SWITCHING HIDDEN-MARKOV MODEL AND HARDWARE IMPLEMENTATION FOR A BRAIN-MACHINE INTERFACE

By

SHALOM DARMANJIAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2005
Copyright 2005

by

Shalom Darmanjian
ACKNOWLEDGMENTS

When I was around 6 years old, I saw a television program showing a little girl being fitted with a robotic prosthetic. At that time, I thought she was controlling the arm with her mind. It inspired me to go to college and learn how to help others regain mobility with this ostensibly cyborg technology. As I grew older, I came to understand that no such technology existed in the world and realized that the girl must have been controlling the arm through muscle contractions. My childhood dream was washed away in much the same way that most children who aspire to become astronauts have their dream fade away.

Then one day I made a left instead of a right. I had the choice of going to class in Larson Hall (right side) or asking Dr. Gugel for a letter of recommendation to grad school in Benton Hall (left side). By shear chance, he was in his office and able to talk without student interruptions. That simple letter turned into a meeting with Dr. Nechyba, then a meeting with Dr. Principe and finally an introduction to the Brain Machine Interface (BMI) project. All three professors have given me more than I can ever repay and I am truly grateful for the wonderful opportunities. I am also grateful to my fellow Applied Digital Design Lab (ADDL), Computational Neural Engineering Lab (CNEL), and Machine Intelligence Lab (MIL) comrades from whom I have also learned a great deal. Scott Morrison, Jeremy Parks and Joel Fuster welcomed me into the ADDL lab and put up with my sarcasm. In particular, Scott provided guidance and expertise with all of the hardware that I helped to create. Additionally, Phil Sung Kim greatly enhanced the work in this thesis with his work on the Least Mean Square (LMS) and wiener filters for use in the Bi-Modal structure (discussed in this thesis). Greg and Ben were also encouraging and helpful during the process of making this thesis and the final hardware work. There are many MIL members that I have become friends with. I appreciate them all. Finally, I would like to give special thanks to Jeremy Anderson and
Sinisa Todorvich for their very helpful criticisms of the thesis. I truly am nothing more
than the contributions from the professors and the students of ADDL, CNEL, MIL, all
wrapped into one lovable fuzzy ball of a fella. I personally thank them all for letting me
become an astronaut.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Brain Machine Interface: Collaborative Effort</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Overview</td>
<td>4</td>
</tr>
<tr>
<td>2 EXPERIMENTAL SETUP AND INITIAL ANALYSIS</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Duke-Primate Behavioral Experiments</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Duke-Data Acquisition</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Elementary Statistical Classifier</td>
<td>9</td>
</tr>
<tr>
<td>2.3.1 Basic Feature Extraction</td>
<td>9</td>
</tr>
<tr>
<td>2.3.2 Basic Structure</td>
<td>12</td>
</tr>
<tr>
<td>2.3.3 Results</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Discussion</td>
<td>15</td>
</tr>
<tr>
<td>3 HIDDEN MARKOV MODELS</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Motivation</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Hidden Markov Model Overview</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Vector Quantizing-HMM</td>
<td>18</td>
</tr>
<tr>
<td>3.3.1 Structure</td>
<td>18</td>
</tr>
<tr>
<td>3.3.2 Training</td>
<td>20</td>
</tr>
<tr>
<td>3.3.3 Results</td>
<td>23</td>
</tr>
<tr>
<td>3.4 Factorial Hidden Markov Model</td>
<td>25</td>
</tr>
<tr>
<td>3.4.1 Motivation</td>
<td>25</td>
</tr>
<tr>
<td>3.4.2 Structure</td>
<td>28</td>
</tr>
<tr>
<td>3.4.3 Training</td>
<td>31</td>
</tr>
<tr>
<td>3.4.4 Results</td>
<td>32</td>
</tr>
<tr>
<td>3.5 Discussion</td>
<td>35</td>
</tr>
<tr>
<td>4 LOCAL LINEAR MODELS WITH HMM PARTITIONING</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Structure</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Results</td>
<td>38</td>
</tr>
<tr>
<td>4.3 Conclusion</td>
<td>40</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>BMI concepts drawing</td>
<td>2</td>
</tr>
<tr>
<td>1–2</td>
<td>UF-BMI overview</td>
<td>3</td>
</tr>
<tr>
<td>2–1</td>
<td>Rhesus and owl monkeys</td>
<td>5</td>
</tr>
<tr>
<td>2–2</td>
<td>Feeding experiment</td>
<td>6</td>
</tr>
<tr>
<td>2–3</td>
<td>Neural prosthetics and operations</td>
<td>7</td>
</tr>
<tr>
<td>2–4</td>
<td>Example neural waveforms</td>
<td>8</td>
</tr>
<tr>
<td>2–5</td>
<td>Optotak</td>
<td>8</td>
</tr>
<tr>
<td>2–6</td>
<td>N.M.A.P. system</td>
<td>9</td>
</tr>
<tr>
<td>2–7</td>
<td>Binned neural data and corresponding velocity (with threshold)</td>
<td>10</td>
</tr>
<tr>
<td>2–8</td>
<td>Instantaneous velocity</td>
<td>11</td>
</tr>
<tr>
<td>2–9</td>
<td>Velocity vs displacement</td>
<td>12</td>
</tr>
<tr>
<td>2–10</td>
<td>Sample spatial statistics (one-second sliding window)</td>
<td>13</td>
</tr>
<tr>
<td>2–11</td>
<td>Sample thresholded spatial statistics (one-second sliding window)</td>
<td>14</td>
</tr>
<tr>
<td>2–12</td>
<td>Performance of threshold-based classifier (spatial variance, all neurons)</td>
<td>15</td>
</tr>
<tr>
<td>3–1</td>
<td>Discrete HMM chain</td>
<td>18</td>
</tr>
<tr>
<td>3–2</td>
<td>Stationary/moving classifiers</td>
<td>19</td>
</tr>
<tr>
<td>3–3</td>
<td>LGB VQ algorithm on 2D synthetic data</td>
<td>21</td>
</tr>
<tr>
<td>3–4</td>
<td>&quot;Leaving-k out&quot; testing</td>
<td>24</td>
</tr>
<tr>
<td>3–5</td>
<td>Sequential testing</td>
<td>25</td>
</tr>
<tr>
<td>3–6</td>
<td>Single neural channels</td>
<td>27</td>
</tr>
<tr>
<td>3–7</td>
<td>Average ratios</td>
<td>27</td>
</tr>
<tr>
<td>3–8</td>
<td>Before and after log</td>
<td>29</td>
</tr>
<tr>
<td>3–9</td>
<td>General FHMM</td>
<td>29</td>
</tr>
<tr>
<td>3–10</td>
<td>Comparison of our FHMM to general model</td>
<td>30</td>
</tr>
</tbody>
</table>
Abstract of Thesis Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Master of Science

SWITCHING HIDDEN-MARKOV MODEL AND HARDWARE IMPLEMENTATION FOR A BRAIN-MACHINE INTERFACE

By
Shalom Darmanjian
May 2005

Chair: Jose Principe 
Major Department: Electrical and Computer Engineering

In pursuit of the University of Florida’s Brain Machine Interface (BMI) goals, this thesis focused on how to improve the 3D modeling of a primate’s arm trajectory and the implementation of such algorithms. In terms of 3D modeling, we argue that the best way to improve the trajectory prediction is by first using a switching classifier to partition the primate’s neural firings and corresponding arm movements into different motion primitives. We show that by switching or delegating these isolated neural/trajectory data to different local linear models, prediction of final 3-D arm trajectories is markedly improved. Although our study focused on primitives of motion and non-motion, we propose that our work can expand to include more primitives and thus increase the final performance manifold.

Concerning implementation of BMI algorithms, our first aim was to achieve a portable wireless computational DSP. Next we determined the software and hardware component layers integrated in this evolving design. Finally, we detail the distributed implementation of the switching model over a parallel computing architecture, investigating offline training and evaluation as well as possible future real-time implementations.
CHAPTER 1
INTRODUCTION

1.1 Motivation

The Olympics are not the only place to witness masterful control of muscles and limbs. Simply attending a local ping-pong match or a basketball game can also provide examples of precise dexterity and bodily control. Whether we are Olympiads or regular people, we all require the ability to plan trajectories and execute accurate motions to serve the needs and desires of our daily lives.

To achieve this precision, we rely on the brain to decode our thoughts and control many individual components in our bodies without specifically commanding it to do so. Along with the brain, the spinal cord and peripheral nerves make up a complex, integrated information-processing and control system. Specifically, they help the brain control blood flow, oxygen intake, blood pressure, and a myriad of other functions that help us grow and stay healthy. The brain must continuously do all of these actions while processing feedback from visual centers, tactile sensors, and many other internal systems; and provide higher-order cognition for thesis writing.

Unfortunately, millions of people’s brains suffer a partial or full disconnect from their bodies, hindering the control of physical movement. Although some technological solutions give these individuals various interaction with the external world, they often fall short of what is required for daily life. The noted physicist, Stephen Hawking, is an example of a human possessing an exceptional mind with little or no control of his exterior limbs. He must navigate daily life with a simple computer joystick that often limits his ability to present his theories to peers and conference proceedings. This crude device and others like it can only help select disabled individuals who still possess some motor control. If a communication pathway could bridge the gap between the brain and
the external world, it would empower millions of disabled people who now have little or no motor control. This is the idealistic vision of a Brain Machine Interface (BMI).

A Brain Machine Interface is one of many ways that researchers are trying to develop this pathway between man and machine. Specifically, a BMI is a system that can directly retrieve neuronal firing patterns from dozens to hundreds of neurons and then translate this information into desired actions in the external world. The functionality is similar to how the brain currently works but provides a bypass between the patient’s brain and an external device. A BMI is not limited to only patients with paralysis in their extremities; future applications could serve military or commercial sectors. Although progress has been made toward these goals since the 1970s [1], work must be accomplished before machines (under brain control) can hit a simple baseball as well as a smiling 8 year old child.

1.2 Brain Machine Interface: Collaborative Effort

To design a complicated system like a BMI, experts must come together from multiple fields to solve the many technological and biological problems. To circumvent these barriers, the Defense Advanced Research Projects Agency (DARPA) brought
biologists, neurologists and a broad spectrum of engineers to collaborate under a single umbrella of leadership and finance. Dr. Miguel Nicolelis and his staff at Duke University are undertaking this leadership role for the entire project by coordinating the collaboration of all member institutions. His group also provides neurological expertise and an experimental primate-testing platform, which is required before implanting the BMI into humans. MIT, SUNY and UF are the other institutions providing their expertise to complement the skills at Duke.

Specifically, our group at UF is involved in developing algorithms that can predict a primate’s arm trajectories based on the spatial sampling of hundreds of neurons within the multiple cortical and subcortical areas. In turn, these models must be multiple-input multiple-output systems of huge dimensionality to accommodate all of these data at once. In designing such a structure, accuracy must be tempered with speed, especially when implementing these models in hardware. Some of our group’s initial experiments examined linear and nonlinear models to determine which models are the most efficient with regard to the constraints of accuracy and speed in our application [2].

Figure 1–2: UF-BMI overview
In tandem with algorithm development, our group is investigating the use of hybrid analog and digital technologies to achieve low-power portable devices that will run these models. Consequently, we are evolving the digital and analog designs to evaluate their feasibility as we shrink and integrate them into a single system. Our group’s Very Large Scale Integration (VLSI) expertise can assist in this type of system by designing custom low-power hybrid (VLSI-DSP) chips. Eventually though, all processing needs to move into a single-chip solution so that it can be implanted into a patient’s body and can independently control an external device with actionable thoughts.

1.3 Overview

In service of UF’s BMI goals, this thesis focuses on how to improve the 3-D modeling of a primate’s arm trajectory and the implementation of such algorithms. In terms of 3D modeling, we argue that the best way to improve the trajectory prediction is by first using a ‘switching’ classifier to partition the primate’s neural firings and corresponding arm movements into different motion primitives. We show that by switching or delegating these isolated neural/trajectory data to different local linear models, prediction of final 3D arm trajectories is markedly improved. Although this thesis focuses on primitives of motion and non-motion, we propose that our work can expand to include more primitives and subsequently increase the final performance manifold.

Concerning implementation of BMI algorithms, we first discuss our initial step in trying to achieve a portable wireless computational DSP. Secondly, we describe the software and hardware component layers integrated within this evolving design. Finally, we detail the distributed implementation of the switching model over a parallel computing architecture, discussing the results in offline training/evaluation as well as possible future real-time implementations.
CHAPTER 2
EXPERIMENTAL SETUP AND INITIAL ANALYSIS

In this chapter, we first depict the experimental environment in which the non-human primates carried out their behavioral tasks. Secondly, we describe the retrieval of neurological and trajectory data along with certain properties they exhibit. Finally, we discuss our elementary analysis of this data and present the results of a rudimentary classifier.

2.1 Duke-Primate Behavioral Experiments

Dr. Nicolelis’s laboratory, at the Duke University Medical center, is responsible for carrying out the behavioral experiments with primate subjects for the DARPA funded BMI project. The primate species they use within these experiments are the Rhesus Macaque (Macaca mulatta) and the Owl Monkey (Aotus trivirgatus), each having varying physical characteristics. Different exemplars of these small species train on a multitude of experiments like 3-D food grasping, 1-D pole control, 2-D pole control and 2-D pole control plus gripping. For the purposes of this thesis, we focus on 3-D food grasping experiments that involve a female Owl Monkey.

![Figure 2–1: Rhesus and owl monkeys](image-url)
In this particular 3-D food grasping experiment, once an opaque barrier lifts, the female monkey is required to retrieve fruit from four fixed tray locations and then place it in her mouth. In order to constrain the feeding movements, Dr. Nicolelis’s group, use a constraining apparatus to hold the primate’s body in place. The neck, torso, left arm and legs fasten into a clamp-like gripper so that the only motion that can take place is right arm movement. In turn, the right arm movement is digitized and transmitted to the prediction models along with the neural data and subsequently to a robotic arm [21, 25]. This type of experiment is important for two reasons. First, it mimics the behavior that a human would require in daily life, and second, the experiment is cyclic so that the prediction models can have similar data examples with which to train and evaluate.

![Figure 2–2: Feeding experiment](image)

### 2.2 Duke-Data Acquisition

The acquisition of data falls within two areas: the neurological data and the physical arm trajectory data. Within these two areas, we explain how the data are acquired and the characteristics they display.

To retrieve neurological data, Dr. Nicolelis and his staff first drill craniotomies into the primates. For each primate subject, his team then places five to ten different cortical implants in multiple cortical regions. Specifically, they implant different arrays in the
posterior parietal cortex (PP), primary motor cortex (M1), and dorsal pre-motor cortex (PMD) since each region has been found to contribute to motor control [3]. The micro-arrays placed in these regions consist of traditional metal microelectrodes [3] that are electrolytically sharpened wires (pins), 25 to 50 um in diameter and insulated to define an exposed recording area at the tip of around 100 um. On average, each microelectrode records from four neurons, consequently requiring signal conditioning and spike sorting of the waveforms at a later stage in order to distinguish single neural cells [19].

To record the primate’s arm trajectory, Dr. Nicholelis’s group employed a non-invasive recording system to accurately measure the position of the monkey’s arm in 3-D space. This commercial product, Optotrak, is accurate to .1 mm with a resolution of .01 mm when recording real-time 3-D positions. The system employs three cameras that track infrared markers placed on the monkey’s wrist. By using different marker
combinations, the Duke team can isolate and track different parts of the arm as it moves through space.

As both data streams are simultaneously recorded from the monkey, it is necessary to combine them in order for the mapping models to train and evaluate the trajectory. This can be a difficult task since both types of data have different properties associated with them. For example, the neural signals come in the form of raw electrode voltage potentials that range from the noise level [3](20 V or so) to about 1 mV. The signals
can range in frequency from 30 Hz to 9 KHz depending on the spiking rate and activity. Whereas the Optotrak system generates digitized 200 Hz signals to define the tracking of the monkey’s 3-D arm trajectory. In order for this processing to be useful for the models, the neural data are binned into 100 ms bins and the trajectory data are down-sampled by 20 to generate a corresponding 10 Hz signal, which is consistent with other investigators [12, 17, 19].

Figure 2–6: N.M.A.P. system

2.3 Elementary Statistical Classifier

2.3.1 Basic Feature Extraction

Among neural scientists, there is an on-going, unresolved debate regarding how the motor cortex encodes the arms movement [12]. Various researchers argue that the motor cortex encodes the arms velocity, position or both within the neuronal firings [12, 17]. In an attempt to observe any obvious patterns or correlations in this encoding, we began with a visual and numerical inspection of the velocity-neural data. In viewing the data, we were able to discern some basic properties. First, there appeared to be a slight increase in binned firing counts when the hand moved at noticeable velocities. Second, some of the 104 neurons rarely fire, while others virtually fire all the time; but interestingly, we witnessed that specific neurons increased their firing rate at certain points along the arm trajectory [18].
To complement our visual analysis of identifiable correlations, we also computed the cross correlation with up to a one second shift in time (forward and backward) between individual neural channels and hand velocities. Overall, visual and correlation analysis showed that patterns existed between neural firings and hand movement [18].

However, this method is limited in that only a few obvious (visually discernable) patterns were perceivable with the velocity-neural data. This lack of correspondences complicated our training and evaluation of the switching model, since we needed to have clearly definable and separate classes.

Therefore, we wanted a more objective approach to define the segmented classes before training and evaluating our model. To avoid exclusion of potential neural encodings, we generated the data classes for two different segmented data sets, one based on velocity, the other based on displacement.

Figure 2–7: Binned neural data and corresponding velocity (with threshold)

For the velocity hypothesis, ideally, the first class of neural firings should contain data where the arm appears to be stationary, while the second class should contain data where the monkey’s arm appears to be moving. We used a simple threshold to achieve
this grouping: if the instantaneous velocity of the arm is below the noise threshold of the sensor (determined by inspecting the velocity data visually), the corresponding neural data were classified as stationary; otherwise, the neural data were classified as moving. In Figure 2-8, we plot the instantaneous velocity of the monkey’s arm for a 500 second segment of the data, where the monkey is repeatedly performing a reaching task. Based on this plot, we chose 4 mm/sec as the noise threshold for the above procedure.

![Instantaneous velocity](image)

**Figure 2–8: Instantaneous velocity**

For the position hypothesis, we wanted to classify the monkey’s arm movements based on displacement. To demonstrate this concept, we plot a sample feeding session for the monkey (Figure 2-9). The three colored trajectories represent displacement along the Cartesian coordinates, as the monkey is moving its arm from rest to the food tray, from the food tray to its mouth and, back to the rest position. Figure 2-9B indicates the segmentation of this data into two distinct displacement classes: rest and active, which are analogous to the stationary and moving classes in the velocity-based segmentation above.

In Figure 2-9C and 2-9D, we plot the velocity of the trajectories from figures 2-9A and 2-9B, here we see that this segmentation is not the same as the velocity-based segmentation. Note from the dotted line (indicating the velocity threshold previously described) that some of the active class in the displacement-based segmentation are classified as stationary in the velocity-based segmentation. Now that we understood
what our segmented classes should represent, we wanted to apply simple statistical methods to evaluate their performance.

![Figure 2–9: Velocity vs displacement](image)

**Figure 2–9: Velocity vs displacement**

### 2.3.2 Basic Structure

In the next sequence of experiments, we computed the mean and variance for the neural spike data, temporally across individual neural channels, as well as spatially across all neural channels. Due to the relatively sparse nature of the neural data, we computed the statistics over 50% sliding windows of one- and four-second lengths. With a statistical description of the neural data, we proceeded to test if the aggregate quantities could tell us anything about the corresponding hand movement. In this initial analysis, we set out to distinguish hand movement from non-movement by applying thresholds to the computed statistics. If a particular statistical indicator for a given time index was below a corresponding threshold, we set the predicted hand motion to stationary. While indicator values above that same threshold were labeled as moving. For each time index, we also labeled the one-dimensional velocity data as stationary or moving. We then checked the accuracy of this simple movement predictor. To compensate for possible misalignment between the neural spike data and the hand
movement data, we repeated the same procedure for data shifted by a maximum of two time indices both forward and backward in time. Finally, the entire analysis above was repeated for a subset of neurons that appeared most relevant for hand-movement prediction, as indicated by the weights in previously trained recurrent neural networks [18].

2.3.3 Results

With the rudimentary analysis (Section 2.3.2), we computed four basic quantities:

- Temporal mean per neural channel.
- Temporal variance per neural channel.
- Spatial mean across neural channels.
- Spatial variance across neural channels.

Of these computed quantities, the spatial statistics appeared to be the most useful predictive indicator of hand movement. Figure 2-10 shows a sample of the computed spatial statistics for all neurons as well as for a subset of neurons (determined from the recurrent neural network models).

![Sample spatial statistics](image)

Figure 2–10: Sample spatial statistics (one-second sliding window).
Given the statistical data (Figure 2-10), we proceeded to develop a threshold-based classifier to discriminate between hand movement and non-movement. From Figure 2-10, we observe that spatial variances appear to be a better predictor of hand movement than spatial means. Therefore, we designed our classifier to rely on spatial variances, rather than spatial means. We applied two different variance thresholds: the first was chosen as the mean of the variances, while the second was somewhat lower to reduce the number of hand movement (moving class) misclassifications.

![Figure 2–11: Sample thresholded spatial statistics (one-second sliding window).](image)

To test the classifier, we used 4,000 data samples, corresponding to 201 distinct instances for statistics computed over four seconds, and 801 for statistics computed over one second. In this data sample, 58 out of 201 instances corresponded to significant hand movement for the four-second statistics, while 125 out of 801 instances corresponded to significant hand movement for the one-second statistics. Figure 2-12 summarizes the performance of this spatial-variance based classification scheme, while Figure 2-11 illustrates the effect of a threshold on the spatial statistics in Figure 2-10. Note from both Figure 2-12 and Figure 2-11 that even this simple classification scheme is able to achieve successful discrimination of hand movement from non-movement. Results for
neuron subsets selected from trained recurrent neural networks proved to be similar to those in Figure 2-12.

<table>
<thead>
<tr>
<th>four-second statistics \ (γ = 0.500)</th>
<th>stationary hand \ (% correct)</th>
<th>moving hand \ (% correct)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>57.0% (81/142)</td>
<td>84.8% (49/58)</td>
</tr>
<tr>
<td>four-second statistics \ (γ = 0.557)</td>
<td>85.9% (122/142)</td>
<td>65.5% (38/58)</td>
</tr>
<tr>
<td>one-second statistics \ (γ = 0.500)</td>
<td>57.4% (388/676)</td>
<td>90.4% (113/125)</td>
</tr>
<tr>
<td>one-second statistics \ (γ = 0.665)</td>
<td>92.9% (628/676)</td>
<td>62.4% (78/125)</td>
</tr>
</tbody>
</table>

Figure 2–12: Performance of threshold-based classifier (spatial variance, all neurons)

2.4 Discussion

The rudimentary classifier and analysis discussed in this chapter had two goals. First, we sought to familiarize ourselves with the neural firing and trajectory data to better understand the intricacies of the primate experiment. Second, we explored whether or not even simple statistical analysis can yield useful insights into this problem. To do this, we developed a threshold-based classifier of hand movement vs. non-movement that relied exclusively on a single statistical indicator – namely, spatial variance in the neural firing data.

However, the question remains as to which type of segmentation (velocity or displacement) is likely to be more biologically plausible and, consequently easier to learn. While we will defer our thoughts on this question until Chapter 3, we do note that keeping an arm stationary (1) at rest, or (2) in extension requires different muscle actions. In the first case, muscles can be relaxed, while in the second case, at least some muscles must be tensed.

Despite the simplicity of this approach, we nevertheless were able to achieve surprisingly good results in classification performance (see Figure 2-12) of the two simple
primitives of motion and non-motion. Encouraged by these results, we proceeded to develop an advanced classifier that relies on more sophisticated trainable statistical models. We describe that work in the next chapter.
CHAPTER 3
HIDDEN MARKOV MODELS

3.1 Motivation

We trained statistical models corresponding to the two classes of data discussed in Chapter 2. Based on previous statistical work [18], we feel that these statistical models should capture the temporal statistical properties of neural firings that characterize the monkey’s arm movement or lack thereof. One such statistical model, the Hidden Markov Model (HMM), enforces only weak prior assumptions about the underlying statistical properties of the data, and can encode relevant temporal properties. For these important reasons, we choose to model the two classes of neural data (stationary vs. moving) with HMMs. This choice follows a long line of research that has applied HMMs in the analysis of stochastic signals, such as in speech recognition [4, 5] modeling open-loop human actions [13], and analyzing similarity between human control strategies [13].

3.2 Hidden Markov Model Overview

Although continuous and semi-continuous HMMs have been developed, discrete-output HMMs are often preferred in practice because of their relative computational simplicity and reduced sensitivity to initial parameter settings during training [16]. A discrete Hidden Markov Chain (Figure 3-1) consists of a set of $n$ states, interconnected through probabilistic transitions, and is completely defined by the triplet, $\lambda = \{A, B, \pi\}$, where $A$ is the probabilistic $n \times n$ state transition matrix, $B$ is the $L \times n$ output probability matrix (with $L$ discrete output symbols), and $\pi$ is the $n$-length initial state probability distribution vector [14, 16]. For an observation sequence $O$, we locally maximize $P(\lambda|O)$ (i.e. probability of model given observation sequence $O$) with the Baum-Welch Expectation-Maximization (EM) algorithm.

The discrete HMMs discussed in this thesis, are trained on finite-length sequences, so that rare events with nonzero probability may be possible yet, at the same time,
may not be reflected in the data (i.e. may not have been observed). The probabilities corresponding to such events will therefore converge to zero during HMM training. Alternatively, a sample sequence may have erroneous readings due to sensor failure, etc., and such sequences will evaluate to zero probability on HMMs previously trained on less noisy data. In order to train discrete-output HMMs on continuous-valued data effectively, we use discretization compensation, namely, semi-continuous evaluation.

In semi-continuous evaluation, the HMM is first trained on discrete data (vector quantized –VQ– from real-valued data). When new sequences of real-valued data need to be evaluated, we assume that the VQ codebook previously generated represents a mixture of Gaussians with some uniform variance \( \alpha \) that can be thought of as a smoothing parameter. Below, we first discuss the overall structure of this VQ-HMM and then detail the training of the classifier in section 3.2.2.

3.3 Vector Quantizing-HMM

3.3.1 Structure

In this section, we broadly describe our VQ-HMM-based classifier (Figure 3-2 illustrates the overall structure). The classifier works as follows:

1). At time index \( t \), we convert a neural firing vector \( v_t \) of length 104 (equal to number of neural channels) into a discrete symbol \( O_t \) in preparation for discrete output HMM
evaluation. The method of signal-to-symbol conversion will be discussed later in this section.

2). Next, we evaluate the conditional probabilities \( P(O|\lambda_s) \) and \( P(O|\lambda_m) \), where,

\[
O = \{O_{t-N+1}, O_{t-N+2}, O_{t-1}, O_t\}, N > 1,
\]

and \( \lambda_s \) and \( \lambda_m \) denote HMMs that correspond to the two possible states of the monkey's arm (stationary vs. moving).

3). Finally, we decide that the monkey's arm is stationary if,

\[
P(O|\lambda_s) > P(O|\lambda_m),
\]

and is moving if,

\[
P(O|\lambda_m) > P(O|\lambda_s).
\]

In order to explicitly compute \( P(O|\lambda) \), we use the practical and efficient forward algorithm [16]. For the forward algorithm, let us define a forward variable \( \alpha_t(i) \):

\[
\alpha_t(i) = P(O_1, ..., O_t, X_i|\lambda),
\]
which refers to probability of the partial observation sequence \( \{ O_1, ..., O_t \} \) and being in state \( X_i \) at time \( t \), given the model \( \lambda \) [16]. As explained by Rabner [16] and others [14] the \( \alpha_t \) variables can be computed inductively with the use of the probabilistic transition and output matrixes, in turn, we evaluate \( P(O|\lambda) \) with:

\[
P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).
\]  

(3.5)

Furthermore, the classification decision in Equations 3.2 and 3.3 is relatively simplistic in that it does not optimize for overall classification performance, and does not account for possible desirable performance metrics. For example, it may be very important for an eventual modeling scheme to err on the side of predicting arm motion (i.e. moving class). Therefore, we modify our previous classification decision to include the following classification boundary:

\[
\frac{P(O|\lambda_m)}{P(O|\lambda_s)} = y,
\]

(3.6)

where \( y \) now no longer has to be strictly equal to one.

Note that by varying the value of \( y \), we can essentially tune classification performance to fit our particular requirements for such a classifier. Moreover, optimization of the classifier is now no longer a function of the individual HMM evaluation probabilities, but rather a function of overall classification performance.

In the following subsection, we discuss signal-to-symbol conversion and HMM training in somewhat greater detail.

3.3.2 Training

Our particular dataset contained 23000 discrete binned firing counts for the 104 neurons (each binned count corresponds to the number of firings per 100ms). As discussed, we must first convert this multi-dimensional neural spike data to a sequence of discrete symbols. This process involves vector quantizing the input-space vectors to
discrete symbols in order to use discrete -output Hidden Markov Models. We choose the well-known LBG VQ algorithm [6], which iteratively generates vector codebooks of size $L = 2^m$, and can be stopped at an appropriate level of discretization, as determined by the amount of available data. By optimizing the vector codebook on the neural spike data, we seek to minimize the amount of distortion introduced by the vector quantization process. Figure 2-3 illustrates the LBG VQ algorithm on some synthetic, two-dimensional data (gray area).

![Figure 3–3: LGB VQ algorithm on 2D synthetic data](image)

After vector quantizing the input, we use the generated discrete symbols as input to a left-to-right (or Bakis) HMM chain; in this structure non-zero probability transitions between states are only allowed from left to right, as depicted in the HMMs in Figure 6. Given that we expect the monkey’s arm movement to be dependent not only on current neural firings, but also on a recent time history of firings, we train each of the HMM models on observation sequences of length $N$. Since the neural spike data used in this study is binned at 100 msec, $N = 10$, for example, would correspond to neural spike data over the past one second (Equation 3.1). During run-time evaluation of $P(O|\lambda_s)$ and $P(O|\lambda_m)$, we use the same value of $N$ as was used during training.
In order to maximize the probability of the observation sequence \( O \) we must estimate the model parameters \((A, B, \pi)\) for both \( \lambda_M \) and \( \lambda_S \). This is a difficult task; first, there is no known way to analytically solve for the parameters that will maximize the probability of the observation sequence [16]. Second, even with a finite amount of observation sequences there is no optimal way to estimate these parameters [16]. In order to circumvent this issue, we use the iterative Baum-Welch method to choose \( \lambda = \{A, B, \pi\} \) that will locally maximize \( P(O|\lambda) \) [14].

Specifically, for the Baum-Welch method we provide a current estimate of the HMM \( \lambda = \{A, B, \pi\} \) and an observation sequence \( O = \{O_1, ..., O_T\} \) to produce a new estimate of the HMM given by \( \tilde{\lambda} = \{\tilde{A}, \tilde{B}, \tilde{\pi}\} \), where the elements of the transition matrix \( A \),

\[
\tilde{a}_{ij} = \frac{\sum_{t=1}^{T-1} \zeta_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, i, j \in \{1, ..., N\}. \tag{3.7}
\]

Similarly, the elements for the output probability matrix \( B \),

\[
\tilde{b}_j(k) = \frac{\sum_t \gamma_t(j) (\text{where } \forall O_t = v_k)}{\sum_t \gamma_t(j)}, j \in \{1, ..., N\}, k \in \{1, ..., L\}, \tag{3.8}
\]

and finally the \( \pi \) vector,

\[
\tilde{\pi}_i = \gamma_1(i), i \in \{1, ..., N\}, \tag{3.9}
\]

where,

\[
\zeta_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \quad \text{and} \quad \tag{3.10}
\]

\[
\gamma_t(i) = \sum_{j=1}^{N} \zeta_t(i, j). \tag{3.11}
\]

Please note, \( \beta \) is the backward variable, which is similar to the forward variable \( \alpha \) except that now we propagate the values back from the end of the observation sequence, rather than forward from the beginning of \( O \) [14].
3.3.3 Results

Given our classifier structure in Figure 4 and our decision rule in Equation 3.4, there are a number of design parameters that can be varied to optimize classification performance:

\[ L = \text{number of prototype vectors in VQ codebook}; \]
\[ N = \text{length of observation sequences}; \]
\[ n = \text{number of states for the HMM}; \]
\[ y = \text{classifier threshold boundary}; \]

In Tables 2 and 3, we report experimental results for different combinations of the four parameters and subsets of neural channels. These tables are a small representation of the classification results produced from a large number of conducted experiments. The L parameter (no. of prototype vectors) was varied from 8 to 256; the N parameter (observation length) was varied from 5 to 10; and the n parameter (no. of states) was varied from 2 to 8.

The two tables differ in how the data was split into training and test sets. In the 'leaving-k-out' approach (Figure 3-4), we took random segments of the complete data, removed them from the training data, and reserved them for testing; care was taken that no overlap occurred between the training and test data. In the second approach (Table 3), we split the data sequentially into training and test data in equivalent fashion to our group members at UF [2]. The advantage of the first testing approach is that we can repeat the procedure an arbitrary number of times, leading to more test data, and hence, more statistically significant results. Alternatively, the advantage of the second testing approach is that it uses test data in a manner more likely to be encountered in an eventual BMI system, where a period of training would be followed by a subsequent period of testing.
At this point, we make some general observations about the results in Figure 3-4 and 3-5. First, the displacement-based segmentation results are substantially better than the velocity-based segmentation results. Second, we note that the results in Figure 3-4 are marginally better than those in Figure 3-5. We suggest that one reason for this is that the neural encoding of the small population of 104 neurons is non-stationary to some extent. If the data is non-stationary, we should expect the first testing approach to produce better results, since test data in the 'leaving-k-out' approach is taken from within the complete data set, while the second testing approach takes the test data from the tail end of the complete data set. Overall, we note that the sequential testing with displacement-based data is probably the best. We also note that since a subset of neural channels at the input yielded the best performance, some of the measured neural channels may offer little information about the monkey’s arm movement which subsequently directs the motivation in the next section.

![Table of data](image)

Figure 3–4: ”Leaving-k out” testing
3.4 Factorial Hidden Markov Model

3.4.1 Motivation

As mentioned in section 3.3.3, our previous classifier required the conversion of the multi-dimensional neural data into a discrete symbol for the discrete-output HMMs [7]. We used the LBG-VQ algorithm, since it has the ability to generate this discrete symbol with a relatively minimal amount of distortion. Unfortunately, this 'minimal' distortion was later revealed to hamper classification performance when used with the neural data [7]. We note that since the 104-channel data does not form tight clusters in the 104-dimensional input space, the VQ signal-to-symbol conversion introduces a substantial loss of information and consequently degrades classification performance [6]. Combining this result with the error that occurs from the linear models (within the bi-model mapping framework), we are only able to produce neural-displacement mapping results marginally better than our group’s previous work [2].
As discussed, we attempted to circumvent these VQHMM limitations by exploring different neural subsets to see if we could eliminate noisy unimportant neurons and retain useful ones. To differentiate important neurons from unimportant neurons, we examined how well an individual neuron can classify movement vs. non-movement when trained and tested on an individual HMM chain. We are able to directly train a single HMM chain since the neural data is already in discrete form, ranging in value from zero to twenty (firings per 100ms bin).

During the evaluation of these particular HMMs, we compute the conditional probabilities $P(O^{(k)}|\lambda_s^{(k)})$ and $P(O^{(k)}|\lambda_m^{(k)})$ for each neural channel $k$ with its respective observation sequence of binned firing counts. To give a qualitative understanding of these weak classifiers, we present in Figure 3-6 the probabilistic ratios from the top 14 single-channel HMM classifiers (shown between the top and bottom movement segmentations). Specifically, we present the probabilistic ratio

$$P(O^{(k)}|\lambda_m^{(k)}) \over P(O^{(k)}|\lambda_s^{(k)})$$

for each neural channel in a grayscale gradient format; darker bands represent ratios larger than one (indicating a stronger probability toward movement) and lighter bands for ratios smaller than one (indicating stronger probability toward non-movement). The probabilities roughly equal to one another show up as grey bands. We glean from this figure that the group of single-channel HMMs can roughly predict movement and non-movement from the data.

In order to observe the relevance of these single HMM chains further, we compute the average of the probabilistic ratios

$$\frac{1}{K} \sum_{k=1}^{K} P(O^{(k)}|\lambda_M^{(k)}) \over P(O^{(k)}|\lambda_S^{(k)})$$

for a given observation sequence. Figure 4 presents the average of the ratios (light grey), as well as the variance (dark grey) of the single-channel HMM output probabilities.
We also superimpose our segmentations of movement and non-movement with a dotted line in order to demonstrate that the larger of the two probabilities will cause the output ratio to dip below or rise above the threshold boundary of 'one'. Specifically, the averaged ratios appear to be significantly larger than the threshold boundary during movement and less than the boundary during non-movement. The ratios that appear near the threshold boundary of 'one', indicate that the probabilities of movement and non-movement are equivalent.

The above analysis led us to believe that by combining these weak single-channel predictions, we could generate an improved classification, and subsequently, an improved
final 3D mapping. In the next section, we investigate a framework that will incorporate the probabilistic output from these weak classifiers (see table 1) into a strong correct classification.

Table 3–1: Classification performance of select neurons

<table>
<thead>
<tr>
<th>Neuron #</th>
<th>Stationary %</th>
<th>Moving %</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>83.4</td>
<td>75.0</td>
</tr>
<tr>
<td>62</td>
<td>80.0</td>
<td>75.3</td>
</tr>
<tr>
<td>8</td>
<td>72.0</td>
<td>64.7</td>
</tr>
<tr>
<td>29</td>
<td>63.9</td>
<td>82.0</td>
</tr>
<tr>
<td>72</td>
<td>62.6</td>
<td>82.6</td>
</tr>
</tbody>
</table>

3.4.2 Structure

In section 3.4.1, we illustrated the result of finding the average ratio of the output probabilities from individual movement/non-movement HMM chains using corresponding neural channels as input. This simple measure motivated our first attempt to combine the probabilities into a simple classifier. We used the average ratio and applied a decision rule to the threshold in order to determine whether movement or non-movement occurred,

\[
\frac{1}{K} \sum_{k=1}^{K} \frac{P(O^{(k)}|\lambda_{M}^{(k)})}{P(O^{(k)}|\lambda_{S}^{(k)})} > y. \tag{3.14}
\]

Although simplistic, we demonstrate in section 3.3.5 that this approach is remarkably better than the VQ-HMM model. Unfortunately, the ratio itself is susceptible to infinitesimal probabilities, which in turn can cause extremely large output values. Consequently, these large ratios will bias the overall model and increase erroneous classifications. To minimize this bias, we apply the log to Equation 3.2:

\[
\frac{1}{K} \sum_{k=1}^{K} \log \left( \frac{P(O^{(k)}|\lambda_{M}^{(k)})}{P(O^{(k)}|\lambda_{S}^{(k)})} \right) \tag{3.15}
\]
In turn, this approximates:

$$\log \left( \prod_{k=1}^{K} \frac{P(O^{(k)}|\lambda^{(k)}_{M})}{P(O^{(k)}|\lambda^{(k)}_{S})} \right)$$

(Eq. 3.16)

We see that by applying the log to the ratios (Eq 3.6), we are essentially finding the relative scaling between each chain and avoiding the effects of any infinitesimal output probabilities (Eq 3.7). Similarly, the summing of the log ratios results in finding the log likelihood and subsequently takes the form of a particular HMM variation known as the Factorial Hidden Markov Model (FHMM) \[8\].

The graphical model for a general FHMM is shown in Figure 3-9. The system is composed of a set of $K$ chains indexed by $k$. The state node for the $k$th chain at time
Figure 3–10: Comparision of our FHMM to general model

$t$ is represented by $X_t(k)$ and the transition matrix for the $k$th chain is represented by $A(k)$. The overall transition probability for the system is obtained by taking the product across the intra-chain transition probabilities:

$$P(X_t|X_{t-1}) = \prod A^{(k)}(X_t^{(k)}|X_{t-1}^{(k)}), \quad (3.17)$$

Our departure from this general model occurs at the output vector node (Figure 3-3). Instead of each chain being stochastically coupled at the output node (represented by a vector), our HMM multi-chain structure independently uses a single element from the output vector node for each chain (Figure 3-4) leaving the chains fully uncoupled. This FHMM variant is used in other research involving speech processing [9] and has an association with another structure called parallel model composition [9].

Specifically during evaluation, our model uses the neural binned spike data from the $k_{th}$ channel in order to evaluate the $k_{th}$ conditional probabilities

$$P(O^{(k)}|\lambda_S^{(k)}) \text{ and } P(O^{(k)}|\lambda_M^{(k)}), \quad (3.18)$$

where,

$$O^{(k)} = \{ O_{t-N+1}^{(k)}, O_{t-N+2}^{(k)}, O_{t-1}^{(k)}, O_t^{(k)} \}, N > 1 \quad (3.19)$$

and $\lambda_S^{(k)}$ and $\lambda_M^{(k)}$ denote HMM parameters that represent the two possible states of the monkey’s arm (moving vs non-moving) for a particular HMM chain $k$. Since our FHMM
reduces to a set of uncoupled HMM chains, we evaluate the individual chains with the same procedure described in section 3.3.2. Before evaluation though, each HMM chain is previously trained on the neural spike data (using the Baum-Welch algorithm) which we describe in the next section.

3.4.3 Training

Updating the parameters for a FHMM is an iterative, two-phase procedure and is only slightly different from the training of a single HMM chain as described in section 3.3.3.

In the first phase, we use the Baum-Welch method to calculate expectations for the hidden states. This is done independently for each of the $K$ chains, making reference to the current values of the parameters $\lambda_t^{(k)}$.

In the second phase, the parameters $\lambda_t^{(k)}$ are updated based on the expectations computed in the first phase \([16]\). The procedure then returns to the first phase and iterates. We note that the input into each left-to-right (or Bakis) HMM chain $k$ is the spiking bin counts of a corresponding neuron $k$.

Coincidentally, our simple variation of the FHMM naturally appears as a substructure approximation (Figure 3-10) to the computationally difficult joint probability distribution,

$$P(X_t^{(k)}, Y_t|\theta) = \prod_{k=1}^{K} \left[ \pi^{(k)}(X_1^{(k)}) \prod_{t=2}^{T} A^{(k)}(X_t^{(k)}|X_{t-1}^{(k)}) \right] \prod_{t=1}^{T} P(Y_t|X_t^{(k)})$$ \hspace{1cm} (3.20)

\([8, 14, 16]\). Consequently, our FHMM variant has an advantage in the training procedure since it simplifies to the training procedure for a single HMM chain described in 3.2.2 (just repeated K times). Additionally, with the reduction in computation, our model can be trained with pre-existing software and even has the ability to be distributed over a parallel computing architecture (as opposed to a general FHMM). In Chapter 6, we detail our particular distributed implementation of this model using such an architecture.
3.4.4 Results

After observing some qualitative properties of the ratios, we now seek to quantify our initial approach (Equation 3.2) as well as our FHMM classifier. Shown in Figure 3-12, we present a biasing plot of our initial classifier with the decision criterion

\[
\frac{1}{K} \sum_{k=1}^{K} \frac{P(O^{(k)}|\lambda_M^{(k)})}{P(O^{(k)}|\lambda_S^{(k)})} > y.
\] (3.21)

From this graph, we notice that as the threshold boundary is manipulated, classification performance for the movement/stationary classes shifts. We can clearly see that the joint maximum (or equilibrium point) occurs near the area where the threshold is 1.04 (a value of one represents the 1:1 relationship of the ratio of probabilities). We also observe that this simple classifier has a significant improvement in classification as compared to our previous VQ-HMM classifier since the equilibrium point shows that movement and non-movement classifications occur around 94% (as opposed to 87% in our previous work). Note that without the threshold, the results do not show any important significance (except better than random).

As explained earlier, the criterion in Equation 3.14 is prone to extreme bias if an individual classifier produces infinitesimal probabilities. In an attempt to avoid biased probabilistic ratios, we also evaluated the ratios-of-means criterion.
Figure 3–12: Biasing plot for modified naive classifier

\[
\frac{1}{K} \sum P(O^{(k)}|\lambda^{(k)}_M) = \frac{\sum P(O^{(k)}|\lambda^{(k)}_M)}{\sum P(O^{(k)}|\lambda^{(k)}_S)} > y. \tag{3.22}
\]

Unfortunately, we see in Figure 3.22 that despite the ability to avoid infinitesimal probabilities, the ratio-of-means classifier produced inferior results compared to the initial mean-of-ratios classifier. Considering our discussion earlier, we know that since some neurons are more tuned the movement/non-movement than others, by averaging the probabilities we degrade individual neural contributions and simply provide a diluted consensus to which motion primitive is occurring. Therefore, finding the relative scaling between the outputs of single HMM chains allows us to incorporate amplified predictions into a better overall classification.

Our FHMM model has the ability to find the relative scaling between the outputs and can circumvent infinitesimal probabilities. In Figure 3.22 we see that the FHMM is able to achieve much better performance than the ratio-of-means classifier and slightly better performance than the mean-of-ratios classifier. Consequently, we must now observe which model, after training, can calculate a threshold boundary that will allow maximal classifications during the testing of new data. Without the calculation of this threshold boundary, none of the multi-chain models can perform better than the VQHMM model.
Figure 3–13: FHMM evaluation

Figure 3–14: Training data on naive classifier
Figure 3–15: Training data on FHMM

Since the means-of-ratios classifier retains problems with infinitesimal output values, we only compare the ratio-of-means classifier and the FHMM. We see in Figures 3.22 and 3.22 that both methods demonstrate a similar threshold point is achievable with the training set. Unfortunately, we also notice a distinct classification difference. Specifically, we observe that the ratio-of-means classifier in Figure 3.12 performs less effectively than the FHMM, which confirms our suggestion that the overall averaging effect dilutes the strong neural classifiers into a weak classification (since training data should produce higher classifications). Alternatively, Figure 3.44 shows FHMM results that are consistent with a model trained and tested with the same data (high classifications - 97%).

3.5 Discussion

We make several observations about our results. First, there appears to be a significant statistical difference in the neural spike data for the two classes of arm motion. Second, increased temporal structure leads to better classification performance; that is, arm motion is correlated not only with the most recent neural firings, but a short-time history of neural firings. We can also hypothesize a number of sources of residual classification error.
First, the amount of data analyzed in these experiments is relatively limited, given the size of the statistical models employed. Note, for example, that the classification results for the moving-arm class are worse than those for the stationary class; this may be a reflection of the relative amount of data available for training and testing in each class.

Second, in the signal-to-symbol conversion of the multi-channel data (for VQHMM), we lose a substantial amount of information. Even for 256 prototype vectors, the consequent distortion (uncertainty) in the symbol data is substantial. We see that by using the FHMM to break down the problem space into individual HMM chains, we avoid the introduced distortion from VQ and can classify the monkey’s arm state more accurately.

Overall, we see that the FHMM is a better switching model than our previous attempts. We demonstrated that it could avoid infinitesimal probabilities and yet find a relative scaling between the movement and non-movement probabilities. Additionally, the FHMM is able to represent a large effective state space with a much smaller number of parameters than a single unstructured HMM. Consequently, this model is easily distributed across a parallel computing architecture and can be used with pre-existing training/evaluation software. Finally, we remark that since the FHMM is a probabilistic framework, we can find other unique connections between the nodes in this graphical model to better exploit the spatial and temporal dependencies of the neuronal firings.

In the next Chapter, we describe how we use our switching classifier in combination with multiple continuous local linear models to predict the 3D coordinates of the monkey’s hand.
CHAPTER 4
LOCAL LINEAR MODELS WITH HMM PARTITIONING

4.1 Structure

The final step in our bimodal mapping structure is to take the outputs from the HMM-based classifier and generate an overall mapping of neural data to 3D arm position. To establish a baseline for this approach – namely the prior segmenting of neural data at the input into multiple classes – we assign a single local-linear model (LLM) to each class, and train each of the LLMs only on data that corresponds to its respective class, as shown in Figure 4.1. Each LLM adapts its weights using normalized least mean square (NLMS) [15]. After training, test inputs are fed first to the HMM-based classifier, which acts as a switching function for the LLMs. Based on the relative observation probabilities produced by the two HMMs and the decision boundary, as given in equation (3.4), one of the two LLMs is selected to generate the continuous 3D arm position. With properly trained HMMs, the bimodal system should be able to estimate hand positions with reasonably small errors.

![Figure 4–1: Bimodal mapping overview](image)

Given 104 neural channels, each LLM is defined with 10 time delays (1 sec), and 3 outputs so that its weight vector has 3,120 elements (Figure 4-2). The LLMs were trained on a set of 10,000 consecutive bins (1,000 sec.) of data with a NLMS learning
rate of 0.03. Weights for each LLM were adapted for 100 cycles. After training, all model parameters were fixed and 2,988 consecutive bins of test neural data were fed to the model to predict hand positions. The results of the experiments were then evaluated in terms of short-time correlation coefficients and the short-time signal-to-error ratio (SER) between actual and estimated arm position. For each measure, the short-time window was set to 40 bins (4 sec) since a typical hand movement lasts approximately four seconds.

![Local linear model](image)

**Figure 4–2: Local linear model**

Of course, a correlation coefficient value of 1 indicates a perfect linear relationship between the desired (actual) and predicted (system) trajectories, while 0 indicates no linear relationship. The second measure, the SER, is defined as the power of the desired signal divided by the power of the estimation error. Since a high correlation coefficient cannot account for biases in the two trajectories, the SER complements the correlation coefficient to give a more meaningful measure of prediction performance. Finally, all of our bimodal-system results are compared with two other approaches, namely, a recurrent neural network (RNN) [11] and a single LLM.

### 4.2 Results

In this section, we report results for neural-to-motor mappings of a single LLM, a recurrent neural network (RNN) and the bimodal system. Since the segmentation
results are better for the displacement-based segmentation, we use these HMMs in the first stage of the bimodal system. In Figure 4-3, we plot the predicted hand trajectories of each modeling approach, superimposed over the desired (actual) arm trajectories for the test data; for simplicity, we only plot the trajectory along the z-coordinate. Qualitatively, we observe that the bimodal system performs better than the others in terms of reaching targets; this is especially evident for the first, second, and the seventh peaks in the trajectory. We also observe that during stationary periods the bimodal system produces less noisy outputs than the other models.

Overall, prediction performance of the bimodal system is better than the RNN, and superior to the single LLM, as evidenced by the empirical cumulative error measure (CEM), plotted in Figure 4-4. Figure 4-4 shows that the population distribution functions of L2-norms of error vectors of the bimodal system and the RNN are similar, and significantly better than the single LLM.

The correlation coefficients over the whole test set, averaged over all three dimensions, are 0.64, 0.75, and 0.80 for the single LLM, the RNN, and the bimodal system respectively. The mean of the SER averaged over all dimensions for the single LLM, the RNN, and the bimodal system are -20.3dB +/- 1.6 (SD), -12.4dB +/- 16.5, and -15dB +/- 18.8, respectively. Although these measurements give us insight into the overall performance of the models, they fail to express the difference in accuracy between each model when predicting movement. The accuracy in movement is more important than in non-movement since we can remove non-movement errors by using the output of
the HMM as a filter. This led us to compute the SER and CCs over movement sections of the test set (partitioned by hand). The CCs for only the movement sections were 0.83 +/- 0.07, 0.84 +/- 0.14 and 0.86 +/- 0.11 for the LLM, RNN and bimodal system, respectively. The SER over the movement sections are 2.96 +/- 2.68, 6.36 +/- 3.71 and 8.48 +/- 4.47 for the LLM, RNN and bimodal system, respectively.

4.3 Conclusion

We see the estimation performance of the hand trajectory of the proposed bimodal system is better than the RNN, and superior to a single LLM. It is also apparent from the results that using multiple models improves the estimation performance compared to the single filter, although it adds more computational complexity. Compared to the RNN, the bimodal system reduces the complexity of training significantly and
produces a more accurate estimation. The drawback with the bimodal system is that its estimation mainly depends on the classification ability of the HMM (as seen in Figure 4-1). Chapter 6 focuses on ways to improve HMM classifications to remove these false errors.
CHAPTER 5
DISCRETE SIGNAL PROCESSING HARDWARE IMPLEMENTATION FOR A BMI

5.1 Introduction

As described in chapter 2, the feasibility of building brain machine interfaces (BMI) has been demonstrated with the use of digital computational hardware [24, 25]. For these interfaces, researchers first acquire analog neural recordings and process them through spike detection hardware and software. Once the neural signals are processed, large rack-mountable high-end processors are used in conjunction with Matlab-enabled PCs to predict a subject’s arm trajectory in real-time [10, 25].

The ultimate BMI goal envisions that free-roaming subjects will possess the prediction algorithms and hardware in vivo as they physically interact in the world. Unfortunately, at the current stage of development, most researchers require the subjects to be tethered to a cluster of immobile machines. Other researchers have removed this tether by wirelessly transmitting the analog neural signals off the subject but still require a local cluster of PCs for predicting the output [20, 23]. Although this wireless approach is more inline with the ultimate BMI goal, research emphasis has been placed on shrinking the wireless acquisition hardware which still requires large immobile machines to predict trajectories [25]. Additionally, the transmission of neural waveforms has bandwidth limitations. A bandwidth bottleneck occurs as more neurons are sampled from the brain making the transmission of these signals to the digital hardware arduous and power consuming (which is contrary to the necessities of such a device) [20].

We believe that by shrinking digital hardware that computes the prediction algorithms (on the subject), the wireless limitations and immobility issues are both solvable. Specifically, the proposed solution is to directly connect the analog and digital subsystems with a high-speed data bus that is more power efficient and faster than the fastest available wireless link. Furthermore, using on-board digital hardware to
predict the trajectory removes the need for large external computers and approaches the ultimate BMI goal of patient mobility.

In this chapter, we present a design for a wearable computational DSP system that is capable of processing various neural-to-motor translation algorithms. The system first acquires the neural data through a high speed data bus in order to train and evaluate our prediction models. Then via a widely used protocol, the low-bandwidth output trajectory is wirelessly transmitted to a simulated robot arm. This system has been built and successfully tested with real data.

The organization of the chapter is as follows. We first cover the system requirements and then outline the system design that meet these requirements in terms of the hardware modules and the software layers. Finally, we present the results of the system, followed by a conclusion.

5.2 System Requirements

In order to create a successful system, it is necessary to first address the technical and practical aspects of such a system by determining its functionality within its intended environment. For example, how will the hardware extract information from the brain? How will it be powered? What type of processing power is necessary for a successful algorithm to be implemented? What are the physical constraints? How can faulty hardware be debugged and fixed? How will the neural predictions be expressed in the external world?
In addressing these broad requirements, we look to the ultimate BMI goal of having a transplantable chip under the skin that can acquire neural firings from the brain and translate them into action in the external world. Unfortunately, as with all long term technological goals we must evolve the hardware in order to verify the designs as we shrink and combine them into a final solution. At each stage of development, some of the requirements must be relaxed or tightened depending on which portion of the design we are trying to verify.

Specifically for our current stage of development, the hardware must first receive digitized neural data during the training and evaluation of the neural-to-motor prediction models. After evaluating the output, the predicted trajectory must then be transmitted off-board through a wireless connection to a receiving computer/robot arm representing the desired control. This wireless connection must also provide the ability to remotely program and diagnose the system when being carried by a subject.

The described (WIFI-DSP) system serves as the digital portion of the overall BMI structure and is responsible for translating the neural firings into action in the external world. The first generation of this hardware was housed in a PCI slot of a personal computer and did not possess wireless capabilities [26]. For the second generation (discussed in this chapter) we require a design that is portable, possesses wireless capabilities and is computationally fast.

Since the system needs to be portable and contain a wireless connection, we must resolve how the other system requirements are affected. First, a portable system must be light-weight and small enough for a human or primate to carry. Second, a portable system must also be self-contained and rely only on battery power. Consequently, the hardware needs to be low-power in order to extend the life of this on-board battery. The low power constraint will then influence the choice of the processor since we need a low power device that can still achieve fast processing speeds. This power constraint also
Figure 5–2: Hardware Modules of WIFI DSP

affects the wireless connection since it needs be low power, yet retain enough bandwidth to transmit the output trajectory and any future data streams.

The prediction models running on the hardware platform also constrain the system design. Since most of the prediction models require the use of floating point numbers and arithmetic, we need a system that can process these floating point numbers fast enough to attain real-time model computations. Additionally, these models are sometimes large or contain multiple versions running simultaneously and therefore require large memory banks to handle the data throughput.

5.3 System Design

In this section, we explain what components were chosen for the system, why they were chosen and how they fulfill the requirements outlined in the previous section.
5.3.1 Processor

The central component to any computational system is the processor. The processor can sometimes determine the speed, computational ability and power consumption of the entire system. This central component also dictates what support devices are required for the design. In choosing a processor for our particular system, we looked to the previous work of our colleagues Scott Morrison and Jeremy Parks. They were able to verify that the Texas Instruments TMS320VC33 (C33) was an appropriate processor for our needs [26, 27]. It was also advantageous to use this processor since they had developed a code library and hardware infrastructure for us not to be burdened with.

![Figure 5–3: DSP Features](image)

Specifically, the C33 meets our floating point and high speed requirements since it is a floating-point DSP capable of up to 200 MFLOPS (with over clocking). It achieves such high speeds by taking advantage of its dedicated floating point/integer multiplier. Since it works in tandem with the ALU, it also has the ability to compute two mathematical operations in a single cycle. Another reason this processor is so
efficient in processing is that it can perform two reads, one multiply, and one store in a single cycle by utilizing its dual address generators for simultaneous RAM access [26]. With regard to processing our BMI algorithms, Scott Morrison was able to verify two of our groups cornerstone algorithms worked much faster on the C33 than with a 600MHZ Pentium III computer [26, 27].

The C33 also meets our low power constraint since it uses less than 200mW at 200MFLOPS. It achieves such power savings due in part to its 1.8V core and other power saving measures built into the processor [22]. Although the processor uses such a low voltage for its core processing, the I/O supports 3.3V volt signals. Unfortunately, the DSP requires external translation logic to interface all 5V devices (like the PCMCIA card) in order to meet the 3.3V I/O specs. We used two components for translating 5V to 3.3V. One was the Texas Instruments SN74CBTD3384DBQR 10-bit level shifters since it is fast (0.25ns) and available in small 24-pin SSOP packaging. The other was a epm3064c44 CPLD that was also used for the interconnection logic of the different hardware modules which we discuss later in this section.

Finally the C33 was able to fulfill other requirements of the system. The first is its ability to support a large amount of memory. This is accomplished with its 24-bit address bus allowing 16 million different memory locations. It also allows quick access to these locations by using hardware strobe lines PAGE0-3 to directly provide access to control logic to different memory blocks. This processor also meets the requirement of expandability as discussed in the last section. It is expandable since it has four hardware interrupts, two 32-bit timers, and a DMA controller.

5.3.2 Wireless connection

The second most important hardware module in our system is the wireless connection. We determined that 802.11B would be the most appropriate protocol since it is easy for our group and our collaborators to interface with. The protocol not only provides a large amount of bandwidth, it also has inspired a large code infrastructure
for communication clients and servers. Additionally, by using such a widely accepted protocol, instead of developing a new one, we are able to communicate to any off-the-shelf wireless device that supports 802.11b. Essentially the system can connect to any computer or hardware device on the internet.

We designed the system to use a MA401 PCMCIA wireless 802.11B card. This PCMCIA card was the smallest and fastest card available during our design process. At the core of this card is an Intersil prism 2 chipset which is responsible for handling most of the physical layer and MAC addressing of the 802.11B protocol. The control of this chipset involves different sequences of register calls. Subsequently, these register calls help to configure, initialize, and transmit data to/from the card.

The PCMCIA card met the power requirements for this development stage since we have the ability to vary the power consumption from less than 100mA to 300mA.
depending on the bandwidth we require. It also meets our size requirements since it is slightly larger than a credit card as shown in Figure 5-8. This device also meets the final requirement of high bandwidth since it is capable of transferring data at 10mbits/s. This high bandwidth is appropriate since it may become necessary to retrieve neural data wirelessly if a sub-dermal analog acquisition system is designed and requires data transmission through the skin.

5.3.3 Boot Control

In order for the system to be self reliant the DSP needs to boot from an EEPROM with the appropriate system software. This booting process is accomplished through hardware interrupts on the DSP. Given a certain interrupt when the system comes out reseting or powering up, the DSP with assert certain memory locations to begin reading from. Using the CPLD to provide the control logic and a jumper system we can cause the DSP to boot from the EEPROM or the USB interface. The only time we would boot from USB is the system is being debugged or testing during manufacturing.

The EEPROM device we chose had been selected by Scott and Jeremy previous for their design and showed to be successful with this DSP. The device is an Atmel AT29C256 and has a small PLCC package to meet our size requirements. In terms of our power requirements this chip also meets them since it only uses blah during use (which is only at bootup). Another feature of the device that serves our needs is that it has 256k of memory so that we can place fairly large software packages within it.

Figure 5–5: Two different methods to boot the C33: EEPROM or USB
5.3.4 USB

We chose the FT245BM USB FIFO device for a USB 2.0 interface. This device is small and basic to control. Additionally, it meets our low power constraint since it only uses 25mA in continuous mode and 100uA in suspend mode. Consequently, this chip provides a range of options for data throughput requirements versus power consumption. This chip also provides an 8 mbits/s data bus for the dual purposes of data communications and system diagnosis.

Figure 5–6: Block Diagram of USB Interface

5.3.5 SRAM and Expansion

The DSP has 34K by 32-bits internal high-speed SRAM. As mentioned earlier, the prediction models require more than this internal limit. Therefore, additional external 32-bit SRAM is required to connect to the C33 data bus. Unfortunately, many of the desired components and alternatives were not available during the design process. Consequently, we chose four 8-bit Cypress CY7C1049B-15VC memory chips. These memory chips fulfill many of the requirements of this stage of development. First they possess fast access time (15 ns). Second, they have low active power, (1320 mW.
max.) and low CMOS standby power, (2.75 mW max.) Finally, they provide easy
memory expansion with their chip enable (CE) and output enable (OE) features

Having four CY7C1049B parts yields a total of 512k by 32-bits or 2MB of external
memory. These four parts are incorporated using the same chip enable line connected to
different bytes of the data bus, giving the appearance of a single 32-bit memory [26].

![512K by 32-bit External SRAM Architecture](image)

Figure 5–7: 512K by 32 bit external SRAM architecture.

5.3.6 Power Subsystem

There are three different power requirements in order to power the WIFI-DSP, 1.8V, 3.3V, and 5V. Texas Instruments offers a TPS70351 Dual-Output LDO Voltage Regulator that includes both 1.8V and 3.3V voltages on a single chip but only requires 5V to operate. The two output voltages are not only used by the DSP, they are also used for the other hardware modules in the system. This chip also provides the power-up sequence required by the DSP once it is initialized. Additionally, having one required voltage input source is an advantage for this portable system since only one 5V battery supply is necessary.
5.4 Complex Programmable Logic Device

The Complex Programmable Logic Device (CPLD) is responsible for 6 hardware components: 1) C33 DSP, 2) PCMCIA 802.11B, 3) USB, 4) External SRAM, 5) Bootable EEPROM, and 6) Power Regulator. This chip provides the control logic between the devices using various signals. These signals include Write Enable, Read Enable, Chip Select, and Reset signals. By using address and control signals the CPLD is able to define multiple memory locations of the DSP so that specific hardware components can be read and written to/from. Specifically the decoded address spaces were created for the EEPROM, SRAM, PCMCIA 802.11B card and the USB interface. The CPLD also provides interrupt control to the DSP based on the real-time operating system implemented in the DSP during the communication between the C33 and the USB bus. The final requirement of the CPLD is to provide an interface for any future expansion hardware that may become required.

Given the above requirements for the CPLD control and the number of interface signals needed, the Altera EPM3064 CPLD was chosen. A member of the Altera MAX family, this 100-pin TQFP chip provides 64 pins of I/O with 1200 usable gates and
64 Macrocells. Additionally, it is fast enough to meet our memory and bus speed requirements as well as low power enough to meet our current power requirements.

To control the CPLD device, VHDL code provides the necessary control signals for each component on the board. Once the VHDL code is compiled, the CPLD is programmed through the 10-pin ByteBlaster port. This allows the reconfigurable CPLD to become a flexible architecture as the BMI requirements change and necessary modifications become important.

5.5 System Software

Software is one of the most important components to any hardware system. For a BMI to be successful there will be necessary software that accompanies the final solution. Specifically for our hardware solution, there are five major levels of software in the DSP Board environment: 1) PC Software, 2) DSP Operating System, and 3) DSP Algorithms, 4)802.11B Code 5)TCP/IP protocol. This section will describe in brief the operation of the software and its interaction between the multiple hardware modules.

5.5.1 PC Software

Using Visual Basic, we wrote a PC console program to interface the DSP through the USB. The console program calls functions within the DSP OS to initiate and control the USB communication functions. The DSP OS is also responsible for reading/writing memory locations and various program control functions.

In order to make the communication with the DSP work, the console program needs to talk via the USB bus. The drivers for the USB device support Windows 98, 2000, and XP, and support the following function calls, as well as many others that were not used for this implementation.

Using the console program, the user may modify DSP configuration registers, download and execute DSP code, and view and edit memory locations. The code is easily modifiable to accommodate different testing requirements, such as different methods for streaming data to the DSP.
A simple and expandable protocol was created for PC-to-DSP communication. This protocol involves a series of 32-bit commands containing different opcodes. The DSP operating system was written to support these series of commands, which include the ability to read from the DSP, write to the DSP, and execute a program in DSP memory.

In tandem with the OS layer of the DSP, there is low-level driver code for initializing and controlling the 802.11B wireless controller. This code must interact with the DSP OS and any UDP client code or algorithms that are running simultaneously within the WIFI-DSP. Once the prediction model completes an epoch or computation cycle, the program must interrupt the wireless card and transfer any required data. This process also involves creating the correct UDP packets for transmission to the appropriate UDP server (with a specific IP address).

5.6 Results

The WIFI-DSP has been fully tested via its USB interface and wireless communication in the following manner. Neural data was acquired through the USB port and used in both training and evaluation (forward) modes on the WIFI-DSP.
Figure 5–10: NLMS Performance

system. Specifically, the DSP was programmed to train an NLMS algorithm and upon completion, trajectory predictions were transmitted off board through the 802.11b wireless interface using a UDP client protocol. This communication occurred bi-directionally with an external laptop running as a typical UDP server.

The LMS output results collected at the receiving computer were directly compared to Matlab computed outputs. These results are accurate within 7 decimal places of the Matlab double precision results.

The bandwidth of the wireless link (802.11b) is around 1.8 M bit/sec in continuous operation. This is comparable to what is expected on a 3Ghz Pentium laptop using the same Netgear wireless adapter with a Prism II chipset.

The current consumption is approximately 350 mA for the entire board which equates to 1750mW. Of this consumed power, over 80% or 1400mW is used by the PCMCIA wireless adapter. Overall, this is much less than the 4W previously attained by other acquisition hardware [21].
CHAPTER 6
FUTURE WORK

6.1 Hardware

We have a working system that demonstrates LMS training on a DSP platform. Further, we were able to verify that it wirelessly transmits results to 802.11b enabled devices. However we are disappointed in the size and power consumption of the WIFI DSP board. As mentioned throughout the paper, for this development stage we relaxed some of the constraints in order to verify the technologies and fuse them into a single system. For the next generation, we want to shrink the system in half and reduce the power consumption. Because the majority of the power is being consumed by the wireless adapter, it is necessary to find a lower power wireless link. We also want to verify the ability of the WIFI-DSP to directly communicate to analog acquisition hardware.

The WIFI-DSP system demonstrates that by shrinking digital hardware that computes the prediction algorithms (on the subject), the wireless limitations and immobility issues are both solvable. Specifically, connecting the analog and digital subsystems with a high-speed data bus is more power efficient and faster than any wireless link. Furthermore, using on-board digital hardware to predict the trajectory removes the need for large external computers and approaches the ultimate BMI goal of patient mobility.

6.2 FHMM Applications

We note that the final prediction performance of the proposed bimodal system is much better than the RNN, and superior to that of the single LLM model. Clearly, the use of multiple models improves prediction performance over a single LLM model, at some additional computational cost. Furthermore, by increasing the repertoire of motion primitives we may improve 3D mapping by allowing the linear/nonlinear models, on
the output stage of the bimodal structure, to fine tune to a specific primitive. Since
the underlying motion 'primitives' are unknown, we seek in future work to form an
unsupervised method of segmenting these 'primitives'.

We believe that the FHMM structure may lead us to the solution. Figure 6-1 shows
where our hand segmentation incurred errors (red arrows). We want to find out what the
FHMM is classifying during these user segmentation errors. Specifically, is it classifying
these sections as movement despite the fact that we segmented them as stationary?

In Figure 6-2, there seems to be an indication that the FHMM is in some way
detecting our hand segmentation errors (in green) and producing correct classification
(despite our labeling it incorrectly). This leads us to believe that our classification
results may be higher than what we are reporting since our segmentation process is
slightly flawed.
Figure 6-2: Derivatives of spherical coordinates with blue movement segmentations superimposed
The question arises as to why the FHMM failed to discover our error in the sections labeled as A in both figures (6-1 and 6-2). The FHMM appears to not recognize this section as movement despite our claim that it is movement (based on above assumptions).

We see that monkey makes no discernable motion similar to his food-grasping task. We still observe that the monkey did move its arm in some fashion (though not fitting our criteria for segmentation). In Figure 6-3, we see the probabilities plotted in time respective to the time slice for the desired data.

Figure 6-3 shows $\frac{1}{n} \sum P(O|\lambda_M)$ plotted in green and $\frac{1}{n} \sum P(O|\lambda_S)$ in blue. We see a rough approximation of when the FHMM is indicating that motion may in fact be taking place at both location A and B (which we can confirm with Figure 6-3). This tells us that the FHMM may be able to cluster the input. Looking within the FHMM we beleive the single HMM chains themselves may also be optimized to help in our clustering process.

Since we know the classification performance of the individual HMM chains during training, we believe that this information can allow us to weight the individual classifiers performance by using adaboost or some other scheme. In Figure 6-4 we used neuronal ranking information to see if we could observe the importance of the single HMM chains on the overall classifier. We start with the top-ranked neurons in his list and then
continually add neurons from best to worst and plot their classification performance. This experiment involved no bias/threshold.

We note from the Figure 6-4 that after a certain point of continually adding weaker neural-classifiers, our classification performance only plateaus.

In Figure 6-5, we again display a best to worst neuron adding experiment. However, in this experiment we use a bias/threshold and find the equilibrium point (joint maximum between both classes). In contrast to our previous graph, we see that despite adding weaker neural classifiers we observe continuing improvement in our classification results. Also of interest, is the bias/threshold (in red) continues to converge to zero as we add more and more single-neural classifiers.

Finally, we remark on the fact that we have not yet optimized the single HMM chains within the FHMM framework. With our previous work we exhaustively manipulated the parameter of the VQHMM (sequence length, states, etc.) and were able to increase our results a few percentage points. Perhaps changing these parameters and the decision criterion can garner a performance increase.
Figure 6–5: Worst neuron to best neuron predictors (with bias)
REFERENCES


[27] S. Morrison, A DSP-Based Computational Engine for a Brain-Machine Interface,
BIOGRAPHICAL SKETCH

Shalom Darmanjian graduated from the University of Florida with a Bachelor of Science in Computer Engineering in December 2003. After completing this thesis in May 2005 Shalom plans to continue his pursuit of knowledge in the Ph.D. program at the University of Florida.