Investigating the Robust Ability of Classic, Modern, and Contemporary Brain Machine Interface Decoders

A Thesis Proposal

Submitted to:

Temple University Graduate Board

In Partial Fulfillment

Of the Requirements of the Degree

MASTER OF SCIENCE

In ELECTRICAL ENGINEERING

Michael Samarco

November 2023

Thesis Approval(s):

Iyad Obeid, PhD., Thesis Advisor, Electrical and Computer Engineering

Joseph Picone, PhD., Electrical and Computer Engineering

Chang-Hee Won, PhD., Electrical and Computer Engineering

**Abstract**

Brain Machine Interface (BMI) decoders make it possible to mathematically model brain electrical activity in relation to a task or thought process by a human or other animal. Decoders are thus vital to the development of prosthetics, as well as other forms of brain-machine control. Many efforts today are concerned with decoding brain activity to interpret tasks carried out (or desired to be) by the body’s motor system. This Thesis builds from existing modern BMI decoding research concerned with arm reaching tasks. In this project, four decoders will be evaluated when the neural data presented to them is imperfect or not of a conventional format. Specifically, the four decoding algorithms of focus are the classic linear regression, the popular Kalman Filter in both a supervised and unsupervised format, and a contemporary choice—the recurrent Exponential-Family Harmonium (rEFH). In a recent paper ([Makin et al., 2018](#Makin_2018)) these algorithms were evaluated for performance on conventional spike sorted neural datasets, where the rEFH proved superior. This paper is interested in pushing that effort a little further and gauging robustness of the decoders. That is, quantifying how well they can perform with minimum training (transfer learning); quantifying how well they can perform when the spike sorting process is skipped, and the decoders are fed multi-unit data; and quantifying how well the decoders can cope to electrode channels being dropped at random.

**Contents**

[1 Introduction 1](#_Toc152099827)

[1.1 Motivation 1](#_Toc152099828)

[1.2 Background 3](#_Toc152099829)

[1.2.1 Data Acquisition Overview 3](#_Toc152099830)

[1.2.2 Spike Sorted Data 5](#_Toc152099831)

[1.2.3 Multi-Unit Data 6](#_Toc152099832)

[1.2.4 Neural Decoding 7](#_Toc152099833)

[1.2.5 Data Collection Issues 9](#_Toc152099834)

[2 Research Objectives 10](#_Toc152099835)

[3 Methodology 12](#_Toc152099836)

[3.1 Data Formats 13](#_Toc152099837)

[3.1.1 Original Data Format and Binning Concept 13](#_Toc152099838)

[3.1.2 Multi-Unit Data Format 15](#_Toc152099839)

[3.1.3 Channel Dropping Format 17](#_Toc152099840)

[3.2 Data Processing 17](#_Toc152099841)

[3.2.1 Transfer Learning 17](#_Toc152099842)

[3.2.2 Decoding Algorithms 18](#_Toc152099843)

[4 Implementation 25](#_Toc152099844)

[4.1 Preliminary Work 25](#_Toc152099845)

[4.2 Proposed Work 30](#_Toc152099846)

[5 Appendix 35](#_Toc152099847)

[5.1 Variable Notation 35](#_Toc152099848)

[5.2 Variable Definitions 35](#_Toc152099849)

[6 Bibliography 37](#_Toc152099850)

# Introduction

## Motivation

A brain-machine interface (BMI) is a system in which the brain’s electrical signals, or neuronal activity, are acquired and processed directly by a computer or other external device(s). These systems are important both for their applications and for basic research. As applications, they can be tools that allow patients of injury or disease to regain a measure of autonomy through direct control of computers or prosthetics. As a research tool, they allow direct analysis of neural signaling at the single neuron level.

The BMI field has seen substantial growth in recent years. In 2016, and for the first time, a non-invasive electroencephalogram BMI system demonstrated a reach, grasp, and release task effectively amongst 13 human subjects in a 3-dimensional maneuverable space ([Meng et al., 2016](#Meng_2016)). Another group of researchers in 2016 demonstrated for [what they also believe to be] the first time, an intracortical reanimation of muscle control from a BMI system involving surgically implanted microelectrode arrays in the brain ([Bouton et al., 2016](#Bouton_2016)). In that effort, Bouton and group demonstrated successful hand/arm gestures with overall accuracy of their system showing ~70% (at by permutation test). [Bouton et al., 2016](#Bouton_2016) also captured and published video of their experiment, which included the test subject (human with quadriplegia) performing various hand and wrist gestures as well as functional tasks for grasping, pouring, and stirring.

Additionally, and from the standpoint of pure neuron analysis, research published in 2015 ([Aflalo et al., 2015](#Aflao_2015)) took aim at validating a hypothesis that a region of the brain called the posterior parietal cortex (PPC)—a non-conventional BMI neural acquisition region—could be used in BMI control. The PPC is primarily attributed with thoughts and planning and not actual motor function like the region commonly employed in BMI systems, the primary motor cortex. Prior to this publication, there was no direct neural recorded evidence or attempt to control external devices from BMIs using electrodes placed in the PPC. The researchers recorded at the single neuron level in the PPC and—in observing single neuron activation, or “spiking,” whilst a human tetraplegic subject was directed to imagine certain tasks or goals—they were able to deduce the composition of “goal” and/or “trajectory” focused neurons in that region of the brain. They then took that a step further, capturing test video and demonstrating control of a robotic limb from neural data collected in the PPC region of the brain.

More generally, research in the BMI field has gained tremendous interest since 2000 with the number of publications in the field following an exponentially increasing trend ([Salahuddin and Gao, 2021](#Salahuddin_Gao_2021)). This has led to the realization of commercialized BMIs and BMI acquisition software (for example, the [EmotivPRO](https://www.emotiv.com/emotiv-xtrodes-eeg-solution/) software and [EPOC X](https://www.emotiv.com/epoc-x/), [FLEX](https://www.emotiv.com/flex-saline/), etc. hardware by Emotiv and the N1 implantable device by Neuralink, which is “beginning its first-in-human clinical trial” ([Neuralink, 2023](#Neuralink_2023_blog)).

BMIs are an evolving technology that have the possibility to extend human capabilities remarkably. This project aims at continuing research efforts for recent BMI work. There are various motivating factors driving this effort. From a humanitarian perspective, BMI systems extend hope and independence to paraplegic individuals, who are victims of unfortunate circumstances (whether that be by disease or traumatic accident), in the prospect of restoring their lost natural motor function(s). From a perspective of learning how the brain works, BMIs are a practical tool that make it possible to study the brain in more detail. From the stance of contributing to technological advancement for humans, BMI systems present the possibility of realizing a world where devices can be controlled directly from human thoughts. Finally, simply from the argument of sustainability and wanting to promote growth in an already fast-growing field, BMI research is commanding interest from researchers from multiple disciplines (for example, the field of medicine, engineering, business, politics, ethics, etc.).

## Background

BMIs can be built using signals derived from many different sources such as single-units/neurons, electrocorticograms (ECoG), electroencephalograms (EEG), or peripheral nerve activity. This work focuses exclusively on neural decoding algorithms for BMIs intended for single-unit acquisition.

### Data Acquisition Overview

Figure 1 illustrates the typical overview and flow for acquired neural data in a single-unit BMI system. In single-unit BMI collection systems, microelectrode arrays are chronically implanted extracellularly into neural tissue. The electrodes, forming the array, sense the electrical signals generated by neurons in their proximity. A common assumption is that each electrode records spikes of individual neurons (especially in spike sorting), though some electrodes can measure activity from the same neuron ([Rinberg et al., 1998](#Rinberg_1998)). Each electrode will produce a raw signal comprised of the superposition of potentials sensed from neurons closest to it. These electrodes produce very small (typically in the range of 100s of microvolts) and noisy analog signals.

The electrode signals are typically passed to a pre-amplifier prior to processing the signals on a digital platform. The pre-amplifier attenuates unwanted noise via filtering, amplifies the signals, and then converts these analog signals into digital signals with quantized amplitudes and periodic sampling. A typical BMI preamplifier is the PZ2 Preamplifier (Tucker-Davis Technologies, Alachua, FL). The digitized electrode recordings can be processed raw (for example, potential vs. time) or “spike detected” to detect neuron action potentials. A leading theory of neural function is that information is encoded in neural spike timing.

To extract neural spike times, a processor system will typically be equipped with some spike detection, sorting, and binning algorithms. Spikes can be detected using static or adaptive simple thresholds, or with more sophisticated tools such as Wiener Filters. Following spike detection is spike sorting which classifies the detected spikes, sorting them to individual neurons. While spike sorting is technically an optional process, it is generally accepted as an important step. At this point, the “spikes” are reduced to merely “firing” times, or the time that the neuron spiked/activated. Since a spike can happen at any given time, these firing times are asynchronous and do not align with any periodic sampling rate. Therefore, to synchronize the firing times to a sampling period, the firing times are “binned” to produce spike counts at each periodic sampling interval.

A computer screen shot of a computer

Description automatically generated

Figure 1. This is an overview of a typical brain machine interface system. (a) The subject has chronically implanted electrodes in its brain to record neural activity. The subject can be a person or non-human primate. (b) An example photo of micro-electrode arrays implants in brain tissue (photo from [Rajan et al., 2015](#Rajan_2015)). Specifically, this image shows Utah arrays ([Blackrock Neurotech](https://blackrockneurotech.com/products/utah-array/#:~:text=What%20is%20the%20Utah%20Array,degree%20of%20precision%20and%20accuracy.) (New York City, New York)). (c) Each electrode measures raw electrical potential from the neurons that neighbor it. (d) A pre-amplifier, filters and amplifies as well as samples, or digitizes, the raw electrode signals. This conditions the signal for the processing system. (e) The processing system processes the digitized electrode signals and runs application algorithms on the data to achieve a task. (f) The algorithms typically consist of a spike detection phase, followed by (or optionally skipped) a spike sorting/classification algorithm. The “spiked” data is asynchronous and usually needs to be time aligned, or binned. The binned spike data then feeds the main application algorithm/model which essentially decodes that data and transforms it to an equivalent action, task, and/or state of a system. (g) Finally, the prediction from the decoder algorithm can be used to update the state of a device (for example, control position of a prosthesis).

The spike counts for each neuron (or electrode) on each sampling interval is passed to a decoder, which maps those spike counts to an equivalent state of a system. This allows for the potential to update a system to that predicted state.

Finally, it is worth noting that the pipeline for the neural data flow can be wired or wireless at any point after electrode measurement at the source (the brain). Wireless transmission can present some challenges that might need to be considered such as dropouts and mis-sorted data.

### Spike Sorted Data

Spike “sorting” is a method aimed at differentiating between multiple single neurons detected on the same electrode. Conventionally, spike sorting entails a three-step process ([Zhang et al., 2023](#Zhang_2023)). First, a spike detection algorithm reduces the electrode data from all time samples to just segments, or periods, where the electrodes are thought to have recorded a neuron firing/producing an action potential. Then, a feature extraction algorithm is deployed to discover features that best explain the differences amongst the different neurons. Finally, a classification algorithm is applied to the features and labels are placed for most likely fit of which neuron produced which “spike”. Spike sorting is computationally expensive but provides finest grain detail on neuronal function.

As [Zhang et al., 2023](#Zhang_2023) has illustrated (see Figure 1 in that paper), spike sorting is gaining traction as a fundamental process to BMI systems with the number of spike sorting publications increasing exponentially since the 1950s. However, spike sorting also has its drawbacks. Even putting aside the added computational complexity, spike sorting is an added process that typically requires rounds of training for development of classification/clustering models. Furthermore, as [Zhang et al., 2023](#Zhang_2023) points out in some of their descriptions for the various spike sorting algorithms (for example, K-means, Spiking Neural Networks, Template Matching, etc.), this development can require manual calibration. This limits actual time to application (for example, controlling a prosthesis).

### Multi-Unit Data

An alternative approach to spike sorting is using multi-unit spike detected data. In multi-unit data, each electrode is essentially treated as a single neuron with all the spikes lumped together into a single binned dimension in the neural measurement that feeds the decoder, . There has been recent work aimed at skipping the conventional spike sorting process and in testing the feasibility of multi-unit data decoding performance (for example, [Chestek et al., 2011](#Chestek_2011), [Todorova et al., 2014](#Todorova_2014), and [Trautmann et al., 2019](#Trautmann_2019)). [Trautmann et al., 2019](#Trautmann_2019) reproduced the results from three separate spike sorting publications, but instead of spike sorting, used multi-unit data for decoding and demonstrated that the results were very similar to the original spike sorted case. Their conclusion was that multi-unit data can be especially effective when decoding activity is reliant on population neural data as opposed to single neurons.

Spike sorting adds an additional layer of complexity to the BMI chain, which can make multi-unit more favorable to some applications. This complexity will also scale with the number of electrode channels, which can be an issue with spike sorting as the number of electrodes employed in recent BMI studies are reaching the thousands ([Musk and Neuralink, 2019](#Musk_2019); [Steinmetz, 2020](#Steinmetz_2020)). Added complexity comes with the demand for more powerful computational resources, which comes with added size, power, and thermal requirements. For embedded/real-time applications, this may prove non-feasible based on the inherent biological requirements at hand—again, making a case for multi-unit processing.

### Neural Decoding

Extensive research in the BMI field is aimed at decoding neural activity with the intent to translate that into or predict a certain action or task performed by an animal. “Decoding” is the process of deciphering what the neurons are ‘thinking’ about with respect to a particular task. It is common to see a research effort directed at decoding an action involving an arm reach, finger movement, or some other bodily kinematic state. This type of research is vital to the development of BMI systems targeted for prosthesis. Specifically, in this effort, the decoding will involve the trajectory (or the position, velocity, and acceleration) of a fingertip from a monkey performing reaches to targets in space.

In the past, researchers employed linear models trained with regression to do this decoding. Later models used more sophisticated probabilistic linear filters—primarily a variant of the Kalman Filter—to do this. Contemporary approaches allow for the neural-kinematic model to have non-linearities, be non-Gaussian, and for the training be unsupervised. Specifically, one of the more recent filters introduced into the BMI field for this contemporary style of modeling is the recurrent exponential-family harmonium (rEFH). In 2018, a research paper was published that introduced the rEFH used in this regard and compared performance amongst the different existing type of filters/modeling methods mentioned here ([Makin et al., 2018](#Makin_2018)). The [Makin et al. 2018](#Makin_2018) work pitted a decoding algorithm based on the rEFH against conventional decoders (for example, Linear Regression, Kalman Filters, Wiener Filters, etc.) in decoding monkey fingertip kinematics. The authors demonstrated that, over the average of 49 trials and amongst two separate monkeys, an algorithm based on rEFH outperformed every other decoding algorithm in prediction of each kinematic state (that being, lateral-position, velocity, and acceleration and longitudinal-position, velocity, and acceleration). Performance evaluation was determined from signal-to-noise ratio () and coefficient of determination ().

The proposed work in this Thesis will build from the work of [Makin et al., 2018](#Makin_2018), adapting the results from their published dataset ([O’Doherty et al., 2020](#ODoherty_2020)). As with [Makin et al., 2018](#Makin_2018), the focus will be on evaluating the rEFH and conventional decoders. The implementation for these decoders will be detailed further in the Methodology section (specifically, section 4.2.2). However, prior to moving onto that section, two final concepts in decoding should be addressed. The first is the concept of decoding from unsupervised vs. supervised datasets. The second is the training step involved in decoding.

Supervised datasets have labels associated with the states to be estimated. This allows for a model to be built that can establish a direct mapping from measurement data (inputs) to the action or state that is desired to be estimated (outputs). In the case of neural to kinematic state decoders, this corresponds to having ground truth kinematic measurements available at the time of developing the model. However, this is not always feasible as this requires a calibration round where ground truth data is collected for various states of the system. This calibration may not be possible, especially when considering the decoding be applied in a system involving prosthetic control and the state be a trajectory of a body limb that a subject does not have control of (that is, the subject has a disability). In the case of unsupervised datasets, only the measurements, or inputs to the system, are available and the states that are to be estimated are not visible. This usually involves an intermediate step of going from measurements directly to a “latent” state and, by means of a pre-determined model, going from “latent” state to the actual state of the system (for example, the fingertip trajectory).

Finally, “training” a decoder entails solving for parameters of the model employed in decoding. In training, neural decoders, data is collection on sessions, exclusive for training, where subjects are precisely instructed on what actions to perform or imagine performing. This is termed “calibration”. The calibration step is not only necessary for building the parameters of the decoding model, but also in helping the decoders converge to an optimal solution faster. For example, in the case of a Kalman filter, which optimizes by dynamically weighting a measurement vs. a prediction from a model, training is a crucial step for developing an accurate model for a system—especially when the measurements are noisy. Training can involve supervised or unsupervised datasets, but following training, decoders are used exclusively to estimate states and decoders will only have the measurements/input data available for predicting states.

### Data Collection Issues

As mentioned in the Data Acquisition Overview section (section 1.2.1), at any point in the data pipeline there could theoretically be wireless communications. [Luan et al, 2020](#Luan_2020) review various wireless BMI interface research and go into more detail on these interfaces. With wireless communications, there is always the threat of losing or dropping data or receiving erroneous data. This could be caused by ranging issues, transmission loss through different mediums, electro-magnetic interference, etc. Whatever the case, wireless dropouts or corruption could affect decoding results if vital neural spiking data is not received or received in error at update time.

In addition to inherent wireless communication problems, there is also the chance that data on an electrode gets corrupted. This could be due to an electrode detaching from its amplifier and intermittently streaming data ([Bod et al, 2022](#Bod_2022)) or shorting, etc. [Swindale and Spacek, 2016](#Swindale_Spacek_2016) conducted a study where they were able to survey multichannel electrode arrays and detect issues in integrity automatically. They demonstrated an improved spike sorting process by masking non-functional channels (or those that appear to be shorted, open circuit, or corrupted in large noisy amplitudes). Therefore, consideration should be given to the possibility of receiving imperfect data when decoding neural activity.

# Research Objectives

As mentioned in section 1.2.4, the neural decoding concepts from [Makin et al., 2018](#Makin_2018) will be adapted in this work. Specifically, four of the eight decoders from the [Makin et al., 2018](#Makin_2018) work will be re-implemented here in a custom Python library. That research effort is appealing for several reasons. For one, it is a relatively recent publication, being about five years old at present. Secondly, it comes with a published dataset ([O’Doherty et al., 2020](#ODoherty_2020)), which includes 46 experimental collections, trials from two separate monkeys, and trials with dates spanning almost 11 months. Thirdly, the authors made their code suite public, which could be used to implement their analysis and results ([Makin & O’Doherty, 2018](#Makin_ODoherty_git_2018)). Finally, [Makin et al., 2018](#Makin_2018) was able to show compelling evidence (in their comparisons of typical BMI decoders) and make a case for a new unconventional BMI decoder, which warrants further investigation.

[Makin et al., 2018](#Makin_2018) used spike sorted data to feed their decoders and ultimately reported results for that circumstance. This Thesis will take a different approach and investigate the performances when the decoders are fed unsorted, or multi-unit, neural data. At the present moment, there doesn’t appear to be any published works that make comparison of the conventional decoders to the rEFH when fed with multi-unit data, making this an attractive concept to explore.

Additionally, [Makin et al., 2018](#Makin_2018) performed training (within trial) prior to decoding for all their 49 experimental trials. Training is yet another step that delays the time until actual decoding can be applied and doesn’t support an “out of the box” solution, in which a BMI system could be used at any time and by any subject to perform a task with high accuracy. The concept of having a BMI system work across subjects and/or for different days without the need to train/re-calibrate decoders support the idea of transfer learning. The [O’Doherty et al., 2020](#ODoherty_2020) dataset features two different subjects and data collected across different days. How well transfer learning performs for the four decoders for both single-unit and multi-unit data would be another avenue worth exploring.

Finally, as mentioned already, BMI systems, just like any other electrical/computing system, are subject to faulty, corrupted, or missing data. This could be due to many circumstances such as lossy or spotty wireless communications at some stage in the data acquisition pipeline. This could also be due to faulty electrodes. As mentioned in section 1.2.5, there is work aimed at detecting faulty electrodes. In the case of faulty electrodes, there spikes could potentially be purposely omitted if errors arise during spiking. Therefore, it is worth investigating how well the four decoders perform when the neural data is manipulated such that actual detected spikes are excluded (simulating wireless dropouts or intentional omissions).

For this research effort, several goals are being proposed:

1. Reproduce the algorithms and results from [Makin et al., 2018](#Makin_2018) and develop a custom Python library to do so. This should provide the necessary tools to achieve the rest of the goals to follow and allow so conveniently. The authors’ original code was written in MATLAB, not written to reproduce the results of their paper, and not well documented. A well-documented Python implementation can save future researchers time in implementation, if they so wish to continue any efforts or explore any other outlets from the [Makin et al., 2018](#Makin_2018) work.
2. Quantify algorithm performances in and when the data is multi-unit and there are no labels associating neuron “spikes” to any individual neuron.
3. Explore the concept of transfer learning, in which a quantified assessment (in and ) is made on how well different algorithms can be trained from just one experimental trial and then used to decode different trials. Transfer learning will be evaluated for both single and multi-unit configurations.
4. Finally, performance will be quantified (again, in and ) among the algorithms as they are introduced to more and more neural spike dropouts, or missing spikes.

The overall goal is to quantify how robust different decoding algorithms are when fed imperfectly collected data. A final note here, just as [Makin et al., 2018](#Makin_2018) has done, goals 2-4 will also involve evaluating performance for different bin sizes (that being 16, 32, 64, and 128 milliseconds).

# Methodology

In this section, the different neural to kinematic state decoding algorithms will be discussed as well as the different formats of data that are expected to feed these decoding algorithms in this research effort. Prior to introduction of the different data formats and algorithms, it should be noted that variables are defined in section 5.2. The reader is encouraged to refer to that section while reading this Methodology section.

## Data Formats

### Original Data Format and Binning Concept

The provided dataset contains periodic ground truth measurements of the fingertip position, and , for monkeys performing arm reaches in various trials. The reaches were performed in the zone just below shoulder level and position was recorded just for the and axes as defined in Figure 2. During the experimental trials, just fingertip position was recorded and so to get the velocity and acceleration in the and directions (, , , and respectively), the first and second derivatives were taken with respect to sample time to get these additional kinematic states respectively and complete the kinematic vector, . The original sample time for these fingertip position measurements (and thus other kinematic states) is 250 Hz. As Figure 2 illustrates, not only are fingertip kinematics sampled, but at the same time, electrode recordings are made, and spikes are detected. The neural spiking times are what is ultimately acquired, after further processing, from the electrode recordings. These spiking events however can occur at any time, which makes this neural information asynchronous. As an example, see Figure 3 which shows a snippet of spike sorted events for each neuron. In the figure, the neurons are offset vertically, and each have markers along its horizontal to indicate when that neuron activated. As can be seen in the figure, the neuron firing events do not exhibit periodic behavior (that is, they are asynchronous in nature). This isn’t conducive to decoders that work and update state predictions at periodic intervals. To convert this neural data into synchronous data, the spike data is “binned” for each neuron. In binning, a fixed time window is defined, on which the number of spike events that occur in that window are counted. The total test is divided up into time windows/intervals, where the spike events are counted on. Instead of the neural data being comprised of spike times, it is now spiking counts, , that can be aligned with a time vector. Just as [Makin et al., 2018](#Makin_2018) has done, the bin window sizes tested in this effort will be 16, 32, 64, and 128 milliseconds. A final note is that neurons that fired less than 0.5 Hz were dropped from the dataset to keep consistency with [Makin et al., 2018](#Makin_2018).

A screen shot of a black background

Description automatically generated

Figure 2. The axes defined for fingertip kinematics from Makin et al., 2018. Reaching tasks were performed in the x-y plane to hit targets with a fingertip. +x corresponded to reaches to the right of the subject and +y corresponded to reaches rostral to the subject.

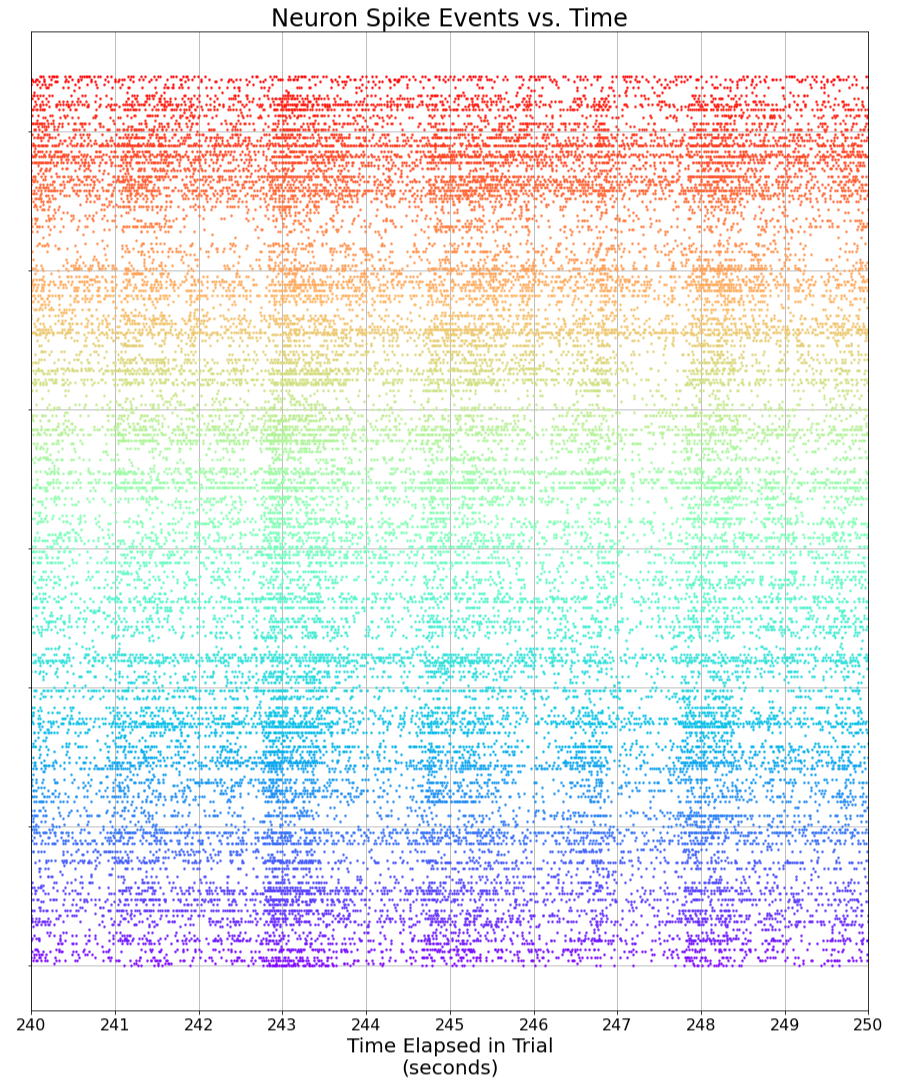


Figure 3. This plot is over a snippet of time for a trial in the [O’Doherty et al., 2020 dataset](#ODoherty_2020). In this plot, each spike sorted neuron is offset vertically. The markers across the horizontal indicate when a spike detected event occurred for a neuron corresponding to the vertical position of those horizontal mark sets.

### Multi-Unit Data Format

As mentioned already, a goal is to exercise the four decoders of interest using multi-unit neural data. The dataset at hand already associates which spike sorted times come from which neuron and which electrode those neurons go to. Therefore, to simulate multi-unit data, simply pool all the neuron spike times from each electrode into one big array. This will then treat each electrode essentially as a single neuron. This ultimately changes the Figure 3 picture to that seen in Figure 4. As can be seen, the spiking event vectors (along the horizontal and differing in color) are more spread out (along the vertical axis) and less dense, effectively reducing the dimensionality of the neural dataset. Following this dimensionality reduction, continue with binning and decoding as usual for the pooled spike times for each electrode.

A rainbow colored lines on a white background

Description automatically generated

Figure 4. This is the effect of converting the spike sorted data in Figure 3 to multi-unit data. A dimensionality reduction essentially occurs here.

### Channel Dropping Format

To simulate and test the effect of channel dropping, the spike occurrences will be decreased at random. That is, given a number of spikes to be dropped, call it , a combination of pairs of bin time slots and electrodes will be randomly selected, with replacement, for where spike counts are to be decreased. This random selection will be done uniformly, that way channel dropouts are equally likely to occur at any point in time during data collection.

## Data Processing

In [Makin et al., 2018](#Makin_2018), there are eight total neural to kinematic state decoding algorithms being compared. Considering the limited timeframe given to complete this research effort, only four of the eight decoders from that paper will be implemented and tested in this research effort, namely:

1. Linear Regression
2. A Supervised Kalman Filter ([Wu et al., 2003](#Wu_2003))
3. An Unsupervised Kalman Filter (Modeling adopted from [Wu et al., 2003](#Wu_2003) with parameter training adopted from [Ghahramani and Hinton, 1996](#Ghahramani_Hinton_1996))
4. The Recurrent Exponential-Family Harmonium ([Makin et al., 2018](#Makin_2018))

### Transfer Learning

Prior to deploying the decoding algorithms for prediction, there’s generally a training step that is either required or done out of necessity to either initialize the decoder or aid in its convergence to an optimal solution. The training is performed on data for a defined training interval (for example, ). In this project, just like in [Makin et al., 2018](#Makin_2018), the data from an experiment is partitioned into a training subset and test subset, with the training data consisting of that acquired in the first 320 seconds of an experiment and the test data being everything following that. The test data is where the performance of an algorithm is evaluated on its intended purpose—that is, how well its model makes predictions. In [Makin et al., 2018](#Makin_2018), this procedure of training followed by testing was carried out for every new experiment/data file. This approach will also be followed in this research effort except for when evaluating the performance of the algorithms in transfer learning.

In the [O’Doherty et al., 2020 dataset](#ODoherty_2020), there is data collected from experiments on 46 different days, spanning almost 11 months, and collected from two different monkeys. Therefore, this dataset should be a good candidate to exercise the transfer learning concept. To do that, an arbitrary data file from one of the experiments will be selected, each algorithm will be trained solely from that data file, and then the decoding algorithms will be applied to all the test subsets in all the files and an evaluation made on prediction performance.

### Decoding Algorithms

#### Linear Regression

Of all the models that will be tested, the simplest will be a linear regressive model, which will serve as a baseline and map the binned neural spike counts to the kinematic state(s) by means of a linear transformation. This can be described mathematically as follows:

|  |  |
| --- | --- |
|  | (1) |

In equation (1), the coefficients are the weights for the spike count of the respective neural recording channel at time , or . Those coefficients assign contribution that a respective neural channel has towards each dimension of the kinematic state, . The combination of those contributions for every neural channel transforms the neural data to the space associated with the kinematic state(s). The offset vector, , then simply scales the combinations of those linear contributions appropriately to compensate for any offset and complete the regression of onto its equivalent state in a linear sense. To solve for the parameters of best fit in this model, first, an overdetermined system of equations should be formulated from observations:

|  |  |
| --- | --- |
|  | (2) |

Equation (2) builds on equation (1), with variables and being matrices with successive observations in time filling out their rows. As mentioned in section 3.2.1, the samples collected here for this overdetermined equation will be that from the training set. The parameters will then be solved for on that period, following an ordinary least squares approach. In applying ordinary least squares, a “best fit” is had by finding the parameters that minimize the sum of squares of the error in the estimation of the kinematic state(s), (with ). Following through with the math and this optimization problem, ordinary least squares will arrive at the following equation for the parameters of “best fit”:

|  |  |
| --- | --- |
|  | (3) |

Going forward, Equation (1) is then applied at each time step to compute the best linear kinematic prediction from the measured neuron firing counts, , and the parameters found from the training interval.

#### Kalman Filter – Supervised

The next featured model to be discussed is the widely adopted Kalman Filter. As Kalman Filters are intended for linear-Gaussian dynamical systems (LGDS), it is typical to model the neural to kinematic system as such. Therefore, for this model and—as first introduced by [Wu et al., 2003](#Wu_2003)—the system will be expressed as such, with the following set of state-space equations:

|  |  |
| --- | --- |
|  | (4) |
|  | (5) |

Equation (4) evolves the kinematic state from one time step to the next. Equation (5) is the generative model of the neuron spike counts (that is, it relates the current kinematic state to its equivalent neuron spike counts by means of a linear transformation). In both equations here, there are noise terms which are assumed Gaussian and 0 mean (that is, and ). This model thus has four parameters that must be solved for prior to applying the Kalman Filtering algorithm: , , ,. Solving for these parameters is what entails the training session interval as mentioned in section 3.2.1. Since equations (4) and (5) are linear with 0 mean noise, the transformation matrices ( and ) which transform the independent states to the dependent states, can be solved for using the ordinary least squares approach as explained in the previous section (section 4.2.2.1). To be clear, the process will be to first formulate an overdetermined system of equations in equations (4) and (5) by appending all successive time samples in the training set onto the and measurements as new rows. Then, apply ordinary least squares and use equation (3)—substituting the parameters or with , with , and or respectively with . Then, from [Wu et al., 2003](#Wu_2003), the variance terms can be found with the following equations:

|  |  |
| --- | --- |
|  | (6) |
|  | (7) |

Following training, the Kalman Filter will be applied iteratively over the rest of the test set(s). This iterative process can be outlined in the following way:

Make a prediction on the initial kinematic state,  , and the initial uncertainty of that prediction, . As done in [Makin et al., 2018](#Makin_2018), these will be set to the average and variance of the kinematic state at every time step in the training data respectively.

Take a measurement at the current time step, , and acquire the latest neural binned counts, .

Posteriori/Update step—that is, with the latest measurement and prior knowledge (prediction), make an informed estimate of the kinematic state and uncertainties. This is driven by a weight term (Kalman Gain, ) that dynamically updates at each iteration and allows for the estimates to be a weighted combination of the prediction and solution from the measurement. The equations solved at this step are the Kalman Gain Equation, the State Update Equation, and the Covariance Update Equation and, in that order, these are as follows:

|  |  |
| --- | --- |
|  | (8) |
|  | (9) |
|  | (10) |

Predict the next kinematic state and estimate uncertainty in that prediction (a priori). At this step, two equations are solved to compute the predictions for both the estimate of the kinematic state at the next time step and the uncertainty in the estimate at the next time step. One of the equations, the State Extrapolation Equation, has already been introduced and that is Equation (4). However, in this case now, becomes , becomes , and need not be considered in evaluation of the next state prediction, , as it is 0 mean (no bias). The second equation is called the Covariance Extrapolation Equation and is expressed as follows:

|  |  |
| --- | --- |
|  | (11) |

Finally, repeat steps 2-4 throughout in real time data acquisition.

Before moving on to the next decoding algorithm featured in this project, it should be noted that the training method described here makes this decoder a “supervised” decoder. That is because the kinematic measurements are made available in the training dataset and applied directly in solving for the LGDS parameters.

#### Kalman Filter – Unsupervised

The next decoder is, again, a Kalman Filter, but with a difference in how the LGDS parameters are trained in the model and in which state of the system the Kalman Filter is estimating. In this decoding method, the decoding is done assuming there is no access to kinematic state measurements and with inference of the existence of an intermediate latent state, . The neural data maps directly to the latent state and that, in turn, allows for recovery of the kinematic state. This is an “unsupervised” Kalman Filter model.

The first step in implementation of this decoder is solving for the parameters of the LGDS (equations (4) and (5)). These parameters are , , ,. This is done by following “The EM Algorithm” section step for step in [Ghahramani and Hinton, 1996](#Ghahramani_Hinton_1996). This is an Expectation Maximization (EM) algorithm, which iterates between an expectation and maximization step. In expectation, the expected latent variables are generated given the latest model parameters. In maximization, the model parameters are updated to maximize the likelihood of the model (that is, that the generated data came from that model). Prior to performing the EM algorithm, latent states, , ([Ghahramani and Hinton, 1996](#Ghahramani_Hinton_1996) refer to this as ) as well as their covariance matrix, , and the covariance matrix between the previous () and current () latent state () must be initialized. and are initialized with factor analysis by following section 2 of [Ghahramani and Hinton, 1997](#Ghahramani_Hinton_1997) (Note that to conform to [Makin et al., 2018](#Makin_2018), the number of latent states is set to the number of neural observations ()). matrix was arbitrarily initialized to a diagonal matrix of ones.

These initialized latent states and covariance matrices are then passed directly into the M step of EM and used along with observational data (that is, the neural data, ) to update parameters as shown in [Ghahramani and Hinton, 1996](#Ghahramani_Hinton_1996). Following the M step, the E step is conducted as shown in [Ghahramani and Hinton, 1996](#Ghahramani_Hinton_1996) (refer to the appendix in [Shumway and Stoffer, 1982](#Shumway_Stoffer_1982) for additional clarity on this). This EM algorithm (as well as the factor analysis initialization previously done) can be run until convergence, which following [Makin et al., 2018](#Makin_2018), corresponds to 100 iterations or until a “cross entropy decrease less than of a percent of the total decrease [since starting the algorithm]”.

The EM algorithm will give latent states, , and model parameters. Although, we excluded kinematic states in training the model, the kinematic states will be used to compute a linear transformation that transforms the latent states to the kinematic states. This process is done here, but theoretically, this transformation matrix could be prior knowledge. To obtain the transformation matrix, linear regression (as explained in section 3.2.2.1) will be used.

Finally, the Kalman filter can be applied as described in the previous section, but with using the initial latent states, to initialize instead of the initial kinematic states. This Kalman Filter result will give the predicted latent states at each time step. From here, just convert the estimated latent states to the estimated kinematic states using the linear transformation matrix that was solved for here with linear regression.

#### Recurrent Exponential Family Harmonium (rEFH)

As of now, the recurrent exponential family harmonium decoder implemented by [Makin et al., 2018](#Makin_2018) has yet to be implemented or fully understood. This section will be filled out later after the methodology for the decoder is worked out and further investigated. For now, refer to section 5.2 for future work on this effort.

## Decoder Performance Evaluation

To stay consistent with [Makin et al., 2018](#Makin_2018), the same metrics will be used to quantify performance. That is, Coefficient of Determination, or , and signal-to-noise ratio, or . These are defined with the following equations, respectively:

|  |  |
| --- | --- |
|  | (12) |
|  | (13) |

# Implementation

A graph with blue and orange lines

Description automatically generated

Figure 5. This is an example snippet of actual estimation results when compared to ground truth fingertip kinematic data. In this specific example, the decoder used was the Kalman Filter - Supervised and this is on data file indy\_20160407\_02.mat from the [O'Doherty et al., 2020](#ODoherty_2020) dataset.

## Preliminary Work

Thus far, three of the four algorithms have been implemented into a custom python library. These three algorithms are linear regression, Kalman Filter – Supervised, and Kalman Filter – Unsupervised. In addition to these decoder algorithm implementations, a binning and scoring function has also been implemented. Figure 5 shows actual results from an implemented Kalman Filter – Supervised decoder. In the figure, the actual ground truth fingertip kinematic states (position, velocity, and acceleration in the and directions), as well as their “estimates” (from the Kalman Filter – Supervised decoder) are overlayed on a time snippet of a trial. From a qualitative standpoint, the estimates fit the ground truth data fairly well, matching its overall shape.

Now, to convey how well these decoding algorithms were re-implemented in this effort, a comparison amongst the performance metrics is made—specifically, the metric. Figure 6 and Figure 7 show the distribution of computed on the result of decoding 888 different trial/bin size combinations. Figure 6 compares the custom Linear Regression implementation to the original [Makin et al., 2018](#Makin_2018) implementation and, likewise, Figure 7 does the same for the Kalman Filter – Supervised implementations. As can be seen in those figures, the overall shape of the distributions for the custom implementation matches nicely to the original implementation, but with small differences in counts for some bins. This suggests that the custom implementations are performing as expected and having the same effect intended by the original, but that some subtlety between the original and custom implementation exists.

To quantify exactly how different the decoders are, Table 1 and Table 2 are provided, where percent differences between the two different implementations (for linear regression and Kalman Filter – Supervised, respectively) is computed. The average s from the two implementations, and when considering all possible binwidths (that is 16 ms, 32 ms, 64 ms, and 128 ms), ranges from ~0-6%.

One final remark, the Kalman Filter – Unsupervised algorithm has been implemented, but due to its lengthy compute time, the results for all the trials and possible binwidths have not yet been exercised for it. Though, one file has been processed for a 16 ms binwidth. The resulting kinematic state estimates are overlayed with the actual in Figure 8 for this decoder. As an initial assessment, this decoder seems to produce noisier estimates than its counterpart, the Kalman Filter – Supervised (Figure 5). The verification of this decoder will be left for future work.

A comparison of a graph

Description automatically generated

Figure 6. This is a comparison of the SNR performance metric results between the Linear Regression Decoder implemented by [Makin et al., 2018](#Makin_2018) and the custom python implementation from this work. Note that this distribution is formulated by running the decoder on trials in the dataset with the binwidth (16 ms, 32 ms, 64 ms, and 128 ms) being swept through for each trial. For ease of comparison, each histogram in this figure has the same bins and the respective SNR counts for each bin is placed on top of the bin.

A graph of different sizes and numbers

Description automatically generated with medium confidence

Figure 7. This is a comparison of the SNR performance metric results between the Kalman Filter - Supervised implemented by [Makin et al., 2018](#Makin_2018) and the custom python implementation from this work. Note that this distribution is formulated by running the decoder on trials in the dataset with the binwidth (16 ms, 32 ms, 64 ms, and 128 ms) being swept through for each trial. For ease of comparison, each histogram in this figure has the same bins and the respective SNR counts for each bin is placed inside its bin.

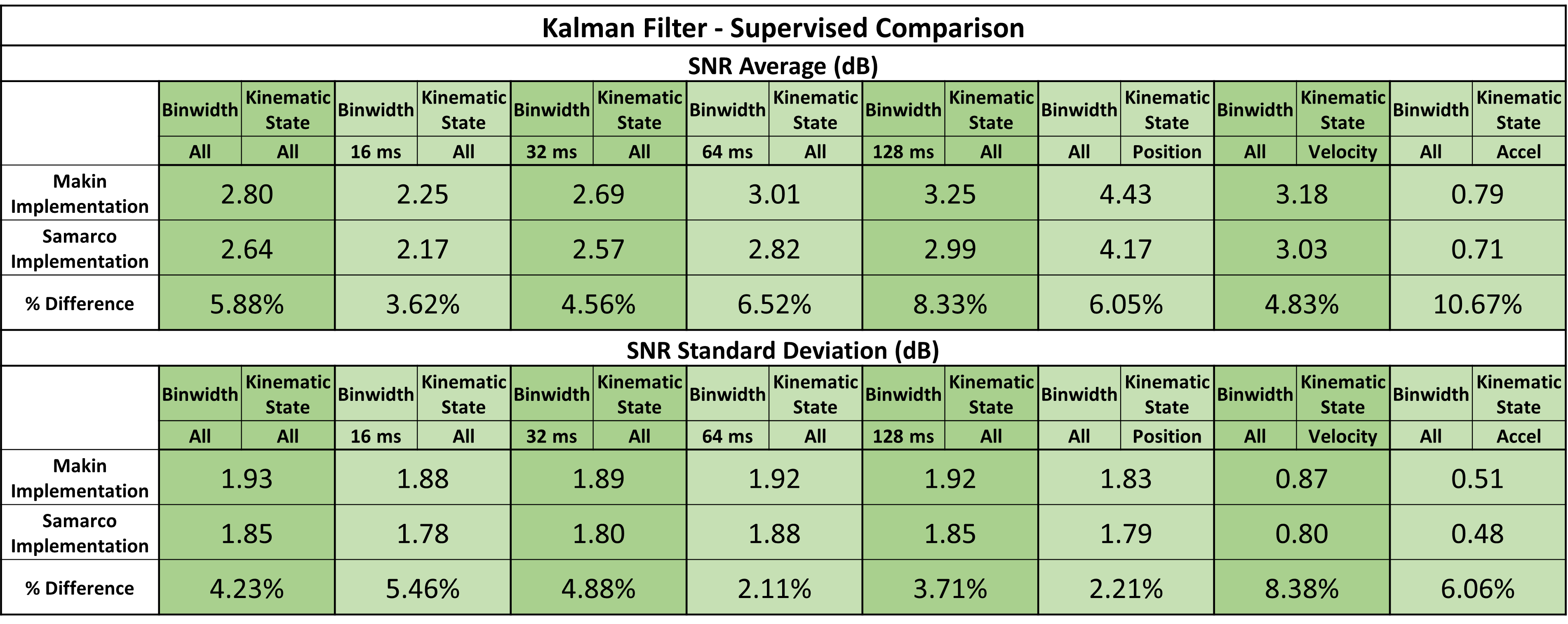


Table 2. The differences in SNR results among the custom Kalman Filter - Supervised implementation and the [Makin et al., 2018](#Makin_2018) implementation are displayed here. The average SNR and its standard deviation are computed for both implementations and for different bin sizes and different kineamatic states.

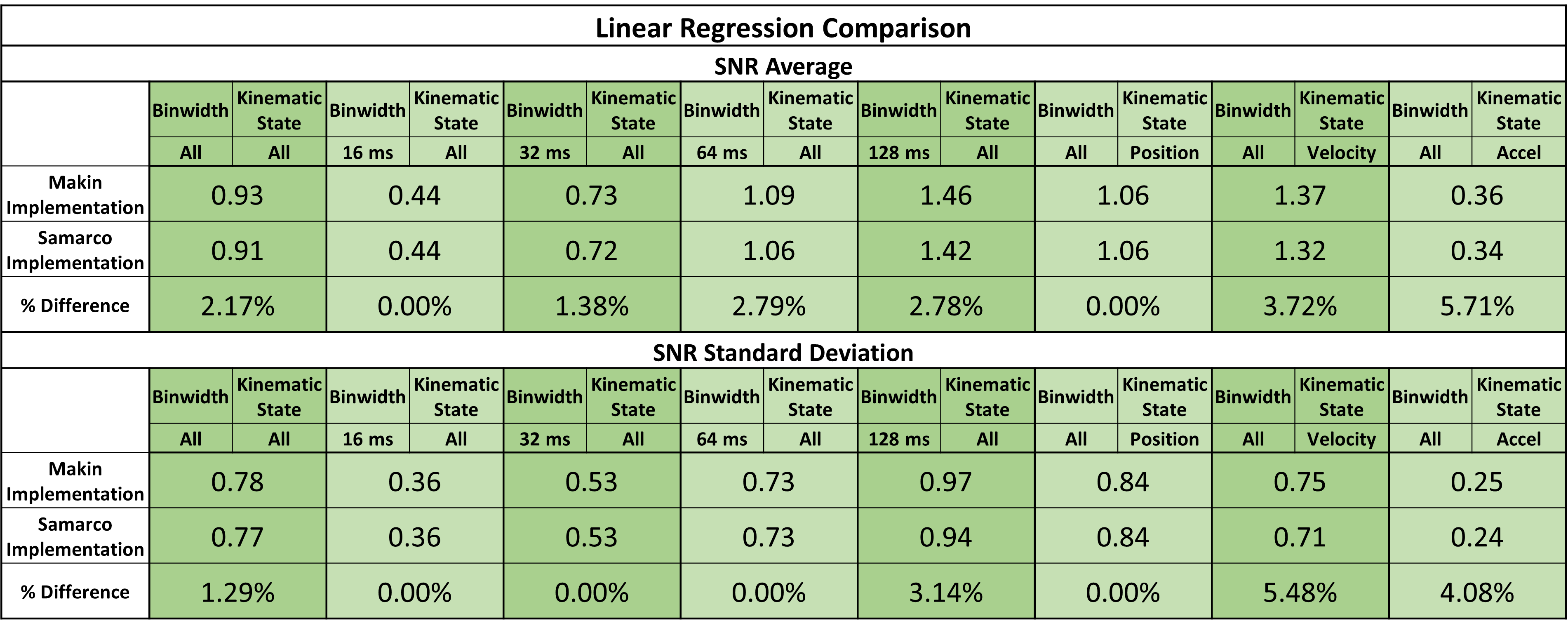


Table 1. The differences in SNR results among the custom linear regression implementation and the [Makin et al., 2018](#Makin_2018) implementation are displayed here. The average SNR and its standard deviation are computed for both implementations and for different bin sizes and different kineamatic states.

A graph of blue and orange lines

Description automatically generated

Figure 8. Decoder Results from running a custom implementation of the KF - Unsupervised.

## Proposed Work

Future work will be aimed at completing several tasks. Specifically:

###### ***Task 1: Validate the Kalman Filter – Unsupervised Implementation***

As stated in the previous section (section 4.1), the issue with the custom implementation of the Kalman Filter – Unsupervised decoder is that its execution time, or time until results, takes several hours to process just one trial file with one bin size applied. There are over 800 different combinations of trials and bin sizes to test. Therefore, running this decoder for all the possible file/binwidth permutations will take substantial time until results. To get over this hurdle, parallel processing will need to be applied. One way to do this is to make use of a GPU device. Another way to do this, is to make use of multi-threaded processing on a computer.

Most of the execution time in the current Python implementation of the Kalman Filter – Unsupervised decoder is spent in EM training, which iterates until convergence, or up to a maximum of 100 iterations. On each iteration, several matrix operations are computed, and a couple nested loop operations are carried out as well. The matrix operations are currently implemented using relatively fast (when compared to looping) single core processor functions from the Python package “NumPy”. The matrix operations could be sped up significantly with use of the Python “CuPy” package. “CuPy” works as a drop-in replacement for many “NumPy” functions and makes use of a GPU (if available) to parallelize matrix operation. The custom implementation of the Kalman Filter – Unsupervised decoder will be modified to use the “CuPy” package.

If “CuPy” does not significantly increase the execution time for this algorithm, the next step will be to write custom bash code to pass all the possible different parameter combinations to this decoder and dispatch these as parallel jobs on a compute cluster. This should essentially allow results to be had with execution time corresponding to the trial/binwidth combination that takes the longest to process.

After acquiring results for spike sorted data for all possible binwidth combinations on each trial, the distribution of results will be compared to that from [Makin et al., 2018](#Makin_2018) to see how well the two implementations match. Additionally, this custom implementation will be quantified by percent difference in results. This will conclude validation for the custom implementation of the Kalman Filter – Unsupervised decoder.

###### ***Task 2: Implement the rEFH Decoder***

For learning the theory behind rEFH and how to implement it, [Makin et al., 2016](#Makin_2016) seems to be the place to start. This was referenced in the [Makin et al., 2018](#Makin_2018) effort and this paper has a section titled “theory” where the authors go into how the exponential-family harmonium is extended to time, creating the rEFH. After implementing this decoder, comparison between [Makin et al., 2018](#Makin_2018) and this Thesis’ implementation will be validated via distribution analysis and percent differences in the metric (as done in the previous section, 4.1 and mentioned in the previous task).

###### ***Task 3: Quantify the Performance of the Neural Decoders in Handling Multi-Unit Data***

The neural data from the [O’Doherty et al., 2020](#ODoherty_2020) dataset is published with neuron spike times already sorted to neurons, which are in turn, sorted to electrodes. To undo this sorting, all the spike times for every neuron belonging to an electrode will be grouped together so that the data reflects spike events for each electrode (see section 3.1.2 for more on this). This multi-unit format becomes the variable (seen in equations in section 3.2).

All the 49 data files/individual trials from the [O’Doherty et al., 2020](#ODoherty_2020) dataset will be trained and decoded in the same fashion that the spike sorted data was in [Makin et al., 2018](#Makin_2018) (with the only exception being that multi-unit data now takes on the variable ). This decoding will be performed for the different binwidths (that is, 16 ms, 32 ms, 64 ms, and 128 ms). The performance amongst the decoders will be evaluated as follows:

* Quantify decoder improvement from baseline decoder, or the linear regression decoder, for each kinematic state (, , , , , ) when the multi-unit data is binned at 64 ms (the main experiment for [Makin et al., 2018](#Makin_2018)). Determine the 95 % confidence intervals for the improvement in terms of and .
* Determine the significance level of the improvement in from the baseline decoder for handling 64 ms bins and on decoding each kinematic state.
* Quantify the 95 % confidence interval for decoder for the different binwidths on the different kinematic types (position, velocity, acceleration).
* Determine the significance level at which any decoder outperforms others for any binwidth/kinematic type combination.

These confidence intervals will be found using bootstrapping (or resampling) from the original performance computations ( and ) on the kinematic states for each of the 49 trials. The bootstrapping will produce a distribution of 100,000 sample means of the metrics (that is, , , difference in between two decoders). Each of the resampled performance metrics for each trial will be weighted by their trial’s number of test samples when computing a sample mean (just as [Makin et al., 2018](#Makin_2018) did).

###### ***Task 4: Quantify the Performance of the Neural Decoders in Transfer Learning***

To test transfer learning, the original spike sorted data from the [O’Doherty et al., 2020](#ODoherty_2020) dataset will be used. 48 and metrics will be computed, with each being the average of different cases of one file/trial being held out for training and the other files being used for evaluation. These averages will be weighted by number of time samples for respective “test” files. That process will be repeated with each file/trial taking a turn as the “train” file. Training will be applied to only the first 320 seconds of the “train” file to coincide with [Makin et al., 2018](#Makin_2018) and avoid overtraining. This will be done separately for each binwidth (16 ms, 32 ms, 64 ms, and 128 ms). The performance amongst the decoders will be evaluated as follows:

* Quantify the 95 % confidence interval for decoder for the different binwidths on the different kinematic types (position, velocity, acceleration).
* Determine the significance level at which any decoder outperforms others for any binwidth/kinematic type combination.

As in the previous task, bootstrapping (100,000 times) the 48 or differences (differences is for comparison of decoders) will make this quantification possible.

###### ***Task 5: Quantify the Performance of the Neural Decoders in Handling Dropped Data***

In simulating dropped data, only spike sorted data from the original dataset will be used. At random, spikes will be removed from bins, with replacement [of that bin], throughout a data file. The number of spike events to be dropped will be a percentage of the total spike events in a data file. The percent of spikes that will be dropped and tested will be 5%, 15%, 25%, and 50%. The decoders performance to coping with these missing spikes will be evaluated as follows:

* Quantify the 95 % confidence interval for decoder for the different binwidths, on the different kinematic types (position, velocity, acceleration), and for the different cases of number of spikes dropped.
* Determine the significance level at which any decoder outperforms others for any binwidth, kinematic type, and percent spike drop combination.

As with the previous two tasks, these quantifications will be computed using bootstrapping (100,000 times) on the 49 trial results for a given binwidth, kinematic type, and percentage of spikes to be dropped.

###### ***Task 6: Complete Documentation of Custom Python Library and Publish Code Online***

The custom Pythonfor this Thesis work The current code has been made into a Git repository. This effort will continue throughout in completion of the tasks above.

# Appendix

## Variable Notation

In this paper, there will be slight differences in notation for the same variable to distinguish exactly the form of the data:

|  |  |
| --- | --- |
| Designation | Designation Description |
| Hat () | The hat designation means the variable is an estimate (not actual). |
| Upper case notation () | Upper case letters represent matrices. |
| Lower case notation ( | Lower case letters represent a scalar or vector. |
| Subscript with range of samples () | A range (e.g. 0:T) in the subscript of a variable specifies the sample range that makes up the variable (e.g. 0:T means samples 0, 1, … , T). |

## Variable Definitions

The variables relevant to this paper and the code for this research effort are as follows:

|  |  |
| --- | --- |
| Variables | Variable Descriptions |
|  | Number of kinematic states. |
|  | Number of observation channels for the neural data. For single-unit data collection this is equal to the number of neurons being observed. For multi-unit data collection, this is equal to the number of electrodes being observed. |
|  | Kinematic state vector at time sample . This vector is size . |
|  | Kinematic state matrix. This is a matrix comprised of the kinematic state samples. This matrix contains successive samples in time (for example, ). This matrix is size (number of samples) . |
|  | This is the binned neural spike count for channel at time sample . This is a scalar. |
|  | Binned neural spike counts at time sample for all the collection channels. This is a vector with and of size . |
|  | This is prepended with a 1 like so . This vector is size . |
|  | Binned neural spike count observation matrix. This matrix contains successive samples in time with a column of ones prepending the observations (for example, ). This matrix is size (number of samples) . |
| , | Linear regression weight coefficient vector. This is used in the linear regressive model and assigns contribution that a neural channel has toward each kinematic state in the transformation from neural state to kinematic state. This vector is size . |
|  | Linear regression offset/intercept vector. This vector is used in linear regression to remove an offset from the transformed neural to kinematic state. This vector is size . |
|  | Regression coefficient matrix. This is the result of concatenating the vectors with the linear regression offset vector, , like so . This matrix is size . |

# Bibliography

Meng, J., Zhang, S., Bekyo, A., Olsoe, J., Baxter, B., &amp; He, B. (2016). Noninvasive

electroencephalogram based control of a robotic arm for reach and grasp tasks. Scientific

Reports, 6(1). <https://doi.org/10.1038/srep38565>

Bouton, C., Shaikhouni, A., Annetta, N. et al. Restoring cortical control of functional movement

in a human with quadriplegia. Nature 533, 247–250 (2016).

<https://doi.org/10.1038/nature17435>

Aflalo, T., Kellis, S., Klaes, C., Lee, B., Shi, Y., Pejsa, K., Shanfield, K., Hayes-Jackson, S.,

Aisen, M., Heck, C., Liu, C., &amp; Andersen, R. A. (2015). Decoding motor imagery

from the posterior parietal cortex of a tetraplegic human. Science, 348(6237), 906–910.

<https://doi.org/10.1126/science.aaa5417>

Salahuddin, U., &amp; Gao, P.-X. (2021). Signal Generation, acquisition, and processing in

Brain Machine Interfaces: A unified review. Frontiers in Neuroscience, 15.

<https://doi.org/10.3389/fnins.2021.728178>

EmotivPRO. EMOTIV. BMI Data Acquisition Software.

EPOC X. EMOTIV. Wireless EEG Headset.

Flex 2.0 Saline. EMOTIV. Wireless High Density Saline EEG Headset.

Studio, P. (n.d.). Neuralink’s first-in-human clinical trial is open for recruitment: Blog.

Neuralink. <https://neuralink.com/blog/first-clinical-trial-open-for-recruitment/>

PZ2 Preamplifier. Tucker-Davis Technologies. BMI Preamplifier.

Rajan, A. T., Boback, J. L., Dammann, J. F., Tenore, F. V., Wester, B. A., Otto, K. J., Gaunt, R.

A., &amp; Bensmaia, S. J. (2015). The effects of chronic intracortical microstimulation

on neural tissue and fine motor behavior. Journal of Neural Engineering, 12(6), 066018.

<https://doi.org/10.1088/1741-2560/12/6/066018>

Utah Array. Blackrock Neurotech. Microelectrode Array.

Zhang, T., Rahimi Azghadi, M., Lammie, C., Amirsoleimani, A., &amp; Genov, R. (2023). Spike

sorting algorithms and their efficient hardware implementation: A comprehensive survey.

Journal of Neural Engineering, 20(2), 021001. <https://doi.org/10.1088/1741-2552/acc7cc>

Chestek, C. A., Gilja, V., Nuyujukian, P., Foster, J. D., Fan, J. M., Kaufman, M. T., Churchland,

M. M., Rivera-Alvidrez, Z., Cunningham, J. P., Ryu, S. I., &amp; Shenoy, K. V. (2011).

Long-term stability of neural prosthetic control signals from silicon cortical arrays in

rhesus macaque motor cortex. Journal of Neural Engineering, 8(4), 045005.

<https://doi.org/10.1088/1741-2560/8/4/045005>

Todorova, S., Sadtler, P., Batista, A., Chase, S., &amp; Ventura, V. (2014). To sort or not to sort:

The impact of spike-sorting on neural decoding performance. Journal of Neural

Engineering, 11(5), 056005. <https://doi.org/10.1088/1741-2560/11/5/056005>

Trautmann, E. M., Stavisky, S. D., Lahiri, S., Ames, K. C., Kaufman, M. T., O’Shea, D. J., Vyas,

S., Sun, X., Ryu, S. I., Ganguli, S., &amp; Shenoy, K. V. (2019). Accurate estimation of

neural population dynamics without spike sorting. Neuron, 103(2).

<https://doi.org/10.1016/j.neuron.2019.05.003>

Musk, E. (2019). An integrated brain-machine interface platform with thousands of channels.

Journal of Medical Internet Research, 21(10). <https://doi.org/10.2196/16194>

Steinmetz, P. N. (2020). Estimates of distributed coding of visual objects by single neurons in the

human brain depend on which spike sorting technique is used. Journal of Neural

Engineering, 17(2), 026030. <https://doi.org/10.1088/1741-2552/ab6cb8>

Makin, J. G., O’Doherty, J. E., Cardoso, M. M., &amp; Sabes, P. N. (2018). Superior arm-

movement decoding from cortex with a new, unsupervised-learning algorithm. Journal of

Neural Engineering, 15(2), 026010. <https://doi.org/10.1088/1741-2552/aa9e95>

O’Doherty, J. E., Cardoso, M. M. B., Makin, J. G., &amp; Sabes, P. N. (2020, May 26).

Nonhuman primate reaching with Multichannel Sensorimotor Cortex Electrophysiology.

Zenodo. <https://zenodo.org/doi/10.5281/zenodo.788569>

Luan, L., Robinson, J. T., Aazhang, B., Chi, T., Yang, K., Li, X., Rathore, H., Singer, A.,

Yellapantula, S., Fan, Y., Yu, Z., &amp; Xie, C. (2020). Recent advances in electrical

neural interface engineering: Minimal invasiveness, longevity, and scalability. Neuron,

108(2), 302–321. <https://doi.org/10.1016/j.neuron.2020.10.011>

Bod, R. B., Rokai, J., Meszéna, D., Fiáth, R., Ulbert, I., &amp; Márton, G. (2022). From end to

end: Gaining, sorting, and employing high-density neural single unit recordings. Frontiers

in Neuroinformatics, 16. <https://doi.org/10.3389/fninf.2022.851024>

Swindale, N. V., &amp; Spacek, M. A. (2016). Verification of multichannel electrode array

integrity by use of cross-channel correlations. Journal of Neuroscience Methods, 263, 95–

102. <https://doi.org/10.1016/j.jneumeth.2016.02.009>

Makin, J. G., &amp; O’Doherty, J. (2018, November 9). Jgmakin/rbmish: Matlab code for

exponential family harmoniums, RBMS, dbns, and relata. GitHub.

<https://github.com/jgmakin/rbmish>

Wu W., Black M. J., Gao Y., Bienenstock E., Serruya M. D., Shaikhouni A. and Donoghue J. P.

(2003) Neural Decoding of Cursor Motion Using a Kalman Filter. Proc. of the 2002

Conf. Advances in Neural Information Processing Systems 15 pp 133–40.

Ghahramani, Z., Hinton, G.: Parameter estimation for linear dynamical systems. Technical

Report CRG-TR-96-2, University of Totronto (1996)

Ghahramani, Z., &; Hinton, G. E. (1997). The EM algorithm for mixtures of factor analyzers.

Technical Report CRG-TR-96-1, Department of Computer Science, University of

Toronto, 6 King's College Road, Toronto, Canada, M5S 1A4.

Shumway, R. H. and Stoffer, D. S. (1982). An approach to time series smoothing and forecasting

using the EM algorithm. J. Time Series Analysis, 3(4):253-264.

Makin J. G., Dichter B. K. and Sabes P. N. (2016). Recurrent Exponential-Family Harmoniums

without Backprop-through-Time (arXiv:1605.05799)

Rinberg, D., Davidowitz, H., &amp; Tishby, N. (1998). Multi-electrode spike sorting by

clustering transfer functions. NIPS’98: Proceedings of the 11th International Conference

on Neural Information Processing Systems, 146–152.

https://doi.org/https://dl.acm.org/doi/10.5555/3009055.3009076