefficiency of current BCI platforms. Thus, there are growing interests in integrating more BCI's components into a tiny implantable unit, which can minimize the surgical risk and maximize the usability. To date, real-time neural prosthetic systems in laboratory require a wired connection penetrating the skull to a bulky external power/processing unit. For the wireless implantable BCI devices, only the data acquisition and spike detection stages are fully integrated. The rest digital post-processing can only be performed on one chosen channel via custom ASICs, whose lack of flexibility and long development cycle are likely to slow down the ongoing clinical research.

This thesis proposes and tests the feasibility of performing on-chip, real-time spike sorting/neural decoding on a programmable wireless sensor network (WSN) node, which is chosen as a compact, and low-power platform representative of a future implantable chip. The final accuracy is comparable to state-of-the-art open-loop neural processing. A detailed power/performance trade-off analysis is presented. Our experimental results show that: 1)direct on-chip neural decoding without spike sorting can achieve 30Hz updating rate, with power density lower than 62mW/cm2; 2)the execution time and power density meet the requirements to perform real-time spike sorting on a single neural channel. For the option of having spike sorting in order to keep all neural information, we propose a new neural processing workflow that incorporates a light-weight neuron selection method to the training process to reduce the number of channels required for processing. Experimental results show that the proposed method not only narrows the gap between the system requirement and current hardware technology, but also increase the accuracy of the neural decoder by 3%-22%, due to elimination of noisy channels.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

First, I would like to express my sincere appreciation to my committee, my advisor Professor Allen Cheng for introduce me to this amazing field of BCI and neural processing, as well as his support and guidance, Professor Zhihong Mao for his trust and giving me the opportunity to continue on my dream in research, and Professor Mingui Sun for his generous advice on improving this work.

Second, I would also like to extend my special thanks to my colleagues and friends Yuwen Sun and Joe Oresko for their collaborating.

Finally, I specially want to thank my close friend and now also my beloved husband, Junqing Wei, for being so inspiring and supportive.

1.0 INTRODUCTION

1.1 BACKGROUND

Brain-computer interfaces (BCIs) are a communication link between the brain and a computer. In simple terms, BCIs can interpret intentions and convert them to commands, allowing a computer to be controlled through thoughts. In literature, they are also termed neuroprosthetics (a.k.a neural prosthetics), brain machine interface (BMI), or neurorobotics. They allow brain controlled prosthetics, communication aids for paralyzed individuals, as well as a variety of other medical devices that could assist patients suffering from epilepsy, seizures, and neuromuscular diseases – all of which would greatly improve the quality of life for disabled individuals.

BCIs work by gathering and processing brain activity signals, most commonly and efficiently measured by the brains' electrophysiological state. The major types of signals include EEG, which is recorded from the scalp; electrocorticography (ECoG), electrical brain activity recorded beneath the cranium; field potentials, brain activity monitored by electrodes from within the parenchyma, and "single units", individual neuron action potential firing mornitored by microelectrodes (Leuthardt et al., 2006). Except for EEG, which is acquired through non-invasive sensor, acquisitions of the other three all utilize invasive electrodes. The deeper the electrodes go, the higher the signal's resolution is, yet the higher surgical risk would have to be taken. Despite its highest acquisition risk among these signals, the action potentials, or "spikes", are sufficiently reliable and believed to have the potential supporting the highest information transfer rate. Thus, this work focuses on neuron action potentials based BCIs.

To interpret intention from neuron action potentials, BCIs should have three basic processing stages: spike detection, spike sorting (optional) and neural decoding. This is summarized in Figure 1. In the spike detection stage, spikes are picked out of raw signal gathered from each electrode (channel), usually through thresholding. The output includes time stamp and the segmented spike waveform of every single spike. The waveforms are required by spike sorting, the technique by which BCIs differentiate which spikes are fired by each of the different neurons that are accessible to a certain channel. This is done separately on every channel. Activities of neurons identified from all channels would then be gathered to form a spike (firing) times sequence. The sequence is passed on to the neural decoding stage, which estimates the intention, such as a desired movement.

It is necessary to point out that the spike sorting stage is neither required nor universally used (Linderman et al., 2007). The described system can be greatly simplified assuming each electrode only observe one neuron. Systems with spike sorting are preferred because there's no information loss, thus in theory could support higher accuracy. In practical, however, there is no guarantee to the amount of improvement.

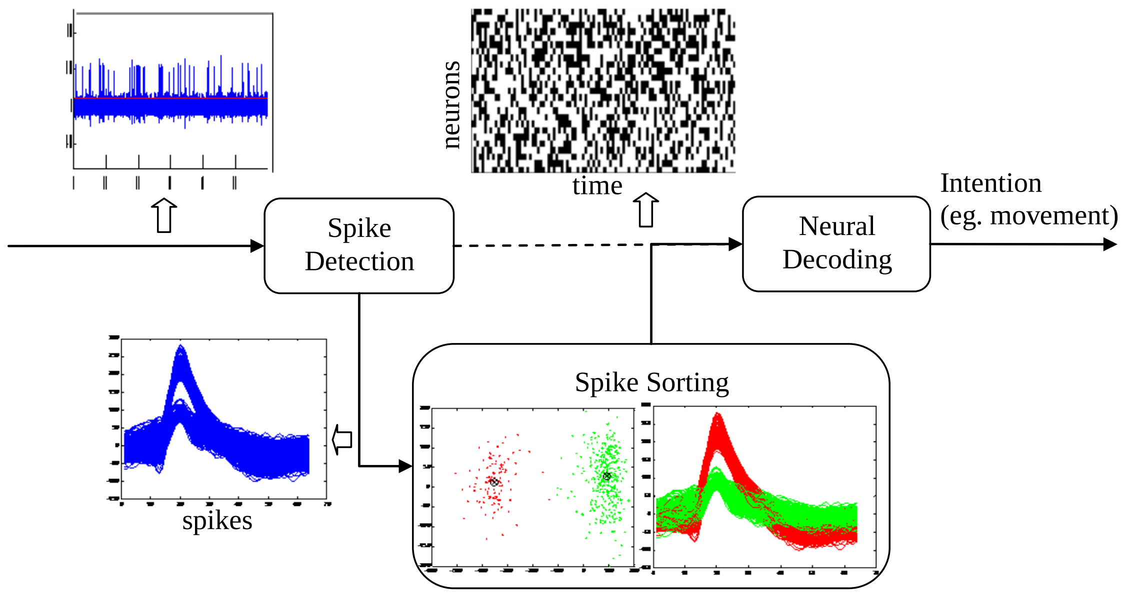


Figure 1 Concept sketch of BCIs' neural processing

1.2 MOTIVATION AND PROBLEM STATEMENT

In the past two decades, great achievements have been made in laboratory neural prosthetics. Several research groups have demonstrated that monkey and human objects are able to control computer cursors or robotic limbs in real-time simply through their thoughts(Hochberg et al., 2006); (Taylor et al., 2002). However, it is still a long-term goal to make clinical utilization of this technology. There are two major obstacles: The first is the surgical risk of data acquisition, which is explained in 1.1; the second is that laboratory BCI devices cannot meet the requirement of daily use in real life. While the first problem is outside the bound of this research, the thesis targets at the second obstacle.

Portability is the most desired property of BCI platforms. Unfortunately, current laboratory real-time neural prosthetics are in general PC-based and rely on wired data transmission. As a result, most objects in real-time experiments have to be immobile. Free-move object test are most wanted for further neuroprosthetic research and demonstration (Chestek et al., 2009). Besides, portability is even more important to BCIs' future users. In conclusion, it is necessary to replace PCs by mobile platforms and to use wireless transmission instead, which also helps reduce the surgical risk.

While current portable devices are powerful enough to handle the neural processing, the wireless transmission has become a bottleneck. Though the required bandwidth is much lower than the capability of state-of-the-art wireless links, (Harrison et al., 2009) pointed out several constraints limit the bandwidth of implantable wireless devices. First, to avoid heating the surrounding tissue, extremely low power dissipation greatly shrink the bandwidth; second, small size requirements prevent the use of efficient antennas; third, increased tissue absorption at high frequencies greatly favors telemetry operation below 1 GHz. Therefore, it is generally believed that some form of bandwidth reduction in data acquisition end is essential for real-time wireless BCIs.

To reduce the bandwidth, previous works on wireless BCI mostly turn to lossy compression: spike sorting is typically skipped so only firing times are sent. An alternative that avoids loss of information is to process it before transmit it, based on the nature of data reduction (by a factor of ~10^6 (Linderman et al., 2007)) along the processing path. This leads to the active research field of implantable neural processing. Indeed, processing would also increase power consumption and size of implantable devices. Yet these would yield with Moore's law, making it more promising in the long run, compared to wireless transmission, whose frequency would be continuously constrained by tissue absorption.

Due to such tight constraints, existing implantable BCI platform are majorly ASIC based. Even though extremely efficient, these hardware solutions lack of flexibility, which is desired by an actively evolving application like neuroprosthetics. To their best efforts, state-of-the-art ASIC designs (Harrison et al., 2009); (Sodagar et al., 2007) are able to support changing of thresholds or turning on/off certain channels. However, demands like modifying algorithms or processing workflow, which are common in experimental stage, would result in a time and money consuming procedure of design, fabricate, and test.

Therefore, this research seeks answers for the questions: 1) how can implantable wireless neural processing platform get more flexibility? 2) how can the resulting system meet real-time requirement and implantable constraints?

1.3 PERFORMANCE REQUIREMENT OF IMPLANTABLE COMPUTING

To answer these questions, it is necessary to address the system requirements and constraints that should be achieved.

A. Constraints

To reduce the surgical risk, at least two factors should be considered: size and heat. Other factors like material of package would also influence the risk, yet are out of the boundary of this work. As to heat tolerance, it is the power density but not the total power that drives the localized temperature distribution. Prior work suggests the upper bound of the power density of a chronic heat source in an *in vivo* setting is estimated to be 62 mW/cm^2 (Reichert, 2007); irreversible tissue damage can occur when this threshold is exceeded.

However, to the best of our knowledge, no literature answers the question of what size is sufficiently small for a safe implant. Current wireless devices that have been implanted into objects' scalps range from 5.1cm×3.8cm×3.8cm (Chestek et al., 2009) to 5.5 cm×5 cm×3 cm (Mavoori et al., 2005), still far too large for a clinical acceptable device.

B. Real-time requirement

Major laboratory neural decoding methods require an update period of 20-100ms to get a smooth result, meaning that neuron activity information should be collected and passed to decoder every 20-100ms.

The next question is how long it should take to process one spike. For motor neurons, a typical spike lasts no longer than 1ms. Recordings also show that almost all spikes can be separated by a 2ms segment window, which is used by most spike detectors. Thus for the worst case of spikes fire continuously, the spike sorting process need to be finished within 2ms. But this overestimates the requirement because continuous firing is hardly observed in motor cortex neurons. From long-term recording results, the maximum number of threshold crossings (the total number of spikes detected from each neuron being recorded by the channel) is 50 crossings per second (Zumsteg et al., 2005). This would indicate a time slot of 20ms/spike, 10 times larger than the worst case. However, this average-case requirement may be loose, especially when considering a decoder updating period lower than 40ms, in which condition each channel can report at most 1 firing. So another statistics is performed on our dataset. Result shows that, for one channel, firing times within 50ms time window never exceed 4, giving a tighter bound of 12.5ms.

C. Processing vs. transmission

The last question to discuss in this section is that what level of data processing is needed before transmission. Because more processing leads to more resource requirement and more power consumption, it is necessary to decide how much processing should be performed on implantable device. As has been mentioned in 1.2, data firstly has to be processed until accommodated by available bandwidth. After that, to transmit or to continue processing should be determined by power efficiencies of the two options. As both technologies are updating, the required level of on-chip data processing should changes accordingly.

Since wireless transmission of neural signal is outside this research, effort is made to integrate as much processing as possible, in order to provide reference for further decision making.

1.4 RESEARCH HYPOTHESIS

In this work, a software neural processing workflow is proposed to be run on low-power micro chips. Under the implantable constraint, it is expected that such software solution can achieve the same level of processing capability as existing hardware design, thus could be considered for future BCI platform.

To demonstrate the hypothesis, three research goals are set. The first is to design an algorithmic workflow that is suitable for implantable platform. The second is to demonstrate its feasibility on a low-power programmable platform. And finally, based on the result of the second goal, a neuron selection mechanism is proposed and tested to further shrink the gap between current hardware technology and BCIs' requirements.

This research is the first step of the top down design of a brain implantable platform. To the best of our knowledge, this paper is the first to demonstrate the feasibility to implement real-time, on-chip spike sorting on a programmable low power platform with a detailed power/performance trade-off analysis to establish the ground for designing the next-generation implantable BCI.

For the rest of this paper, Chapter 2 will summarize previous works related to implantable wireless BCI devices. Chapter 3 would show the works and results of each research goal mentioned above. Finally, the research is concluded in Chapter 4.

2.0 RELATED WORK

To date, there are many custom ASIC-based spike sorting implementations, but none of them provide a complete on-chip solution. Earlier implementations can only perform bio-signal amplification (Harrison & Charles, 2003) or data compression and transmission (Harris et al., 2008). Recent implementations integrated more computationally intensive functions, such as noise filtering (Chae et al., 2008); (Santhanam et al., 2007) and single channel feature extraction (Moo et al., 2009).

One of the most advanced implantable chips is the INI3 newly-fabricated by (Harrison et al., 2009). It can be flip-chip bonded directly to a 100-channel Utah Electrode Array(Maynard et al., 1997). The chip features 100-channel amplifier and spike detector, plus fully integrated wireless data/power transmission. Power and commands are sent to the chip via an inductive (coil-to-coil) wireless link, the first wireless power supply used in BCIs, and data is transmitted from the chip via a radio-frequency (RF) telemetry link. The chip is 5.4×4.7 mm^2 in size with approximately 10 mW power consumption. The chip supports limited programmability: 1) threshold of spike detection are set manually for each channel; 2) fully-digitized waveform from one user-selectable amplifier would be transmitted. However, waveforms from the rest 99 channels would be irreversibly lost.

The INI3 chip is integrated by Stanford's HermesC-INI3 neural recording system, which has been implanted to free-move monkey object(Chestek et al., 2009). The final PCB design with INI3 chip sizes 30×30 cm^2. Despite INI3's wireless power transmission feature, the HermesC system still rely on battery for power supply. As a result, the final enclosure size $51 \times 38 \times 38mm^3$. The total power consumption is 63.2 mW with typical battery life of 2.9 days.

Besides ASIC designs, Programmable System-on-a-Chip (PSoC CY8C27443 from Cypress Semiconductor) is also used as an implantable BCI device (Mavoori et al., 2005). This chip provides configurable array of analog modules and an 8-bit micro-processor. Due to its limited processing capability, only one channel can be monitored and processed. The resulting device sizes 5.5 cm$\times$5 cm$\times$3 cm with single battery life of 60 hours. It has been successfully implanted and is monitoring free-moving objects autonomously

In conclusion, among existing wireless BCI designs, neither the custom ASIC designs nor the embedded system-based designs perform are able to send out neural signal without information loss. While integrated spike detection hardware is capable of handling sufficient amount of channels, full spike sorting is applicable off-chip to merely single channel, due to the loss of waveforms. Programmability is in general limited to changing a threshold or turning on/off channels. To this end, this paper investigates the power/performance feasibility of using a programmable low-power platform to implement a complete on-chip spike sorting solution.

3.0 RESEARCH AIMS, WORK AND RESULTS

Based on the fact that implantable hardware spike detecting has been very well studied and implemented, it can be convincingly assumed that spike time stamps and waveforms are ready for further processing in the implantable end. Therefore, as is marked in Figure 2, the boundary of this thesis is set to concentrate on the later two stages: spike sorting and neural decoding. Also the power supply issue is outside the boundary of this work, the only power constraint is set by brain tissue's heat tolerance.

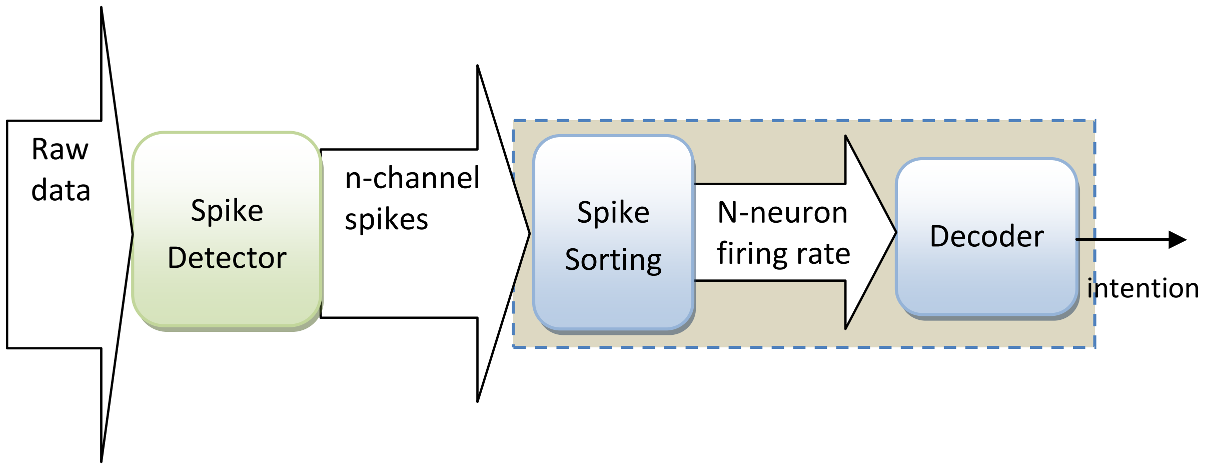


Figure 2 Research boundary

In 3.1 a detailed workflow is proposed by selecting algorithms for these stages and designing the connection between them. Section 3.2 shows how the proposed workflow is implemented on a commercial available low power platform, testing its feasibility. Based on the result of 3.2, section 3.3 proposes a neuron selection mechanism to shrink the gap between current hardware capability and BCI requirements.

3.1 AIM 1: PROGRAMMABLE NEURAL PROCESSING FRAMEWORK FOR IMPLANTABLE WIRELESS BCI

The first research goal aims at designing an implantable processing workflow, which realizes the function of spike sorting and neural decoding. An extensive survey of existing spike sorting and decoding researches is performed to indentify such an algorithmic workflow that is:

- Promisingly light-weight for extreme resource/power-constrained devices
- Sufficiently powerful and fully verified with little prosthetic performance loss
- Capable of real-time processing
- Autonomous (low user respondent burden)

3.1.1 Spike sorting

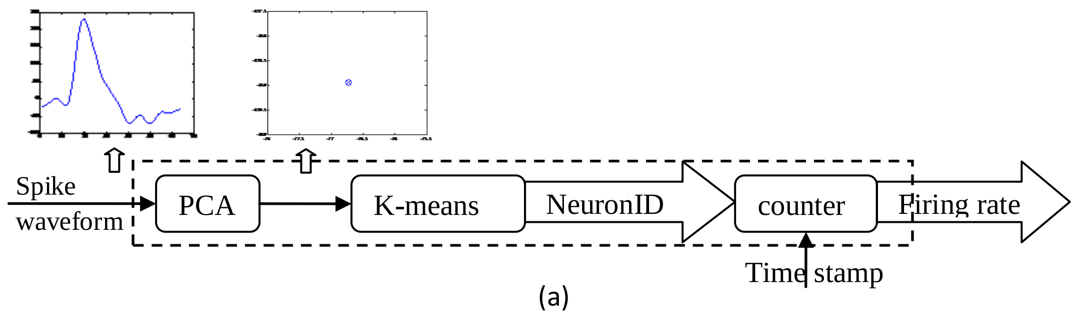
Spike sorting addresses the problem that usually more than one neuron can be recorded by a single channel. It is assumed that each of these neurons would show a relatively unique waveform, due to the distance difference between electrode and the neurons. Thus, spike sorting identifies neurons according to the spikes' waveforms. Autonomous spike sorting approaches usually contain two parts: feature extraction and unsupervised classification (clustering). Both parts require training before processing the signals on-the-fly.

Feature extraction reduces the number of feature dimensions for classification. In this work, we use PCA for this task. PCA is one of the most widely used algorithms to perform feature extraction for spike sorting; it provides the projection that best represents the data in a least-squares sense. The purpose of training is to calculate the principal components (PCs), i.e. the leading eigenvectors of the spikes' covariance matrix. To achieve this, a training set containing several hundreds to thousands of neural spikes are processed to obtain the covariance matrix.

After feature extraction, spikes can be classified based on the new dimension-reduced feature space. Identifying the underlying origin of each neural spike demands an unsupervised classification method. In our work, a *k*-means clustering algorithm was chosen as a light-weight classification algorithm. It is one of the most popular classification approaches used for real time spike sorting, because it is efficient and simple, while providing accurate results in most cases. Training the *k*-means classifier calculates the center (mean) of each cluster.

After the training process of the PCA feature extraction and *k*-means classification, the system can perform on-the-fly spike sorting based on the covariance matrix and the center (mean) of each cluster obtained from PCA feature extraction and *k*-means classification. First, for feature extraction, each sampled spike is projected to the leading two to four PCs directly. The result is passed to the *k*-means classifier, which sorts the projected spike to the cluster whose mean is nearest to it by calculating the Euclidean distance from the projected spike to the center (mean) of each cluster.

The on-the-fly spike sorting need to be performed independently on each channel, shown in Figure 3(a). In this workflow a firing counter is added to the end to further reducing data. As existing decoders only need firing rate information, time stamps can be saved simply by counting the firing times happen within a fixed time window. This also yields the frequency of sending activity to the updating rate of decoder. It is noted that the same workflow must be repeated on every channel to collect all neurons' firing rates for decoding purpose, shown in Figure 3(b).



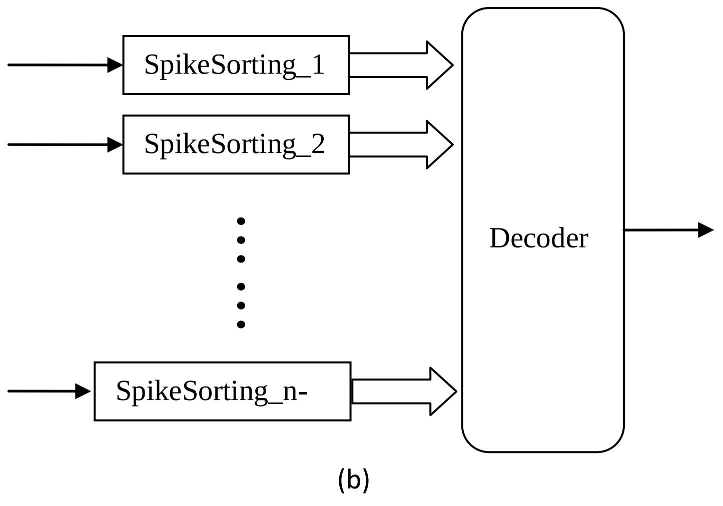


Figure 3 Spike sorting workflow

3.1.2 Neural decoding

Neural decoders translate neuron activities to intention. In this work we focus on the application of reconstructing limb end trajectories from motor neural active potential. The most widely accepted and verified decoding algorithms are simple linear filter (Warland et al., 1997) and Kalman filter (Wu et al., 2003). Previous works claim that there is no absolute winner between the two. The linear filter is supposed to be strong at predicting smooth movements while Kalman filter is believed to be better reacting to fast changing movements. Thus they are both considered in this work.

A. Kalman filter

The Kalman filter is originally used as an optimal linear estimator in stochastic control systems. Its classical application is under the condition when the system's state is desired but cannot be measured directly, what's available is a set of indirect measurements with noise. The function of Kalman filter is to estimate the true state of a system out of these measurements, in a way that minimized the mean square error.

In the application of neural decoding, the neural activities represent measurement and the intention represents the system's state. Since our data is discrete, we use the basic discrete Kalman filter to do the decoding, which assumes 1) the relation between states and relation between state and firing rate are linear and 2) the noise is Gaussian noise.

The following models and equations in the Kalman filter section all refer from (Wu et al., 2003). According to their work, we can define the system state at time $t_k = k\Delta t$ to be x_k, which is a vector containing motion information at time stamp k. In our data, the movement recording includes position L and velocity V in 3 orthogonal directions. So we set $x_k = [\genfrac{}{}{0pt}{}{L}{V}] =$

$[l_1, l_2, l_{3,} v_1, v_2, v_{3,}]^T$. Then we define the measurement to be z_k, which is a $c \times 1$ vector containing firing rate of C observed neurons at time t_k.

The discrete Kalman filter containing two parts: system model(3.1.1) and the measurement model(3.1.2).

$$x_{k+1} = A_k x_k + w_k \tag{3.1.1}$$

$$z_k = H_k x_k + q_k \tag{3.1.2}$$

$$argmin_A(E_s = \sum_{k=1}^{M-1} \|x_{k+1} - Ax_k\|^2 \quad) \quad , \quad argmin_H(E_m = \sum_{k=1}^{M-1} \|z_k - Hx_k\|^2 \ ) \tag{3.1.3}$$

It is assumed that the states are propagate in time according to the system model, in which matrix $A_k \in R^{6\times6}$ relates the states at time t_k and t_{k+1}. In addition, the influence of neural activity is modeled by equation (3.1.2), in which H_k linearly relates the firing rate and system states at time t_k. Noise terms $w_k \sim N(0, W_k)$ and $q_k \sim N(0, Q_k)$ model the Gaussian noise in system. In real world, it is possible that the model A_k, H_k, W_k, Q_k are changing with time. However, to simplify the problem, we assume constant A, H, W, Q, so that the system is time invariant.

Kalman filter also needs training to get the value of A, H, W, Q. This is referred as system identification. For the system identification, a least squares error learning method (3.1.3) is used to get the coefficients in the model. When $x_k \ and \ z_k$ are presented in the training data, the optimal A and H satisfy $\frac{\partial E_s}{\partial A} = 0, \frac{\partial E_m}{\partial H} = 0$. Thus the two models can be derived from equation (3.1.4). W and Q can then be derived by definition of covariance matrix shown in equations (3.1.5), in which M means the total number of time stamps in one movement.

$$A = X_2 X_1^T (X_1 X_1^T)^{-1}, H = ZX^T (XX^T)^{-1} \tag{3.1.4}$$

Where, $$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,M} \\ \vdots & \ddots & \vdots \\ x_{d,1} & \cdots & x_{d,M} \end{bmatrix}, \quad X_1 = \begin{bmatrix} x_{1,1} & \cdots & x_{1,M-1} \\ \vdots & \ddots & \vdots \\ x_{d,1} & \cdots & x_{d,M-1} \end{bmatrix},$$

$$X_2 = \begin{bmatrix} x_{1,2} & \cdots & x_{1,M} \\ \vdots & \ddots & \vdots \\ x_{d,2} & \cdots & x_{d,M} \end{bmatrix}, \quad Z = \begin{bmatrix} z_{1,1} & \cdots & z_{1,M} \\ \vdots & \ddots & \vdots \\ z_{C,1} & \cdots & z_{C,M} \end{bmatrix},$$

$$W = \frac{(X_2 - AX_1)(X_2 - AX_1)^T}{(M-1)}, \quad Q = \frac{(Z - HX)(Z - HX))^T}{M} \tag{3.1.5}$$

Previous work has shown that a training movement longer than 3.5 min is required to obtain a good accuracy. Unfortunately, each clips of movement we have in the data set is no longer than 2 seconds, which is far from that is required. So a learning equation (3.1.6) is derived to achieve the least square error across clips of movement:

$$A = (\sum_i X_2^{(i)} X_1^{(i)T})(\sum_i X_1^{(i)} X_1^{(i)T})^{-1}, H = (\sum_i Z^{(i)} X^{(i)T})(\sum_i X^{(i)} X^{(i)T})^{-1} \tag{3.1.6}$$

The decoding process for t_k is shown in Figure 4. At each time stamp t_k, the Kalman filter takes estimation of system's state x_{k-1} and neural firing rate z_k as input, the initial state x_0 is the true initial state of the system. The algorithm has two steps for each time stamp:

(1) Getting priori estimation using system model and estimation of previous state at t_{k-1}, equation (3.1.7)(3.1.8). In equation (3.1.8), P_k^- is the error covariance matrix.

$$\hat{x}_k^- = A\hat{x}_{k-1} \tag{3.1.7}$$

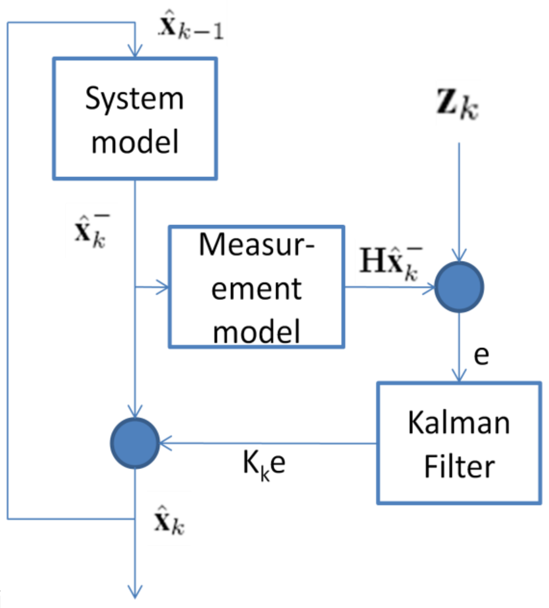
$$P_k^- = AP_{k-1}A^T + W \tag{3.1.8}$$

(2) Updating the priori estimation using new measurement data, equation(3.1.9)(3.1.10). In this step, K_k is the kalman gain, given by equation(3.1.11):

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.1.9}$$

$$P_k = (I - K_k H)P_k^- \tag{3.1.10}$$

$$K_k = P_k^- H^T (HP_k^- H^T + Q)^{-1} \tag{3.1.11}$$



j

Figure 4 Kalman filter updating

B. Simple linear filter

The simple linear filter models the movement position as a linear combination of previous neural activity during a window of N time intervals (usually 0.5 – 1.5 seconds). The time interval has a fixed length ranges from 20ms to 50ms, same as decoder's update period. The model is shown in equation (3.1.12) (Wessberg et al., 2000), in which u_i is the estimation of movement at the *i*th time interval, r_i^v is the *v*th neuron's firing times during time *i*th interval, and f_j^v is one of the linear filter coefficients. These coefficients are acquired through linear regression on training dataset, similar to the training of Kalman filter.

$$u_i = a + \sum_v \sum_{j=0}^{N-1} r_{i-j}^v f_j^v \tag{3.1.12}$$

3.1.3 Experimental result

The data set we use for experiment comes from Prof. Andrew Schwartz's lab. The data contains 87 clips of 3D center out movements, each length around 1 second, recorded from a monkey subject. For each clip of movement, the monkey is required to start from origin and move toward one of 8 directions in the 3D space. The data contains two parts of information, hand movement and neural activity. Hand movement contains the time stamps for sample points during the movement, the sample period ranges from 20-40 ms, but is not fixed. For each sample point of movement, the position and velocity of hand is recorded. The neural activity part is recorded from a sensor array implanted in the motor cortex. The recording contains segmented spikes and their time stamps from 62 channels.

A. Spike Sorting

For the PCA training, 900 neural spikes (each spike contains 48 samples) were used as the training set, shown in Figure 5(a) as aligned raw neural spikes. The leading two PCs were used for the projection. The spikes and the projection points after PCA are shown in Figure 5(b).

The 900 two-dimensional data points, which were the results of the PCA feature extraction, were classified by the k-means clustering algorithm for training. The clustering result is shown in Figure 5(c) and the aligned sorted spikes are shown in Figure 5(d).

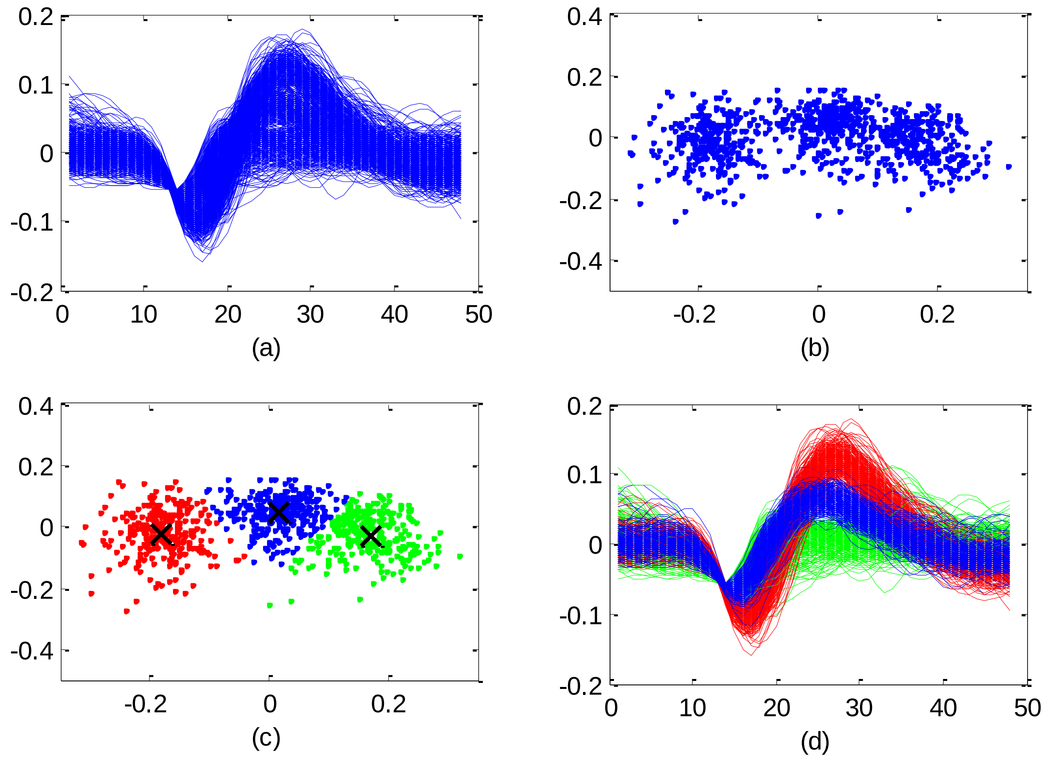


Figure 5 Spike sorting result

B. Neural decoding

The dataset is divided into two parts, the first 77 clips of movement are used for training and the rest 10 clips are used for testing. Figure 6 shows one clip of the actual trajectory recorded in data and the estimated trajectory by decoder. The accuracy is quantified by two parameters: the mean square error (MSE) is calculated between the actual and the estimated trajectory in 3D space; and a correlation coefficient is calculated on each direction. The average MSE and correlation across 10 testing sets is used describe the accuracy of the decoders. In order to fully use the movement record, both decoders are set to update every 30ms.

For Kalman filter, 9 state variables (3D location, velocity and acceleration) are used in the model, in which acceleration is calculated by difference of velocity recording. An alternative of 6 state variables (3D location and velocity) is also tested, but the accuracy is not as good as the one using acceleration information. Since we are reconstructing recorded movement, a lag is added to model the nature delay between a brain activity and the happening of the resulting movement. In our dataset, the kalman filter performs the best when lag equals to 50ms. The result on testing set is shown in the right half of Table 1

For linear filter, only the 3D locations are used as state variables, because in this model all state variables are independent thus more state variables won't help the reconstruction. Other than lag, linear filter has another parameter, window length, which needs to adjust. Window length determines how many previous time intervals are considered when estimating current movement. In our experiment, the linear filter achieves the best accuracy when lag is 50ms, same as kalman filter, and when window length equals to 21 time intervals (a.k.a bins). The result is shown in the left half of Table 1.

Table 1 Decoding accuracy

Simple Linear Filter				Kalman Filter			
ave_mse(cm^2)	ave_corrcoef			ave_mse(cm^2)	ave_corrcoef		
	x	y	z		x	y	z
4.316	0.904	0.857	0.785	4.119	0.978	0.910	0.705

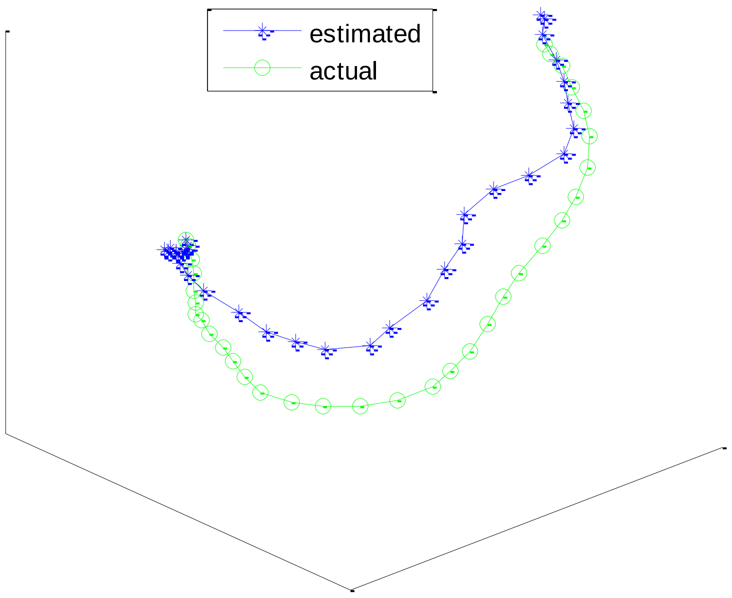


Figure 6 Neural decoding result

3.2 AIM 2: IMPLEMENTATION AND FEASIBILITY TEST ON IMOTE2 PLATFORM

In this section, we investigate the feasibility of performing on-chip, real-time spike sorting using a commercially available and programmable wireless sensor network (WSN) mote – Imote2 – to function as a programmable BCI platform capable of performing the following computing tasks: 1) neural signal processing to obtain spiking rates, 2) on-the-fly neural decoding using linear filter. The WSN mote is arguably the closest programmable platform that can be built from today's technology to the ultimate multi-use brain-implantable computing platform for BCI devices due to its small size, wireless telemetry capabilities, as well as similar resource and ultra low-power constraints.

The contributions of this section are: 1)We proposed to adopt the compact and efficient TinyOS event driven execution model and fully implemented this lightweight real-time spike sorting algorithmic workflow, including the compute-intensive feature extraction and classification, in the nesC language; 2) we used the Imote2's dynamic voltage and frequency scaling (DVFS) capability to characterize the correlation between execution time, power dissipation, and power density of our proposed spike sorting workflow. To the best of our knowledge, this is the first work to demonstrate the feasibility to implement real-time, on-chip spike sorting on a programmable platform with a detailed power/performance trade-off analysis to establish the ground for designing the next-generation implantable BCI

3.2.1 Implementation on Imote2

The operating system (OS) selected for use is TinyOS (Levis et al., 2005). It is an extremely memory and power efficient event-driven OS designed to support networks and I/O devices on a small WSN platform using only approximately 200 bytes of memory, thus making it attractive for an implantable BCI chip. Both the TinyOS kernel and the applications deployed, such as our proposed real-time spike sorting workflow, are written in nesC (Gay et al., 2003), which is an extension to the C programming language designed to support the TinyOS concurrency model based on tasks and hardware event handlers. The whole-program compilation process allows for better code generation and static timing analysis to increase runtime efficiency. The usage of the RAM and ROM for the TinyOS are shown in Table 2, 0.5% of a 32

MB ROM, and 6.5% of a 256 kB RAM. Figure 7 illustrates how the applications (spike sorting and simple linear filter) are integrated to the TinyOS system. The main application module (for instance, SpikeSortingM) is interconnected with five TinyOS Components. The SingleTimer is a timer used to trigger the spike sorting operation, and will be stopped once the spike sorting program starts; the LEDs module is used to control the on-board LEDs to indicate the states of the spike sorting; the SysTimeC module is used to calculate the runtime of the spike sorting algorithm; and the DVFSC module is used change the frequency/core voltage of the CPU. The spike sorting code is called in the interrupt handler of SingleTimer.

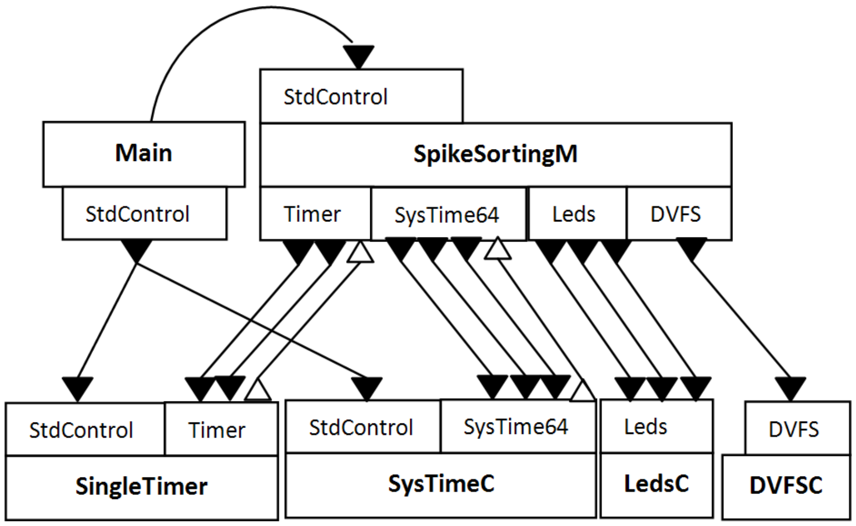


Figure 7 TinyOS integration

3.2.2 Experimental procedure

The TinyOS environment was setup in Windows XP. This step consisted of installing Cygwin, the TinyOS source tree, a nesC compiler, and GCC cross-compilers, which allowed for the development of applications for the Imote2. All spike sorting processing, compression, and transmission algorithmic components were first implemented and verified in ANSI C codes. To deploy these algorithms on the Imote2 board, we manually converted the ANSI C into nesC in order to fit the TinyOS event-driven execution model, and compiled them with the TinyOS cross compilation tools. Binaries were obtained, as well as the size of the code for the RAM and ROM portions of memory. Figure 8 shows the procedure workflow to deploy the PCA, *k*-means clustering onto the Imote2, then the core frequency and core voltage is changed to measure the varied execution time and power consumption. Matlab also was used to verify and visualize the output results of each algorithm from the Imote2.

Code was downloaded to the Imote2 board via the USB interface and was run making use of the pre-installed bootloader for TinyOS. We used the Imote2's dynamic voltage and frequency scaling (DVFS) capability to determine the correlation between application execution time and power dissipation. Power dissipation was calculated by measuring the voltage and current supplied from the Imote2 battery board using a digital multimeter, as shown in Figure 8. Execution time was measured using the built-in timer provided by the TinyOS. The Imote2-Console application provided with the TinyOS distribution is an interface between the host PC and the Imote2 board and was used to obtain execution time results.

Table 2 Imote2 memory usage

	Imote2 Total Memory	TinyOS (kB)	PCA (kB)	*k*-means (kB)	Linear Filter (kB)
ROM	32MB (Flash)	157 (0.5%)	178 (0.5%)	246 (0.7%)	187(0.6%)
RAM	256kB (SRAM)	17 (6.5%)	34 (13.4%)	17 (6.5%)	20(7.8%)

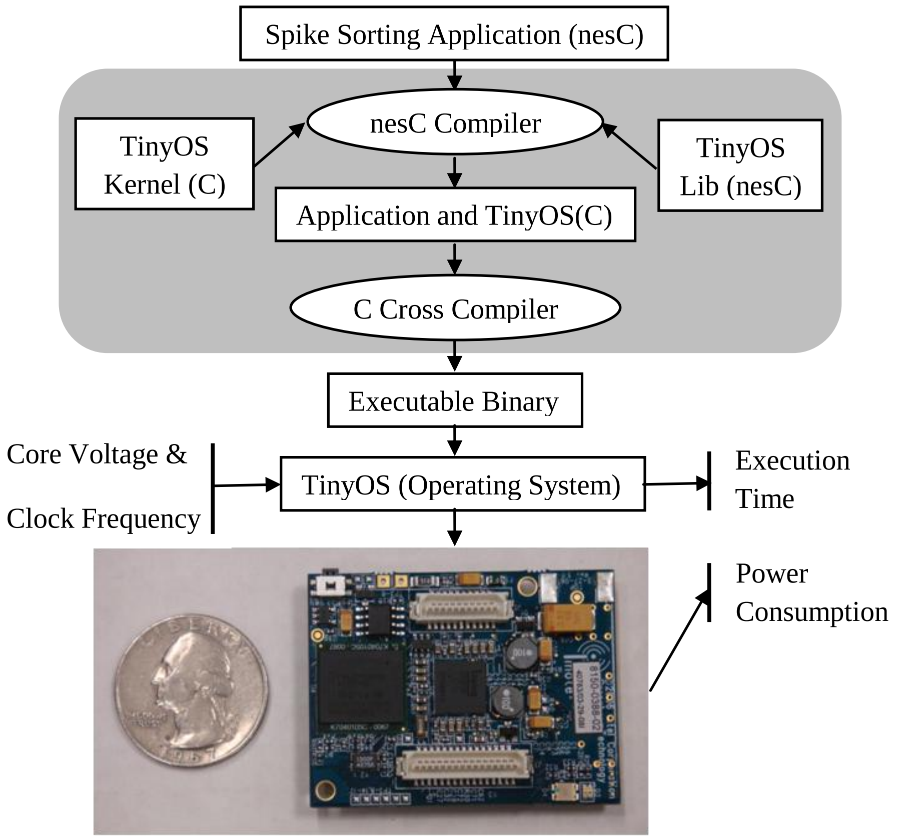


Figure 8 Imote2 experimental procedure

3.2.3 Experimental Results

The proposed implantable neural processing workflow was implemented on the Imote2, and measurements of the execution time, power consumption, and memory resources were taken. In section A to C, we present the performance, power, and memory footprint analysis for spike sorting algorithms; in section D, we discuss the implications and insights gained from these results. The program's execution time, power consumption, power density, and memory resource utilization were analyzed for each spike sorting stage. The dataset used is a 30 minute multiunit recording from a human epilepsy patient obtained from http://www.vis.caltech.edu/~rodri /Wave_clus/ Wave_clus_home.htm. In section E, the same analysis is performed on on-the-fly simple linear filter decoding. For this test, the dataset used is the same as that in section 3.1.

A. Training: Feature Extraction

For the PCA training, 800 neural spikes (each spike contains 64 samples) were used as the training set. The memory resources used for PCA are shown in Table 2. The PCA algorithm occupies 0.5% of the ROM and 13.4% of the RAM. Execution time and power consumption were measured at each of the four available core frequencies (13, 104, 208, and 416 MHz) on the Imote2, shown in Figure 9.

B. Training: Classification

The code size for the k-means clustering algorithm is shown in Table 2. Execution time, power consumption, and power densities are shown in Figure 10. The classification algorithm executes faster than the feature extraction algorithm. However, both processes have roughly the same power consumption.

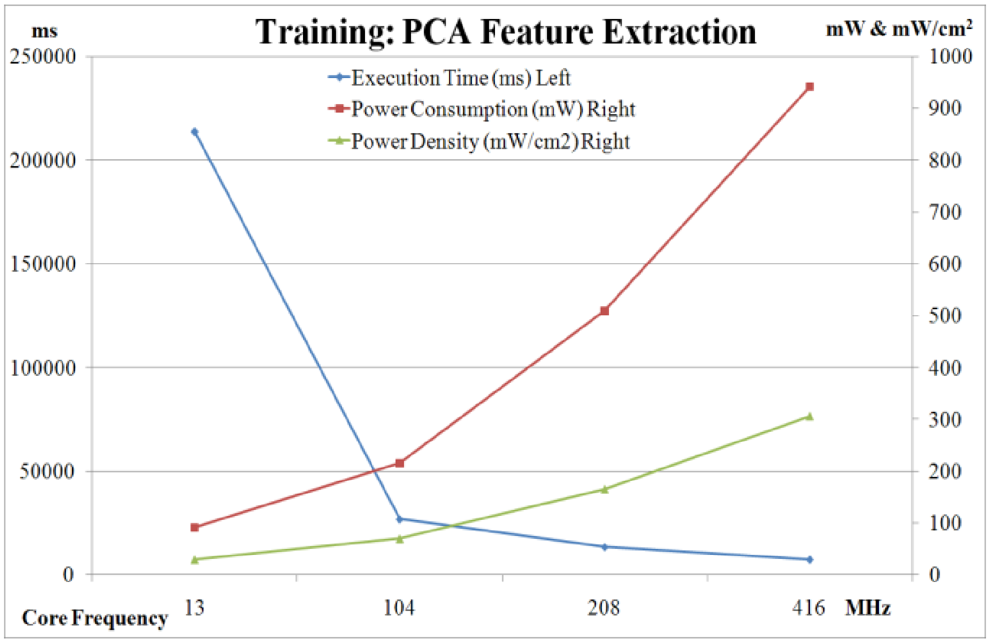


Figure 9 PCA performance measurements

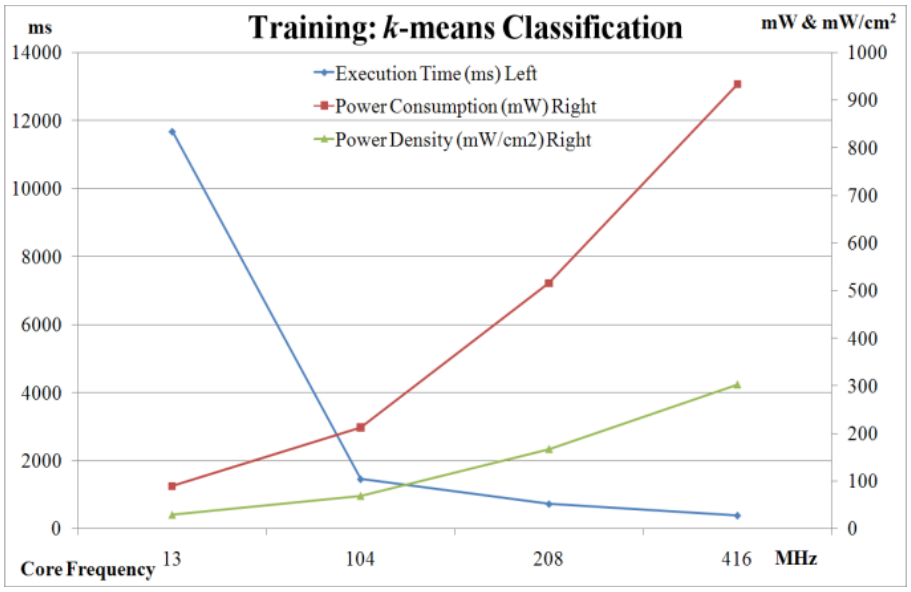


Figure 10 *k*-means performance measurements

C. On-the-Fly Spike Sorting

The performance of the on-the-fly spike sorting for 800 spikes was measured. The spikes were continuously fed to the Imote2 without a pause between spikes, simulating the fastest possible spiking rate. Each spike was directly projected to the leading two PCs and then classified according to its distance from the mean of each cluster. Figure 11 shows the execution time, power consumption and power density results. At 13 MHz the on-the-fly spike sorting takes 7,353ms and at 416 MHz it takes 240 ms to complete – an appreciable decrease in execution time. However, the power consumption increases from 87 mW at 13 MHz to 915 mW at 416 MHz.

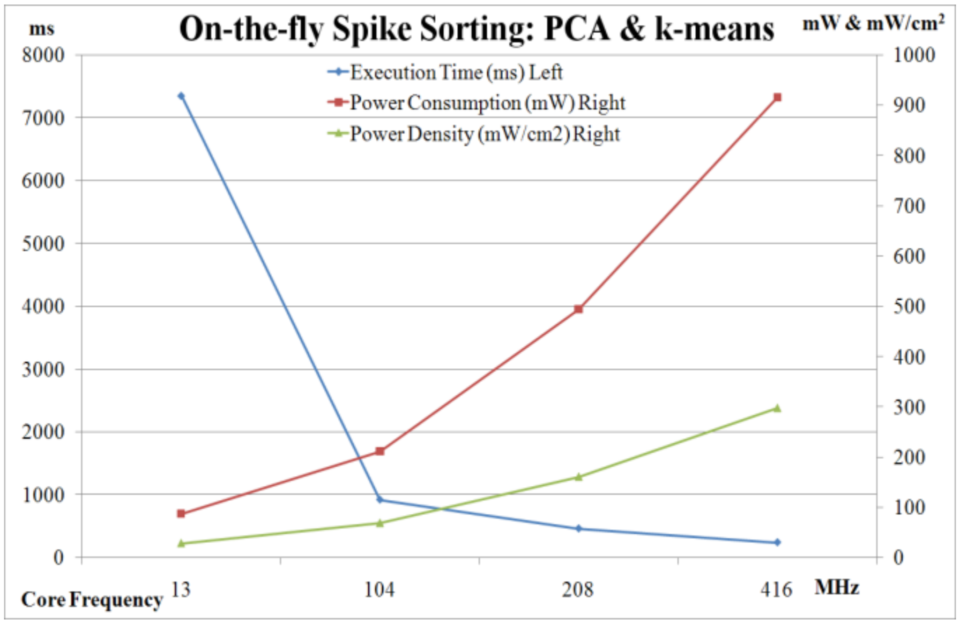


Figure 11 On-the-fly spike sorting measurement

D. Discussion on on-chip spike sorting

So far, three performance areas were measured for different spike sorting tasks, which are code size, execution time, total power consumption and power density. In general, we observed a 2-phase behavior for execution time: an approximately one order of magnitude reduction from 13 MHz to 104 MHz and a consistent 2x linear reduction beyond 104 MHz. On the other hand, the power that the Imote2 consumed increased linearly by a factor of 2x from 13 MHz to 416 MHz. In this research, the power consumption of the entire Imote2 board was measured, because besides the processor, other functional modules, such as the radio and the antenna, are also present for a BCI. Power densities were calculated from the real power measurements using the assumption that the total power was solely dissipated by the PXA271 processor chip, whose single-side area is 1.54 cm2. This gives the worst-case power density upper bound for the Imote2. To satisfy the physiological limit of 62 mW/cm2 as discussed in introduction, 13 MHz is the only core frequency among the four available frequencies that meets this power density requirement.

To calculate how long it takes to process each spike in real-time, the normalized sum of the on-the- fly spike sorting was calculated. Even at the slowest core frequency of 13 MHz, the entire processing flow finishes in 9.6 ms, thus meeting the real-time requirement of 12.5ms derived in section 1.3. The on-the-fly execution time is proportional to sample rate, i.e. the number of data used to represent a waveform. Meanwhile, the real-time requirement would be inverse proportional to total number of channels, assuming there is no parallelism or resource reuse. Note that for feature extraction and classification training, there is no real-time requirement since it can be performed off-line, so only the power density requirement needs to be met.

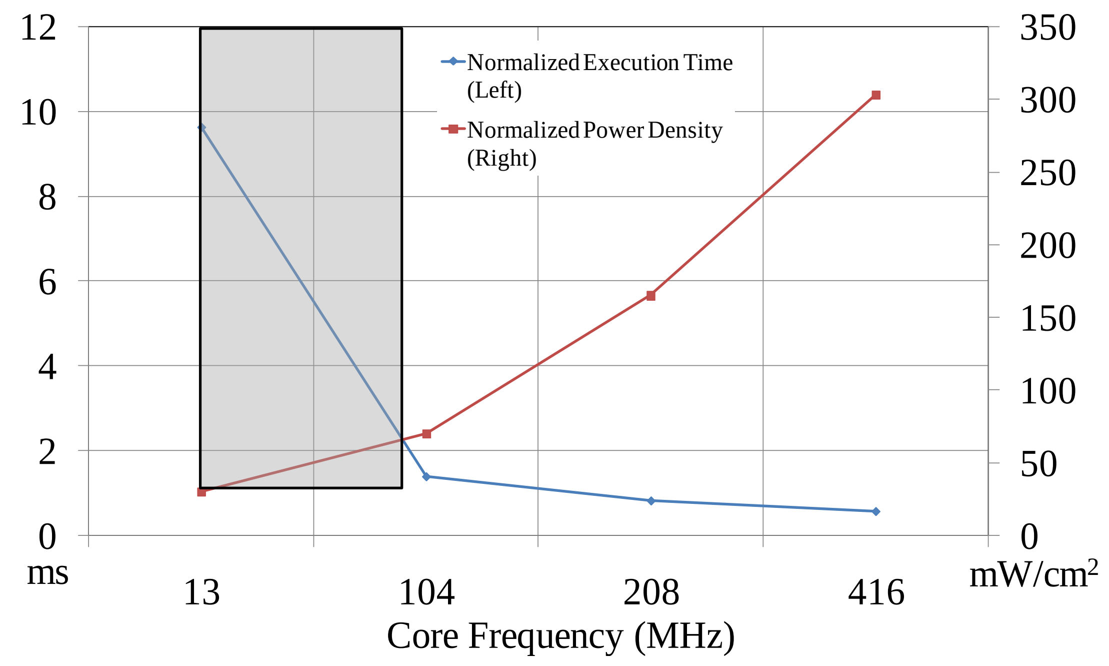
Theoretically, if there were more frequency choices provided by the power management unit on the Imote2, the maximum clock frequency that meets the power density requirement can be estimated to be 86 MHz, which is estimated from Figure 12 (a), which shows the power density –frequency relationship. Also plotted in Figure 12 is the normalized execution time for the on-the-fly processing case. The range of operation, highlighted in Figure 12 (a), indicates that the Imote2 can meet the power density and execution time requirements from 13 MHz to 86 MHz, resulting in power densities ranging from 30 mW/ cm2 to 62 mW/ cm2 and the execution time ranges from 9.6 ms to 3.0 ms.

As a result, it is only feasible to perform spike sorting on-chip on 1 to 4 channels, still far from 50 to 100 channels available on state-of-the-art sensors. However, this capability is comparable to what can be achieved by current ASIC designs.

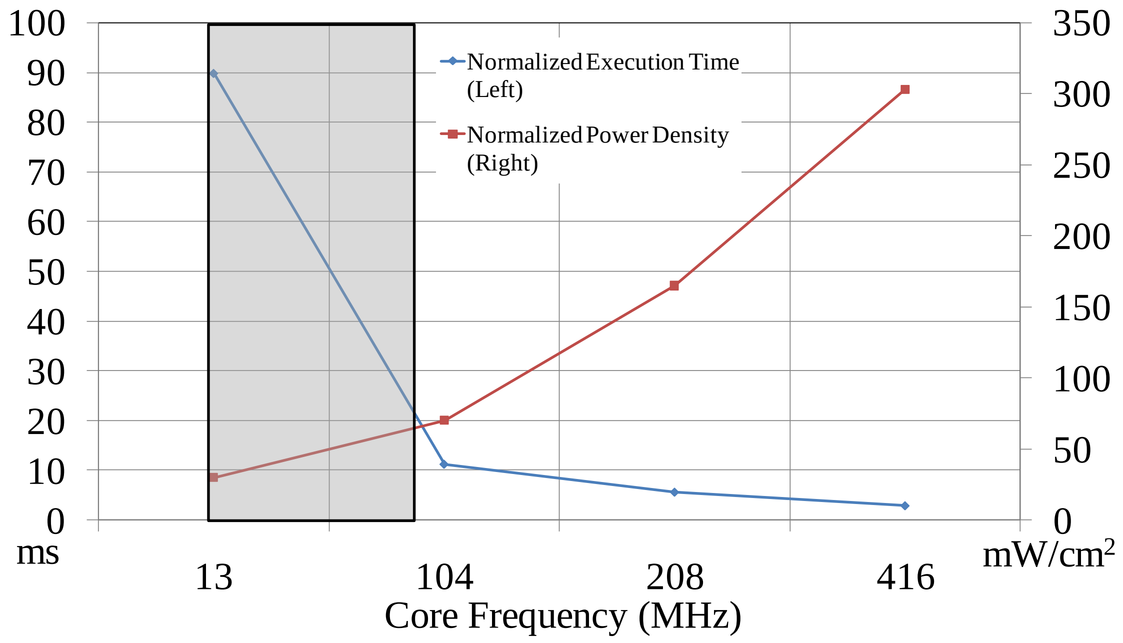
E. On-the-fly simple linear decoding

Up till now, the on-chip neural processing has reached to its bottleneck, spike sorting on all channels. As a result, it is not possible to generate the full firing rate information on Imote2, thus is not feasible to integrate later stages. However, for future performance reference, the simple linear filter decoding is also tested on Imote2, assuming the firing rate is available or the no spike sorting option is taken yet further data reduction is needed.

The normalized execution time (time used to calculate one step of movement) is plotted in Figure 12 (b). It is proportional to window length and number of neurons. Given the updating period of 20-100ms and the power density bound, the range of operation is also highlighted.



(a)



(b)

Figure 12 Excution time vs. power density

3.3 AIM 3: A NEURON SELECTION MECHANISM FOR NEURAL DECODER

Base on the experimental results on Imote2, it can be concluded that current hardware technology cannot support on-chip spike sorting for all channels. In order to fill this gap, two obvious directions can be explored: 1. Improve current hardware design; 2. Reduce the computation requirement of current workflow. This thesis focuses on the later.

Since light-weight algorithms have been chosen for each stage of spike sorting, there is little potential to further reduce the work load of a single channel. Therefore the research goal is set to be reducing the number of channels which need to be processed. This brings up the questions of how many channels are sufficient for the neural decoder, and if not all channels required, which channels should be selected. In this chapter, a light-weight neuron selection mechanism based on correlation is added to the training of decoder. Result shows that not only up to two third of computation could be reduced, but the MSE could also be reduce by 3%-22%.

3.3.1 Related works

Several works on neural decoders have investigated the dependence of decoding accuracy on the number of neurons available to the decoder.

Wessberg and colleagues performed a neuron-dropping analysis on both linear and nonlinear decoding algorithms, which are used for real-time predictions of one- and three-dimensional arm movement trajectories (Wessberg et al., 2000). It is an offline analysis, starting from the full ensemble used for decoding, then randomly dropping one neuron at a time, re-training and testing the decoders on one dimension with the remaining ensemble and. By doing this, a curve describe prediction accuracy (measured in terms of mean percentage of total hand position variance accounted for) as a function of ensemble size is obtained. Such neuron-dropping analysis is performed on both linear and ANN decoders. For each decoder, neurons from different cortical areas are analyzed independently as well as in combination. The authors claim that all neuron-dropping curves can be fitted by hyperbolic functions $y = \frac{cx}{1+cx}$, though the parameter c is different for each cortical area.

The same method is also used by Hatsopoulos and colleagues on their linear decoder for continuous limb trajectories (Hatsopoulos et al., 2004). A monotonic improvement in mean reconstruction performance with ensemble size is observed in all neuron-dropping tests. Moreover, their result indicates that the performance is highly unevenly distributed among neurons, suggesting that certain neurons or groups of neurons perform much better than others. However, how to pick these high-performance neurons without the neuron-dropping analysis is not addressed in this work.

Kemere and colleagues suggests using the number of neurons required to achieve certain performance as a metric for evaluating different decoding techniques (Kemere et al., 2004). In this work, a model-based decoder is proposed and compared with linear decoder. Decoding MSE is plotted as a function of ensemble size, showing that given a maximum acceptable MSE of 10cm^2 their decoder requires less than half of the neurons the linear decoder would need. Unfortunately, only synthetic neural data is used for this analysis in order to avoid errors introduced by spike sorting.

From these papers, it can be concluded that,

i) For a given ensemble size, decoders perform differently with each unique subsets of the same group of neurons, indicating that if ensemble size is reduced, it is possible to minimize the loss of accuracy by choosing the optimal subset.

ii) Existing neuron dropping analysis generates subset of neurons randomly. Little attention has been paid to develop a computational efficient method for selecting the best subset.

3.3.2 Neuron selection mechanism based on correlation

This work aims at reducing the number of channels to be processed with little as possible sacrifice on decoding accuracy. To achieve this, it is desired to keep the channels that contribute the most to decoding, and turn off those that are collecting data from trivial neurons. As a result, a neuron selection mechanism is inserted to the training process of BCI. Its function is to pick out the most significant subset of an ensemble, and only keep those channels monitoring them on. In this work, it is proposed to measure the contribution of each neuron by correlation coefficients between neuron activity and state variables. Using the ranking of absolute value of these correlation coefficients, the most significant subset can be determined given either the minimum acceptable accuracy or the desired size of ensemble.

The decoder training process that includes the proposed neuron selector is described in the diagram in Figure 13(a). The selection (red block) is performed before the training of decoder. It takes sequence of firing rate and movement state variable as input, and output a list of significant neurons. According to the channel-neuron mapping given by the spike sorting stage, a list of significant channel is generated. The channels outside the list would not be processed by on-the-fly spike sorting, and could be turned off for further power saving. The training of decoder and overall on-the-fly BCI processing remain the same, as is shown in Figure 13(b), despite that only a subset of ensemble is monitored.

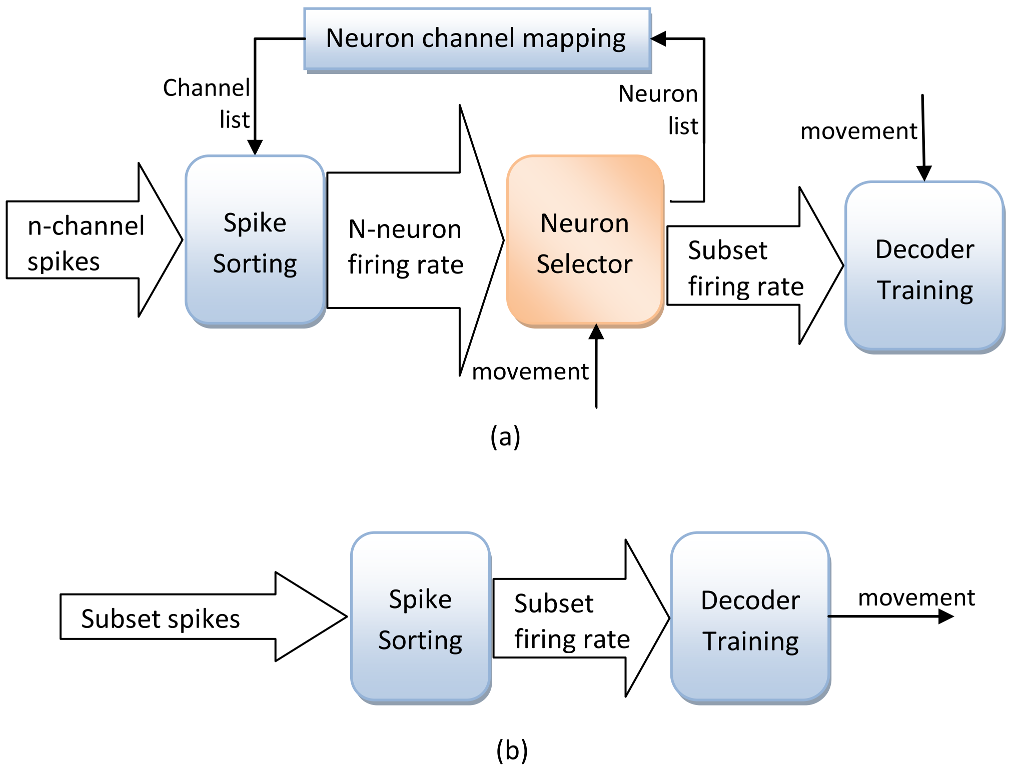


Figure 13 Workflow with neuron selection

3.3.3 Implementation and experimental procedures

The neuron selection is implemented and verified in Matlab. It is tested on both tuned Kalman filter and simple linear filter. The experimental dataset is the same as that is used in previous decoding test.

In this work, the selection mechanism is implemented in three steps:

First, a correlation matrix is formed by calculating one sample correlation coefficient between each movement variable sequence and each neuron's delayed firing rate sequence, in which the delay is set according to the optimal delay acquired from Kalman filter experiment. Assuming the size of ensemble is *M* and the number of state variables is *D*, the correlation matrix would have *M* rows and *D* columns. The sample correlation is calculated using equation (3.3.1), in which $r_{xy}(p,q)$ is on row p and column q of the correlation matrix, $x_p[n]$ represents the time sequence of the pth neuron's firing rate, with sample mean $\bar{x}_p$ and standard deviation σ_{x_p}, and $y_q[n]$ is the time sequence of the qth movement state variable, with sample mean $\bar{y}_q$ and standard deviation σ_{y_q}, N is the length of these sequences. This sample correlation is an estimation of Pearson's linear correlation between x_p and y_q.

$$r_{xy}(p,q) = \frac{\sum_{i=1}^{N}(x_p[i]-\bar{x}_p)(y_q[i]-\bar{y}_q)}{(N-1)\sigma_{x_p}\sigma_{y_q}} \qquad (3.3.1)$$

For the experimental dataset, there are 91 neurons after spike sorting and 9 state variables, resulting to a 91 by 9 correlation matrix. Each column(state variable) is plotted in Figure 14. Integers 1 to 91 on horizontal axis represent the ID of 91 neurons. The vertical axis is the correlation coefficient between these neurons' firing rates and one of the nine state variables. As is marked in Figure 14, the upper panels correspond to position variables, the middle ones are for velocity variables, and the lower panels correspond to acceleration variables. The three columns of panels represent three dimensions in the movement space. From the correlation matrix, it is observed that the mean correlation is close to 0, supporting the neurons' uneven contributions to the decoding result. It also can be seen that velocity variables provide the most deviation, while the acceleration variables show the least, which implies that the monitored ensemble controls velocity instead of acceleration, which is coincide with observations in other neuroscience research. It is also reasonable that the position variables have in general smaller correlation than velocity variables, since the firing rate at single time stamp determines the transient rather than accumulated effect.

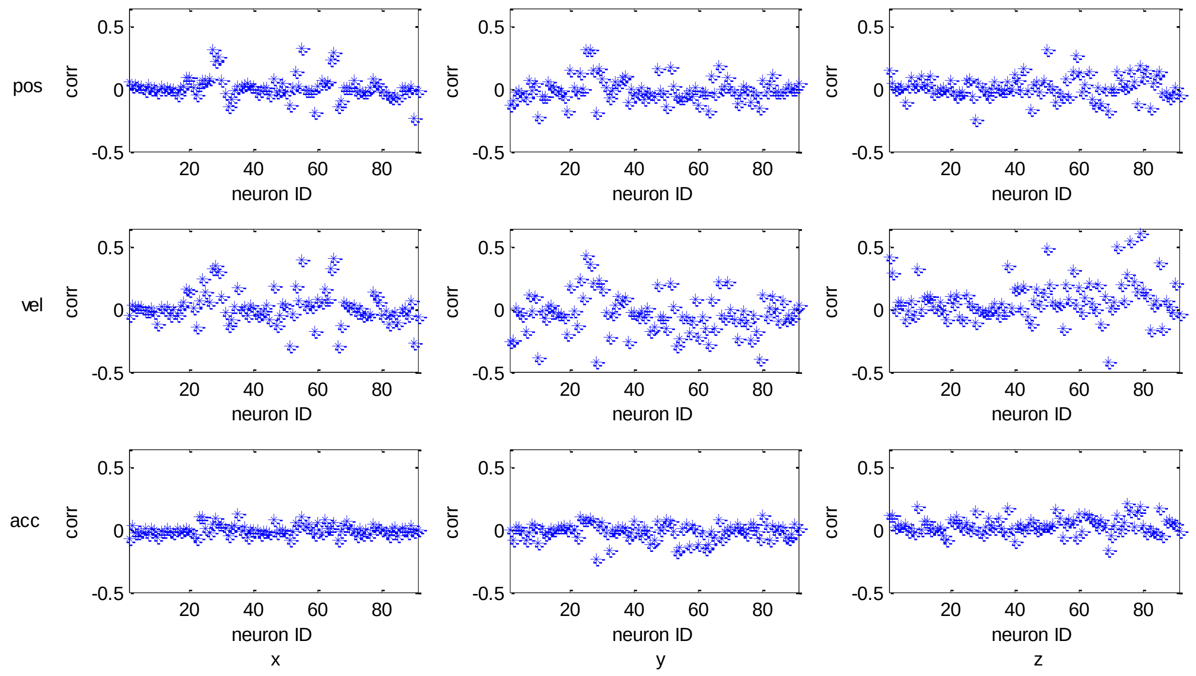


Figure 14 Correlation between firing rate and state variables

The second step is to perform a sorting on the absolute values of correlation along each column (state variable) of the correlation matrix. This would generate a ranking of significant neurons for each state variable. Thus the output of this stage is defined as ranking matrix.

In the third step, a selector would pick the top $m(m<M)$ neurons from each ranking corresponding to the variables, then combine identical neurons to create a list of subset. m is a parameter set by user regarding to the desired size of subset.

To test the proposed neuron selection mechanism, the subset of ensemble gained from selection is used to train and test tuned Kalman filter and simple linear filter. The training and testing procedure is the same as 3.1.1. By changing parameter m in step three, curves that describe decoding accuracy measurements as functions of ensemble size can be obtained. Since our goal is to reduce the number of channels, ensemble subsets are mapped to channel subsets to show the actual saving of computation.

This experimental procedure is similar to the neuron dropping analysis performed by other related works in that each subset is included by all larger subsets. What's unique is that the subset is selected in a certain order described above, in stead of random generation.

3.3.4 Experimental results

The described test is first performed on tuned Kalman filter with a lag of 50ms. The result accuracy vs. number-of-neurons curve is plotted in Figure 15(a), in which the upper left panel shows the average MSE between the predicted and actual trajectories, and the rest panels show the average correlations between predicted and actual trajectories in all 3 dimensions. The blue lines, whose values equal to the point at 91 neurons, show the accuracy measurements using full ensemble. And the red lines show the corresponding accuracy measurements without spike sorting. Similarly, Figure 15(b) shows the test result on tuned simple linear filter with filter length of 21 and lag of 50ms.

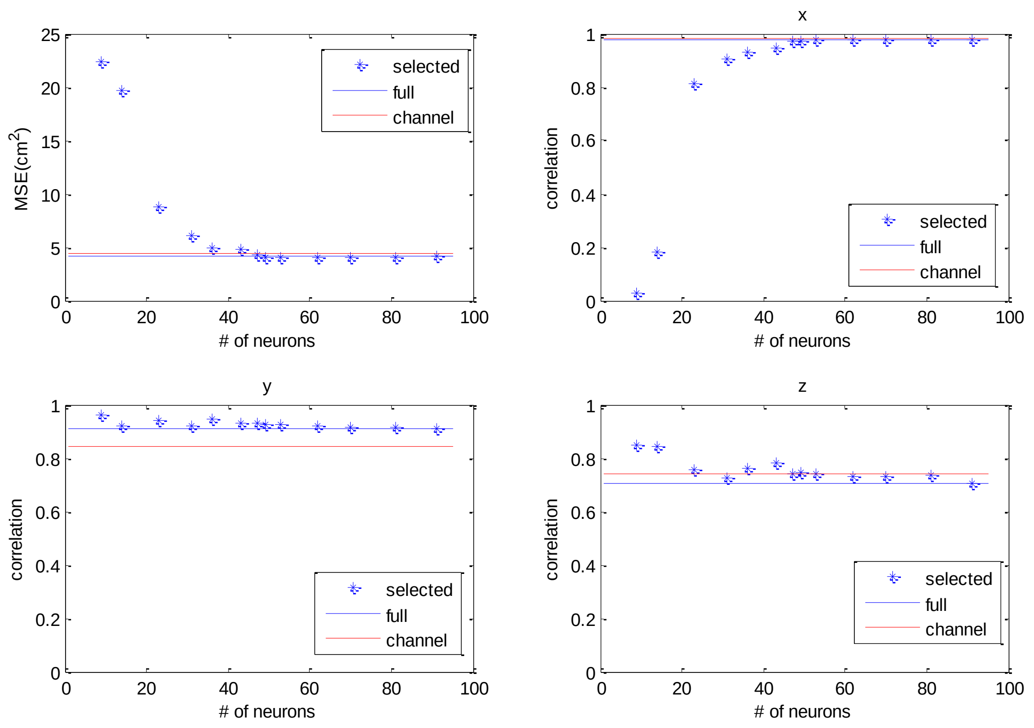
As being observed in previous works, the MSE function has a saturate region, in which the growth of ensemble size leads to little improvement. What has not been reported is that both types of decoder give a smaller MSE by using certain subsets then using the full ensemble. Especially in linear filter's result, the MSE is reduced by about 1 cm^2 using less than half of the neurons. This is possibly because neurons that are barely related to target variables would act as noise in decoder. These noisy neurons can be filtered out by proper selection of ensemble subset such as the proposed mechanism. In contrast, the filtering can hardly be achieved by random selection, which is a possible reason why it has not been observed in previous neuron dropping analysis. Regardless the possible neuroscience explanations lying behind it, this phenomenon is important to BCI designer in that it indicates an optimal ensemble size, which leads to a win-win situation for power consumption and decoding performance.

To determine how many neurons are sufficient, it is desired to know the minimum acceptable accuracy, which is measured by MSE in this paper. However, this requirement is highly application dependent. And because the neural prosthetics technology is still in its infant stage, quantified requirements for different applications have rarely been addressed. (Kemere et al., 2004) used an arbitrarily select 10cm^2 as the maximum acceptable MSE, which is not comparable to the scale of our data. Another problem of comparing the experimental MSE directly with absolute requirement, if existing, is the fact that the prediction is performed open-loop from recorded neuron activity. The same decoder would perform differently, usually better, in closed-loop test on animal or human objects. Due to these two problems, the minimum acceptable accuracy is set from the following relative perspective. Considering our goal is to reduce the number of channels that require spike sorting, and the reason of performing spike sorting is to expect a better decoding accuracy, the maximum acceptable MSE should be no greater than the MSE of decoding without spike sorting(red lines in Figure 15), or the cost of performing spike sorting would be worthless.

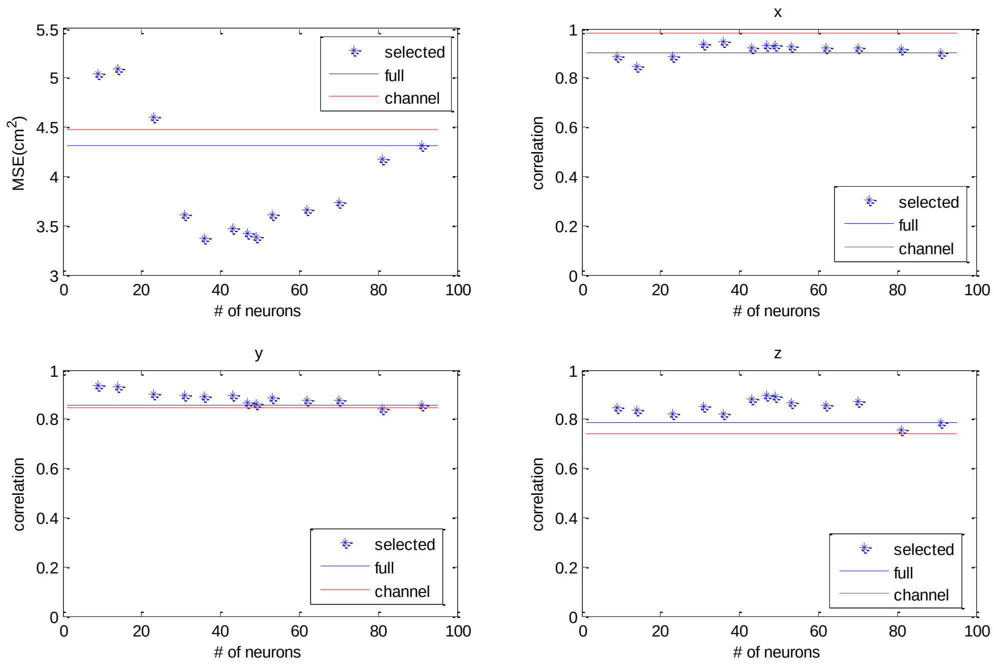
Given this maximum acceptable MSE, Table 3 concludes the channel saving after neuron selection. For each decoder, two significant points in MSE vs. number-of-neurons are shown in the Table 3. The optimal point represents the experimental result that achieves the minimum MSE. And the acceptable point is the data point that falls below the red line with smallest threshold m. The relative MSE is calculated by dividing the actual MSE with MSE achieved when using the whole ensemble. For our dataset, the proposed selection mechanism can reduce the channel to as low as one third of original, meaning that more than 60% of computation is saved. The simple linear filter in general requires fewer neurons than Kalman filter. This is because its time window makes relatively full use of each neuron’s information. Then from the computation consumption point of view, simple linear filter is better than Kalman filter in that it more efficiently utilize the neuron activity.

Table 3 Performance of neuron selector

	Kalman Filter		Simple Linear Filter	
	optimal	acceptable	optimal	acceptable
relative MSE(%)	96.5	104.3	78.1	94.5
channel saving(%)	22.6	37.1	50.0	62.9



(a)



(b)

Figure 15 Neuron selection test result

4.0 CONCLUSION AND FUTURE WORK

In this research, we proposed and implemented a programmable solution for single channel neural signal processing, and simple linear filter decoding with no spike sorting option, prototyped on the TinyOS-enabled Imote2 WSN mote. The training and testing of feature extraction and classification stages of spike sorting as well as on-the-fly neural decoding for BCI neural signal processing were implemented in both ANSI C and nesC and performance, power, and resource utilization were measured from the Imote2 using the TinyOS and a digital multimeter.

The experimental results of 9.6 ms execution time per neural spike and a power density of 30 mW/cm2 presented in this Imote2 feasibility study show that it is possible to perform real-time spike sorting and wireless transmission on a single neural channel using a programmable platform. Given a finer division of available core frequency in the low frequency region, 1-4 channels could be processed. The performance can be further improved by using the standard miniaturization techniques and novel architectural and VLSI innovations, which are expected to provide further reductions in execution time and power dissipation.

This feasibility study establishes the ground for the further exploration of spike sorting on a multi-use programmable platform. With the programmable ability, additional features can be implemented in software, as opposed to designing a new custom ASIC, allowing for a more flexible, multi-use implantable computing platform with rapid development.

Based on the ground work and insights gained from this single-channel recording implementation, we will explore multi-channel recording schemes optimized with chip-scale or distributed parallel processing techniques.

On the other hand, to narrow the gap between available processing capability and the requirement of processing all channels on an implanted electrode array, a light-weight neuron selection method is proposed and tested on Matlab. The correlation based neuron selection is performed during training so no extra burden is added to on-the-fly workflow. Experiment on tuned Kalman filter and simple linear filter shows that not only up to 60% of the 62 channels can be saved, it is also possible to reduce the MSE by 20% using only half of the channels, due to the elimination of noisy neurons.

However, whether such correlation based selection can acquire optimal subset is not proved. So in the future it is desired to model the different contribution of neurons and derive the selection method in theory. If better dataset is available, it is also important to further verify this method.

APPENDIX A

CODE FOR NEURAL PROCESSING

A.1 SPIKE SORTING

```
result_t PCA_Kmean()
{
//FILE *stream;
int  n, m,  i, j, k, k2;
float interm[COLUMNS+1];
//float in_value;
//char option;


  n = ROWS;              /* # rows */
  m = COLUMNS;              /* # columns */
    /* Form projections of row-points on first two prin. components. */
    /* Store in 'data', overwriting original data. */
    for (i = 1; i <= n; i++) {
     for (j = 1; j <= m; j++) {
```

```
    interm[j] = data[i][j]; }   //data[i][j] will be overwritten
    for (k = 1; k <= 2; k++) {
      data[i][k] = 0.0;
      for (k2 = 1; k2 <= m; k2++) {
        data[i][k] += interm[k2] * symmat[k2][k-1]; }
    }
        Class_tag[i] = grouping(data[i][1],data[i][2], Centroid);//grouping based on min distance
  }
        return SUCCESS;

}
```

A.2 NEURAL DECODING

A.2.1 Linear filter

```
%linear_filter_tb
clear
redo_moveset=0;
lag=0;
bin_size=0.03;%0.05
filter_length=24;%0.05
SMOOTHING_WIDTH=17;%for decoding only
```

```
moveset_file='moveset_linear.mat';

if(redo_moveset==1)

%moveset=make_R_matrix_batch(move_filename,bin_size, filter_length,lag)

moveset=make_R_matrix_batch('moveset.mat',bin_size,filter_length,lag);

%%combining the cells to channel

load channel_idx.mat

moveset=ch_comb_batch(moveset,ch_idx);

cd D:\UPitt\academic\research\programming\neural_action_potential

save (moveset_file, 'moveset');

end

%training

%parameters for training set and sample set

ntrain=77;

ntest=10;

filter=linear_training(moveset_file,ntrain,filter_length);

%training evaluation

mse_train=0;

corr_train=zeros(1,3);

for trainset=1:ntrain

    [ori, est]=linear_decoding(moveset_file,trainset,0,filter,SMOOTHING_WIDTH);

    [temp1,temp2]=evalue(ori,est);

    mse_train=mse_train+temp1;

    corr_train=corr_train+temp2;
```

```
end

 mse_train= mse_train/ntrain;

 corr_train=corr_train/ntrain;

%test

draw=0;

mse=zeros(0);

corr=zeros(0,3);

%testset 78:87

for testset=ntrain+1:ntrain+ntest

    [ori, est]=linear_decoding(moveset_file,testset,draw,filter,SMOOTHING_WIDTH);

    [mse(testset-ntrain),corr(testset-ntrain,:)]=evalue(ori,est);

end

ntest=size(mse,2);

ave_mse=sum(mse)/ntest;

ave_corr=sum(corr)/ntest;

result=[ave_mse,ave_corr,mse_train, corr_train];

clear draw ori est trainset testset ntest temp1 temp2 ntrain ntest
```

A.2.2 Kalman filter

```
%kalman filter testbench with lag

%get moveset with lag, lag is in seconds

lag=0.05;

%set the moveset file name, the name is used for store and read
```

```
moveset_file='moveset_acc_lag.mat';

moveset=batchmovecut(lag);% the arrangement of state variable is[location,vol,acc]

%%combining the cells to channel

%load channel_idx.mat

%moveset=ch_comb_batch(moveset,ch_idx);

cd D:\UPitt\academic\research\programming\neural_action_potential

save (moveset_file, 'moveset');

%training

%parameters for training set and sample set

ntrain=77;

ntest=10;

[A H W Q]=kalman_sysid(moveset_file,ntrain);

%training evaluation

mse_train=0;

corr_train=zeros(1,3);

for trainset=1:ntrain

    [ori, est]=kalman_decoding(moveset_file,trainset,0,A,H,W,Q,3);

    [temp1,temp2]=evalue(ori,est);

    mse_train=mse_train+temp1;

    corr_train=corr_train+temp2;

end

 mse_train= mse_train/ntrain;

 corr_train=corr_train/ntrain;
```

```
%test
draw=0;
mse=zeros(0);
corr=zeros(0,3);
%testset 78:87
for testset=ntrain+1:ntrain+ntest
    [ori, est]=kalman_decoding(moveset_file,testset,draw,A,H,W,Q,3);
    [mse(testset-ntrain),corr(testset-ntrain,:)]=evalue(ori,est);
end
ntest=size(mse,2);
ave_mse=sum(mse)/ntest;
ave_corr=sum(corr)/ntest;
clear draw ori est trainset testset ntest temp1 temp2 ntrain ntest
```

A.3 NEURAL SELECTION

A.3.1 Linear filter

```
clear;
%cell_select_linear_tb
SMOOTHING_WIDTH=19;%for decoding only
filter_length=21;%need to be consistent with the file to be selected
```

```
%parameter rank_threshold

rank_threshold=5;%[1;2;4;6;8;10;13;16;20;25;30;40;91];

Nexp=size(rank_threshold,1);

a_result=zeros(Nexp,11);

for expr=1:Nexp

%load full moveset with 0.05 lag

moveset=load ('moveset_acc_05.mat');%use kalman filter moveset with no window

moveset=moveset.moveset;

%cauculate the correlation coeff matrix between cells and states

cor_mv_fr=corr_mv_fr(moveset,0);%0 means won't plot the coeffs

abs_cor=abs(cor_mv_fr);%get absolute value of correlation

ranking=cell_sorting(abs_cor);%get the ranking for each state

%%%%%%%%%%%%%%%%%!!!!!%%%%%%%%%%%%

%ranking=ranking(:,1:3);%correlation with position

%ranking=ranking(:,4:6);%only get the correlation for velocity in 3D

%ranking=ranking(:,7:9);%corr for acc

cell_list=cell_select(ranking,rank_threshold(expr),'moveset_linear_21.mat');%use    linear    filter

moveset

%in cell_selection() the subset is stored in 'moveset_selected.mat'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%

%%%%%%%next, mapping the cell to channel, get the channel list
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
cell_ch=load('cell_channel.mat');
cell_ch=cell_ch.cell_ch;
n_cell=size(cell_list,1);
ch_temp=zeros(1,n_cell);
for m=1:n_cell
    ch_temp(m)=cell_ch(cell_list(m));
end
ch_temp=sort(ch_temp);
pt=ch_temp(1);
channel_list=ch_temp(1);
for m=2:n_cell
    if(ch_temp(m)~=pt)
        pt=ch_temp(m);
        channel_list=[channel_list;pt];
    end
end
clear cell_ch n_cell ch_temp m pt moveset

moveset_file='moveset_selected.mat';%where the selected moveset stores
%training
%parameters for training set and sample set
```

```
ntrain=77;
ntest=10;
filter=linear_training(moveset_file,ntrain,filter_length);
%training evaluation
mse_train=0;
corr_train=zeros(1,3);
for trainset=1:ntrain
    [ori, est]=linear_decoding(moveset_file,trainset,0,filter,SMOOTHING_WIDTH);
    [temp1,temp2]=evalue(ori,est);
    mse_train=mse_train+temp1;
    corr_train=corr_train+temp2;
end
 mse_train= mse_train/ntrain;
 corr_train=corr_train/ntrain;
%test
draw=0;
mse=zeros(0);
corr=zeros(0,3);
%testset 78:87
for testset=ntrain+1:ntrain+ntest
    [ori, est]=linear_decoding(moveset_file,testset,draw,filter,SMOOTHING_WIDTH);
    [mse(testset-ntrain),corr(testset-ntrain,:)]=evalue(ori,est);
end
```

```
ntest=size(mse,2);

ave_mse=sum(mse)/ntest;

ave_corr=sum(corr)/ntest;

result=[ave_mse,ave_corr,mse_train, corr_train];

clear draw ori est trainset testset ntest temp1 temp2 ntrain ntest

a_result(expr,:)=[length(cell_list),result,0,length(channel_list)];

end
```

A.3.2 Kalman filter

```
%cell_select_tb

%kalman filter testbench with 0.05 lag and neuron selection

%parameter rank_theshold

clear;

rank_threshold=[1;2;4;6;8;10;13;16;20;25;30;40;91];

Nexp=size(rank_threshold,1);

a_result=zeros(Nexp,11);

for expr=1:Nexp

%load full moveset with 0.05 lag

moveset=load ('moveset_acc_05.mat');

moveset=moveset.moveset;

%cauculate the correlation coeff matrix between cells and states

cor_mv_fr=corr_mv_fr(moveset,0);%0 means won't plot the coeffs
```

```
abs_cor=abs(cor_mv_fr);%get absolute value of correlation

ranking=cell_sorting(abs_cor);%get the ranking for each state

%%%%%%%%%%%%%%%%%!!!!!%%%%%%%%%%%%

ranking=ranking(:,1:9);

%ranking=ranking(:,1:3);%correlation with position

%ranking=ranking(:,4:6);%only get the correlation for velocity in 3D

%ranking=ranking(:,7:9);%corr for acc

cell_list=cell_select(ranking,rank_threshold(expr,:),'moveset_acc_05.mat');

%in cell_selection() the subset is stored in 'moveset_selected.mat'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%next, mapping the cell to channel, get the channel list

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%

cell_ch=load('cell_channel.mat');

cell_ch=cell_ch.cell_ch;

n_cell=size(cell_list,1);

ch_temp=zeros(1,n_cell);

for m=1:n_cell

    ch_temp(m)=cell_ch(cell_list(m));

end

ch_temp=sort(ch_temp);

pt=ch_temp(1);
```

```
channel_list=ch_temp(1);
for m=2:n_cell
    if(ch_temp(m)~=pt)
        pt=ch_temp(m);
        channel_list=[channel_list;pt];
    end
end
clear cell_ch n_cell ch_temp m pt

moveset_file='moveset_selected.mat';%where the selected moveset stores
%training
%parameters for training set and sample set
ntrain=77;
ntest=10;
[A H W Q]=kalman_sysid(moveset_file,ntrain);
%training evaluation
mse_train=0;
corr_train=zeros(1,3);
for trainset=1:ntrain
    [ori, est]=kalman_decoding(moveset_file,trainset,0,A,H,W,Q,3);
    [temp1,temp2]=evalue(ori,est);
    mse_train=mse_train+temp1;
    corr_train=corr_train+temp2;
```

```
end
 mse_train= mse_train/ntrain;
 corr_train=corr_train/ntrain;
%test
draw=0;
mse=zeros(0);
corr=zeros(0,3);
%testset 78:87
for testset=ntrain+1:ntrain+ntest
    [ori, est]=kalman_decoding(moveset_file,testset,draw,A,H,W,Q,3);
    [mse(testset-ntrain),corr(testset-ntrain,:)]=evalue(ori,est);
end
ntest=size(mse,2);
ave_mse=sum(mse)/ntest;
ave_corr=sum(corr)/ntest;
result=[ave_mse,ave_corr,mse_train, corr_train];
clear draw ori est trainset testset ntest temp1 temp2 ntrain ntest
a_result(expr,:)=[length(cell_list),result,0,length(channel_list)];
end
```

.

BIBLIOGRAPHY

Chae, M.Chen, K.Liu, W.Kim, J.Sivaprakasam, M. (2008). A 4-channel wearable wireless neural recording system. *Circuits and Systems* , 1760-1763.

Chestek, C.A.Gilja, V.Nuyujukian, P.Kier, R.Solzbacher, F.Ryu, S.I.Harrison, R.R.Shenoy, K.V. (2009). Hermes C: Low-power wireless neural recording system for freely moving primates. *IEEE Transactions On Neural Systems And Rehabilitation Engineering*.

Gay, D.Levis, P.Von Behren, R.Welsh, M.Brewer, E.Culler, D. (2003). The nesC language: A holistic approach to networked embedded systems. , p. 11. ACM.

Harris, J.G.Principe, J.C.Sanchez, J.C.Chen, D.She, C. (2008). Pulse-based signal compression for implanted neural recording systems. , pp. 344-347..

Harrison, R.R.Charles, C. (2003). A low-power low-noise CMOS amplifier for neural recording applications. *IEEE Journal Of Solid-State Circuits 38(6),* 958-965.

Harrison, R.R.Kier, R.J.Chestek, C.A.Gilja, V.Nuyujukian, P.Ryu, S.Greger, B.Solzbacher, F.Shenoy, K.V. (2009). Wireless Neural Recording With Single Low-Power Integrated Circuit. *IEEE Transactions On Neural Systems And Rehabilitation Engineering 17(4)*, 322-329.

Hatsopoulos, N.Joshi, J.O'Leary, J. (2004). Decoding continuous and discrete motor behaviors using motor and premotor cortical ensembles. *Journal Of Neurophysiology 92(2),* 1165-1174.

Hochberg, L.R.Serruya, M.D.Friehs, G.M.Mukand, J.A.Saleh, M.Caplan, A.H.Branner, A.Chen, D.Penn, R.D.Donoghue, J.P. (2006). Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *NATURE-LONDON- 442(7099),* 164.

Kemere, C.Shenoy, K.V.Meng, T.H. (2004). Model-based neural decoding of reaching movements: a maximum likelihood approach. *IEEE Transactions on Biomedical Engineering, 51(6),* 925-932.

Leuthardt, E.C.Schalk, G.Moran, D.Ojemann, J.G. (2006). The emerging world of motor neuroprosthetics: a neurosurgical perspective. *Neurosurgery 59(1)*, 1.

Levis, P.Madden, S.Polastre, J.Szewczyk, R.Whitehouse, K.Woo, A.Gay, D.Hill, J.Welsh, M.Brewer, E. (2005). Tinyos: An operating system for sensor networks. *Ambient Intelligence 35*.

Linderman, M.D.Santhanam, G.Kemere, C.T.Gilja, V.O'Driscoll, S.Yu, B.M.Afshar, A.Ryu, S.I.Shenoy, K.V.Meng, T.H. (2007). Signal Processing Challenges for Neural Prostheses. *Signal Processing Magazine, IEEE 25(1)*, 18-28.

Mavoori, J.Jackson, A.Diorio, C.Fetz, E. (2005). An autonomous implantable computer for neural recording and stimulation in unrestrained primates. *Journal Of Neuroscience Methods 148(1)*, 71-77.

Maynard, E.M.Nordhausen, C.T.Normann, R.A. (1997). The Utah Intracortical Electrode Array: A recording structure for potential brain-computer interfaces. *Electroencephalography and Clinical Neurophysiology 102(3)*, 228-239.

Moo, S.C.Zhi, Y.Yuce, M.R.Linh, H.Liu, W. (2009). A 128-Channel 6 mW Wireless Neural Recording IC With Spike Feature Extraction and UWB Transmitter. *IEEE Transactions on Neural Systems and Rehabilitation Engineering, 17(4)*, 312-321.

Reichert, W.M. (2007). Indwelling neural implants: strategies for contending with the in vivo environment. CRC.

Santhanam, G.Linderman, M.D.Gilja, V.Afshar, A.Ryu, S.I.Meng, T.H.Shenoy, K.V. (2007). HermesB: a continuous neural recording system for freely behaving primates. *IEEE Transactions On Biomedical Engineering 54(11)*, 2037-2050.

Sodagar, A.M.Wise, K.D.Najafi, K. (2007). A fully integrated mixed-signal neural processor for implantable multichannel cortical recording. *IEEE Transactions on Biomedical Engineering 54(6)*, 1075.

Taylor, D.M.Tillery, S.I.Schwartz, A.B. (2002). Direct cortical control of 3D neuroprosthetic devices. *Science 296(5574)*, 1829.

Warland, D.K.Reinagel, P.Meister, M. (1997). Decoding visual information from a population of retinal ganglion cells. *Journal Of Neurophysiology 78(5)*, 2336.

Wessberg, J.Stambaugh, C.R.Kralik, J.D.Beck, P.D.Laubach, M.Chapin, J.K.Kim, J.Biggs, S.J.Srinivasan, M.A.Nicolelis, M. (2000). Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature 408*, 361-365.

Wu, W.Black, M.J.Gao, Y.Bienenstock, E.Serruya, M.Shaikhouni, A.Donoghue, J.P. (2003). Neural decoding of cursor motion using a Kalman filter. *Advances in neural information processing systems* , 133-140.

Zumsteg, Z.S.Kemere, C.O Driscoll, S.Santhanam, G.Ahmed, R.E.Shenoy, K.V.Meng, T.H. (2005). Power feasibility of implantable digital spike sorting circuits for neural prosthetic systems. *IEEE Transactions on Neural Systems And Rehabilitation Engineering 13(3)*, 272-279.