Characterization of Circadian Modulation of Neuromotor Control in Mice

Oliver Johnson

A thesis
submitted in partial fulfillment of the
requirements for the degree of
Master of Electrical Engineering

University of Washington
2015

Committee:
Howard Chizeck
Horacio de la Iglesia

Program Authorized to Offer Degree:
Electrical Engineering

University of Washington

Abstract

Characterization of Circadian Modulation of Neuromotor Control in Mice

Oliver Johnson

Chair of the Supervisory Committee:
Dr. Howard Chizeck
Electrical Engineering

Decoding electrocorticographic (ECoG) signals coming from the primary motor cortex (PMC) associated with complex motor programs may represent a fundamental tool for the development of minimally invasive neural prosthetic devices. The development of analytical tools to extract this motor information and of algorithms to use this information to generate motor tasks relies on the understanding of endogenous and exogenous variance that affect the PMC. The circadian system is a predictable source of endogenous variance. The master circadian pacemaker governs overt circadian rhythms of physiology and behavior, leading to 24-hour oscillations in parameters that directly affect our ability to achieve specific motor tasks. Our goal for this study was to identify PMC electrical patterns associated with the specific motor task of initiating running and searching for distinct periodic behavior associated with a circadian period. Wheel running is a stereotypic behavior in mice that can be quantified as wheel revolutions and by changes in muscle activity measured through electromyographic (EMG) electrodes. We first demonstrated changes in the PMC signal that could discriminate running from non-running in mice implanted with both ECoG and EMG electrodes. After confirming our mouse model could show changes in behavior, we repeated the trials across a 48-hour cycle, recording at 3-hour intervals on the mouse's internal circadian clock. Analysis was performed looking at spectral power for how the model parameters would vary as the mouse performed the same behavior over its circadian cycle. We were able to show periodic changes in each model that corresponding to a circadian cycle and showed that it was likely not due to other noise or factors in the trials.

Introduction

Complex motor programs in mammals are the result of neuronal firing patterns within the Primary Motor Cortex (PMC). These specific neuronal activity patterns within the PMC can be associated with the distinct motor tasks being performed. This has been done in humans and primates by recording the firing pattern of cortical neurons with intracerebral electrodes. This information can then be decoded to generate a similar motor task in a neural prosthetic, such as controlling an artificial arm, or even adapted to move a cursor on a computer screen[1,2,3,4]. These systems constitute a Brain Computer Interface (BCI) where the electrical neuronal activity acts as the input signal to directly control an output mechanism. To avoid the severity of direct neural implants, less invasive approaches of electrocorticographic (ECoG) or electroencephalographic (EEG) recordings have been studied and used as an alternate in method for BCIs. Both ECoG and EEG measure behavior from a neural population, and have been demonstrated as a functional BCI input in humans where PMC neuronal activity patterns were extracted for a given motor task[5]. Progress using this approach has been slowed by the lack of inexpensive animal models to correlate motor tasks to ECoG activity patterns. To address this issue, we use mice to measure PMC surface ECoG recordings wheel-running revolutions.

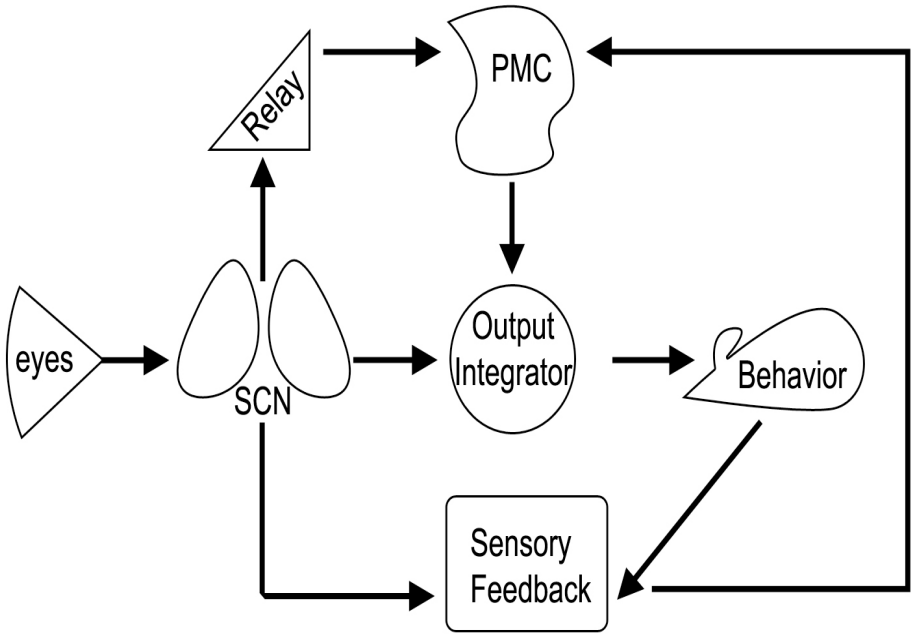


Figure 1: Box diagram of the possible interactions of the circadian system with the primary motor cortex. The motor cortex sends descending signals to trigger behavioral programs, which are modified by sensory feedback. Circadian timing

generated in the SCN may impinge upstream or downstream of the motor cortex which will be revealed by our experimental results

Brain areas like arousal and sleep centers can directly modulate motor activity in a sensory-dependent and independent manner, as well as modulate the effect of sensory feedback on motor programs (Figure 1). One of these centers is within the hypothalamic suprachiasmatic nucleus (SCN), which contains the master circadian pacemaker. The master clock regulates the 24-hour timing of overt behavior and physiological rhythms, including the sleep-wake cycle and locomotor activity[6]. The circadian system organizes almost all behavioral and physiological processes into the 24-hour domain. Because of the circadian input to motor centers, we hypothesize that the same motor tasks performed during the day will require different patterns of PMC neuronal activity than during the night. In other words, our central hypothesis is that the master circadian pacemaker modulates the motor programs within the PMC in the circadian domain. We compare ECoG activity associated with wheel-running, performed at different times throughout the day. Our hypothesis predicts that the same task should be underlined by different ECoG patterns when performed at different times of day, demonstrating a circadian regularity to PMC neural signals.

Systematic variance originating from a circadian cycle is significant in that it can degrade the performance of any ECoG or EEG-based BCIs used over the period of a day. If the circadian modulation in PMC does exist, then factors like the time of day when a calibration is done would be invalid at another time of day or night. It is therefore crucial as these implants go from exploratory research to 'always-on' and continuously used interfaces in clinical trials that any time-varying components are identified and characterized to compensate for.

Approach and Data Collection Methods

Our goal was to detect changes in PMC electrical activity throughout the circadian cycle of the mouse. The only variable of the experiment that we desired to have change across recordings was the mouse's subjective time of day, and try to keep everything else constant. Especially of concern would be other external variations in the environment that

would introduce time varying phenomena for both during the recording and over the period recordings were made. The concern of other time-varying phenomena would be to misattribute those effects as circadian in nature.

Some variables we were able to control for in the system design and others were found through iterations on the experimental procedure. In the end, we had a process that did well to eliminate sources of noise. In each of the following sections we will outline what was done to collect the data and perform the experiment with special emphasis put on choices that were made to make each recording environment as similar as possible so that analysis could be done between recordings that were taken on different dates and times. Jennifer Gile and Dr. de la Iglesia from the Biology department led the surgeries and recordings. They managed and developed the techniques discussed in this project as well as handling the mice and performing the surgeries and data collection. I served in a role of monitoring the data coming in from the collections to check for quality and trying to identify issues in the signal early during the recording periods, as well as the inevitable debugging to find what went wrong when the recordings were off.

The early recordings from the mice were more experimental, with more varying of some experimental parameters to find what gave us the most consistent high quality data. There is a small window of time in which all the data could reliably be collected, so the modeling and analysis was done after collection. These protocols (Appendix A) generated high quality data sets of running data for each mouse in each recording instance. This data library was then available offline to try many different models to search for dynamics that were periodic with respect to circadian period.

Experimental setup

The experimental setup allowed for each mouse to be exposed to the wheel only when we were to record. Under restricted access to the wheel, animals spontaneously run immediately after they are given access to the wheel. Additionally, with unrestricted access to the wheel animals would only run during their subjective night. We also found that moving the mouse between cages was enough of a distraction that it would disrupt the mouse and add an extra dimension of neural activity and arousal. For these reasons,

the mouse stayed in their home cage for the whole experiment and it remained tethered to the data acquisition system (DAQ) the entire time to avoid plugging and unplugging the head mount attached on its head. When it was time to record the mouse's neural and running activity, the divider blocking access to the wheel was lifted and the mouse was able to run on the wheel. This experimental setup prevented unwanted changes induced by arousal or manipulation-induced stress, which could themselves change over the circadian cycle and introduce variability between recordings. Thus, we feel confident that differences detected between recordings taken at different circadian times are truly associated with circadian modulation of the PMC electrical activity that is associated with wheel-running.

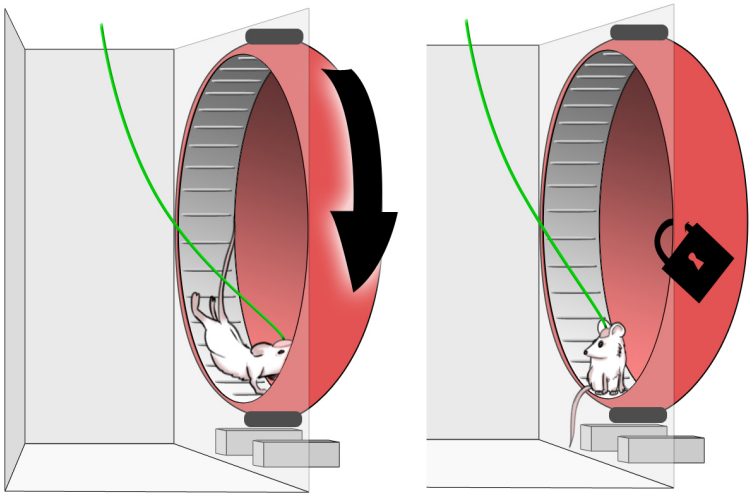


Figure 2: Wheel running enclosure with the mouse connected via its amplifier tether

Process

Prior to surgery, each mouse is allowed to habituate to the cage and learn the wheel-running behavior. We discovered this habituation period would lead mice to readily run on the wheel after surgery. During surgery, dural surface electrodes are implanted on the motor cortex, two EMG electrodes on the neck muscle, and a connector attached to the skull to fasten the tether to. After implantation surgery, the animals are allowed to recover for at least 5 days. This period let the mice physically recover from the process and also allowed scarring around the electrodes to stabilize. The scarring response post-surgery around the electrode alters the electrical dynamics of the tissue, and this period

gives time for the electrode to have consistent signal strength over the period where recordings are taken. Once recovered from the surgery, a pre-amplifier tethered to the DAQ is plugged to the skull connectors and the mouse is allowed another 2 days to adjust to the tethering. The animals are placed into constant darkness, so that the mice endogenous circadian clock oscillates in the absence of light/dark oscillations or other environmental cycles.

The locomotor activity is constantly monitored with infrared-beam detectors to determine the phase of each mouse's circadian locomotor activity rhythm. This is known as "circadian time" (CT), with CT12 being the locomotor activity onset in their home cage. Eight ECoG recordings are across the circadian cycle (CT00, 03, 06, 09, 12, 15, 18 and 21). To avoid fatigue on the mice, 2 recordings per day, spaced 12 hours apart, are made. This helps to ensure that changes across the day are not due to mouse fatigue and losing interest in running. The order of timed recording pairs is randomized for each group of mice, which decouples any effects from the ordering of recordings. We can analyze the ordering effects then separately from CT circadian effects.

Each individual recording consists of lifting the divider to allow access to the running wheel for twenty minutes. Some early trials lasted for ten minutes of running activity and ten minutes of resting (stationary) in which the wheel is locked in order to do a running versus stationary analysis, but later recordings allowed for twenty minutes of free running, in which the animal would spontaneously show non-running periods. Throughout the recording trial the wheel revolutions per minute (RPM), mouse EMG and ECoG activity are recorded.

Brain Surgery – EcoG implantation

Due to the small size of the mouse brain, we cannot state with confidence the exact coordinates in the motor cortex our electrode was placed but attempted to place over the coordinates for limb control. However, as wheel running involves the majority of the mouse body and musculature, placing an electrode in the motor cortex of each hemisphere will record electrical activity associated with motor activity somewhere in the body as the mouse runs on the wheel. Figure 3 shows the layout of the recording electrodes. The reference, ground, and third channel were modified until we had a

procedure that was reproducible for multiple mice, reduced noise in the measurement, and was consistent over the recording period. Multiple iterations and refinements to the surgical process were made to achieve a consistent neural model for each mouse. Early surgery was with silver-wire hooks through the skull, with the end at the desired location. Later the procedure was changed to dental screws, the tip of which makes electrical contact with the dura matter. Two electrodes were implanted bilaterally in the motor cortex (one per hemisphere). One reference electrode was placed in the cerebellum, a ground was placed in associative cortex, and two silver-wire EMG electrodes were placed on the neck muscles. Early surgeries did not record EMG on this third channel, instead recording elsewhere in the brain; this is discussed in more detail in the Results section.

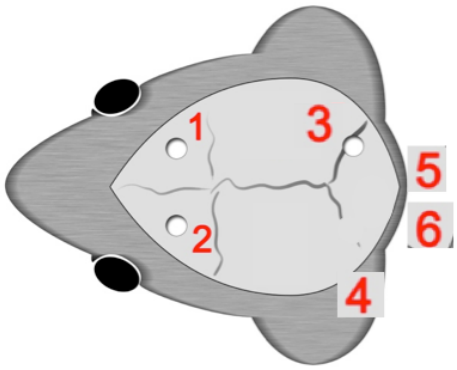


Figure 3: Surgery locations for the 6 electrodes. 1 and 2 are implanted in cortex and used to measure the PMC. 3 is the reference electrode and 4 is also in cortex and acts as the electrical ground. Electrodes 5 and 6 are attached over a neck muscle to measure EMG when the mouse runs.

Data Acquisition

Adult mice were housed in individual cages and their locomotor activity was monitored in 10-minute bins using Clocklab software (Actimetrics) analyzed using the El Temps software[7]. The locomotor activity was used to determine the true circadian period of the mouse. This is an established mechanism to determine and mark the CT intervals by observation. The same Clocklab software package was used for recording from a tachometer to measure the RPM at a one second interval. Initially we thought to

characterize the speed of running with the ECoG motor signals, but found no significant difference in our models and simplified it to three states: actively running, stationary, or neither.

The ECoG recordings were done using a Pinnacle 3-channel system run at a sampling frequency of 400 Hz, which gives proper frequency resolution up through the upper gamma-band of 100-200 Hz that is thought to contain important details in the detection of motor behavior and movement characterization[6]. Two channels were dedicated to measuring over the PMC, with the locations outlined in the Surgery section. The Pinnacle model allowed for flexibility in the third channel and could be adapted to either measure the higher voltage EMG signal or as an additional ECoG channel[8]. The final design recorded EMG from the back muscles in order to improve the time resolution for the detection of running.

Matlab Code for Data Structure

In order to manage reading and processing all of the recordings, a Matlab code library of scripts and data structures was created for the project (code attached in Appendix CODE). This consisted namely of a MouseData object that stores all the information for each mouse. It has all of the wheel and neural data for each recording, and orders these recordings by CT. It allows for each mouse to have a variable size of recordings (mostly either 8 or 16 depending on number of cycles). In having a specified MouseData Matlab object, signal processing methods were written to iterate through the recordings to compute models and/or plot the data for each mouse. We only wanted to make data comparisons contained for each mouse and look for patterns and commonality after modeling each mouse individually as every surgery and electrode placement, as well as any neural or musculature differences between the mice, would vary.

From the two measurement systems, we had two unsynchronized data streams associated with each recording. There was a wheel RPM data set that came from the ClockLab tachometer and an ECoG data set that came from the Pinnacle DAQ system. The MouseData object handled reading in and merging the two data streams so that, based on a timestamp, both referenced the same instance in time. This also had to take into account

the two different sampling frequencies of the two systems, as wheel rotations were sampled at 1 Hz, and the 3 channels of neural data were sampled at 400 Hz. The ECoG signal was not altered, but the wheel running signal was modified using a Zero Order Hold (ZOH) to synchronize the two streams. Once the data was in a defined data structure form, algorithms and models could be easily written to iterate through each of the recordings to analyze and plot model changes over a circadian cycle.

Methods Overview – Rejection Criteria

Our experimental design emphasizes the animals' own endogenous rhythm as the primary variant across recordings. Because of the invasiveness of the procedure and the fact that recordings were done over a period of weeks, we only included signals from mice that were healthy and consistently run throughout the experiment, and that showed similar power levels from the first to last recordings. Further analysis was done in post-processing to confirm that the chronological ordering of when the recordings were taken did not alter the outcome of our analysis. Given our experimental design and strict rejection criteria, we had high confidence in the remaining datasets that any cyclic patterns in the PMC running signal could be attributed to an endogenous Circadian rhythm.

Results

The Pinnacle data acquisition system filters the incoming data from the six implanted electrodes. This is done in hardware upstream from the actual data acquisition unit to reduce how many channels they needed to do the recording as well as cancelling out common background noise[8]. The system has a single ground (electrode 4, fig.1) that is used by all the other electrodes to measure the relative voltage difference. It uses electrode 3 as a reference to subtract from both electrodes 1 and 2, each subtraction forming Channels 1 and 2 in the system with the intention of removing noise present on all ECoG channels so that the signal from Channels 1 and 2 are localized to their resident neural population. Channel 3 is formed from the subtraction of electrode 6 subtracts from electrode, and this channel has two pre-amp settings that can be set to allow for measuring either ECoG or the order-of-magnitude higher EMG.

Using a reference electrode to make a subtraction eliminates a significant level of common noise, for example 60 Hz from any room with electricity that would be picked up. Additionally, EMG noise from the neck would interfere with the signal in channels 1 and 2 if the reference in electrode 4 did not subtract out the noise common to each electrode. However, this hardware-level filter severely limited our options for what was possible for using any other post-processing noise cancellation techniques because the raw signal from each channel was lost in the subtraction, only the difference between each channel and the reference was recorded. We ran several tests to see what the effects of changing the ground and reference electrodes were and analyzed the signal quality and settled on electrode placements that gave us the cleanest signal. A short 18-second trace is shown below in Figure 2 for the 3 channels from Pinnacle along with the wheel revolution count.

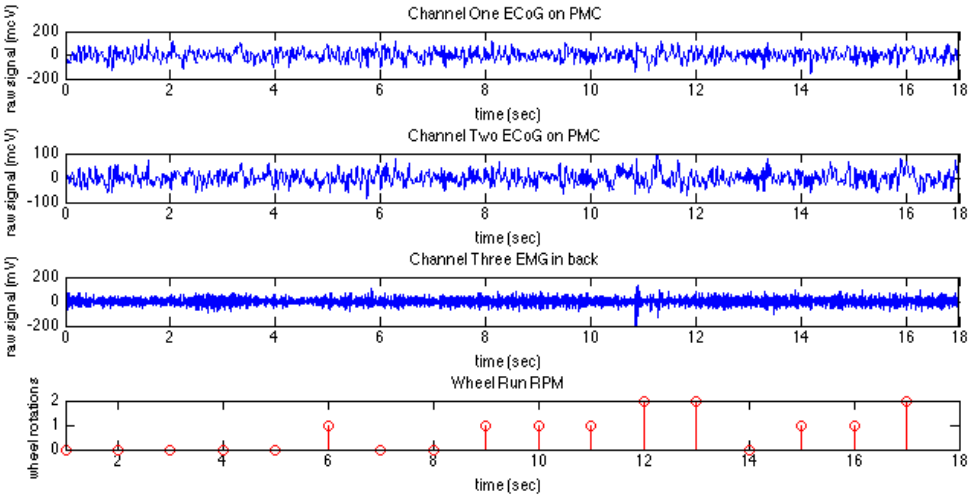


Figure 4: Raw data after synchronizing Pinnacle and Clocklab data streams, despite the differing sampling rates using ZOH to space out the wheel running signal.

Each mouse was given a unique label following the schema of mXX, where XX was the number of mouse in the trial. This naming applied to both Pinnacle and Clocklab files and labeled in the Matlab analysis for each mouse as it is very important that data only be compared within the context of each single mouse.

Data Analysis

Analysis methods and overview

Previous studies on PMC across other animal models have shown that there are changes in the gamma band associated with movement[6]. Because of the lack of specificity with our electrode placement, both in size of the electrode and in size of mouse brain, cannot claim to be recording over a specific motor mapping other than it being within the PMC. Our focus was to find variance that can be explained by circadian time, which affects the whole motor region task; the presence of this variance would impact all researchers working on discerning motor cortex programs over the course of a day or longer.

Our experimental design demands the isolation of PMC signals that specifically correspond to wheel-running events. To confirm that our experimental setup was capable of detecting known differences between running and stationary events, we first focused on the single recording to find intra-recording difference in PMC signal qualities. Not only did this confirm that we could reliably detect behavioral differences, running versus stationary, but we could use that detection to build a running filter to identify running events across all recordings.

Lastly, by analyzing the spectral power change of ECoG signals after the surgery, shown in depth in Appendix C, we confirmed that our surgery and recording process had negligible effects on electrode degradation and scar tissue build-up.

Averaged spectral power over whole recording

We chose to analyze the frequency components of the signal using the spectral power in the Fourier domain and used the Welch transform to calculate it. Our rationale was: much of the existing literature uses this computation for as a BCI control interface and so effects we find with this method will be directly visible for those algorithms, and the loss in frequency resolution does not matter significantly because we are looking at the total power in different frequency bins and not trying to accurately identify the peak frequency.

A *note* on algorithms used to compute spectral power in neural signals. Some previous research has used the Hamming windowing when computing spectral power using the Welch method. This is problematic because the signal strength in the higher frequency bands drops below the side-lobe artifact caused by the Hamming window. While one

could then split this into two calculations, first using a low pass filter on the data and computing the spectral power and then again using a high pass filter and computing the spectral power for that bin, one could still use the Hamming window.

We decided that we preferred a single spectral power curve and not deal with the extra steps of additional filtering, which in our case is unnecessary since we are only looking at frequency regions and not individual frequencies.

Windowing Function	Peak Approximation Error, 20 log10 Sigma	Peak Side-Lobe Amplitude (relative)
Bartlett	-25	-13
Hamming	-44	-41
Blackman-Harris	-74	-57

Table 1: Windowing Functions side lobe effects on spectral power calculations[9] is a trade-off in frequency resolution and the side-lobe amplitude due to the peak frequency.

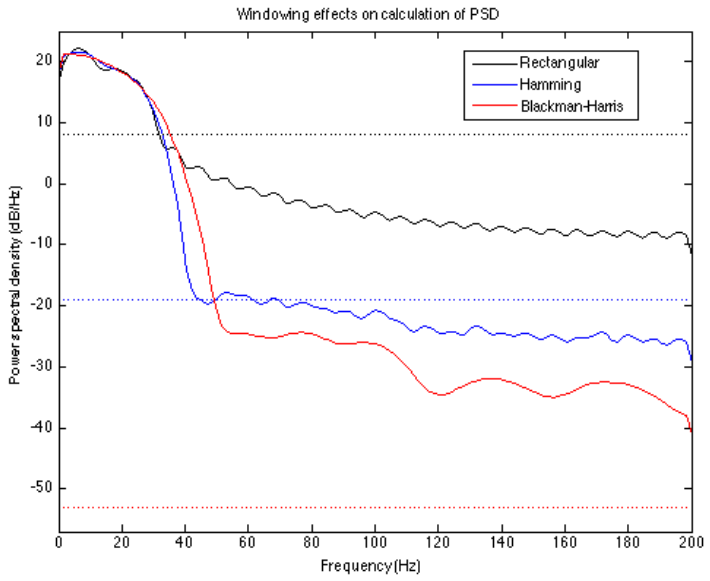


Figure 5: Windowing effects of different methods used in calculating Spectral Power

Figure 5 above demonstrates what the Peak Side-Lobe amplitude will do to the signal in the higher frequencies where the signal strength is weaker. The dotted line shows the side

lobe cutoff for each method given its peak amplitude. The artifact from the algorithm carries the higher magnitude power in the low frequency components and wipes out any features in the high frequency spectrum, making any analysis unusable.

Initially, we looked at taking the general spectral power over all time. Differences in sleep will show up here and even though the mouse was awake and active, we first needed to confirm whether the circadian effects manifested for all behaviors as a baseline modulation over time of day. Analysis was done to test for a general circadian baseline and a look into the different parameters used in calculating the spectral power for the mouse's neural signals, Appendix B. There were no general, behavior-agnostic changes over circadian time while the mouse was awake.

Running Detection

In order to demonstrate that the mouse model could show a circadian difference, we first needed to show measureable differences in its behavior. To detect this we took the measurements from both the tachometer to show RPM of the wheel as well as the EMG output for the mouse's back muscles to find levels of activity.

The earlier surgeries did not use EMG, using instead a 3rd channel in the brain to look for simultaneous circadian cycles in non-PMC areas of the cortex, to see if any circadian phenomena was localized to just motor or the whole brain (mention this up in the animal model/process section too). As we did not see any average behavior change outside the PMC, we moved the 3rd electrode to measure EMG as it was more beneficial to have finer time resolution and accuracy when running movement started. Because of the slow sampling frequency of the wheel, we had too much lag to be able to set the onset time in the ECoG signal for when running began, and so the filter took into account both EMG and wheel revolution data.

We looked at the starting instance of running with the assumption that in the moment prior to the muscle movement beginning, we would capture the motor signal originating from the PMC. While we may have been able to use all the neural signals while the wheel

was rotating, it could be that the motor signal was sent, that wheel running may have been autonomic and the movements could be controlled at the spinal cord level and not the motor cortex. This is an unknown in mice and we did not want to base our dataset on the assumption that PMC would be sending a constant 'run' signal to the mouse's muscles. To play it safe, we only considered events that were at the start of wheel-running from a stand-still and where the EMG transitioned from rest to actively engaged. We chose to set a high bar for our data quality to minimize our assumptions and variability for the motor signals and narrowed what we looked at to this very specific instance. The running pattern for almost all of the mice was not a constant 20 minute long spin as if they were on a treadmill. The mice controlled their running pattern, but were kept inside the wheel enclosure. The mice would typically run for a burst of 20-30 seconds, stop for a short while, run again for another burst, and so on. In order to ensure a high quality data set, we manually sorted through the running data on four recordings for two different mice to create a marked data set for running events and stationary events. Running events were considered events in which the wheel had at least 3 consecutive seconds of rotations as well as the EMG signal having a considerable amplitude increase. The amplitude increase was on the order of 2-3 times the baseline voltage measured from the muscle. Stationary events were events with at least 5 seconds of no wheel motion, as well as at least 3 seconds of no EMG signal amplitude going above baseline. Using these criteria, a couple hundred samples for both running and for stationary were recorded to use for training a filter that could automatically detect running for our animal model.

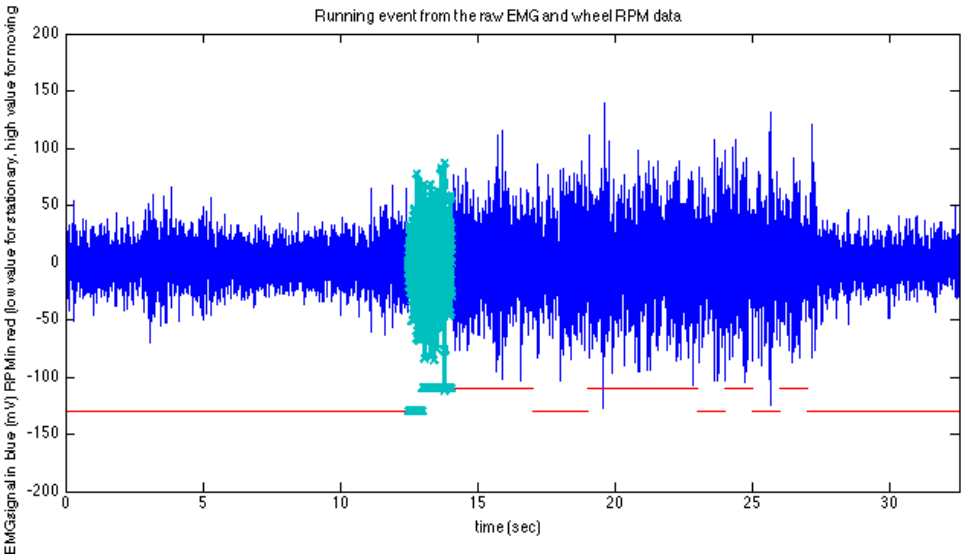


Figure 6: Example of where a running event (in light blue) would be considered. Even though there is other regions where there is a strong EMG signal and wheel RPM, we used the beginning of the movement to classify running events.

The filter used the wheel data and required consecutive rotations to focus on those seconds, and then used the EMG data to find the exact moment to a resolution of $1/200^{th}$ of a second where running began. The EMG filter was made by taking the spectral power of the EMG training data, which showed significant differences in signal amplitude at different frequencies compared to the spectral plot of the stationary event. The spectral power plot was made using the Blackman-Harris windowing function and then squaring the Fast Fourier Transform (FFT) with itself. This analysis highlights the contributions to the overall signal from sinusoidal functions from 1-200Hz as our sampling rate was 400Hz.

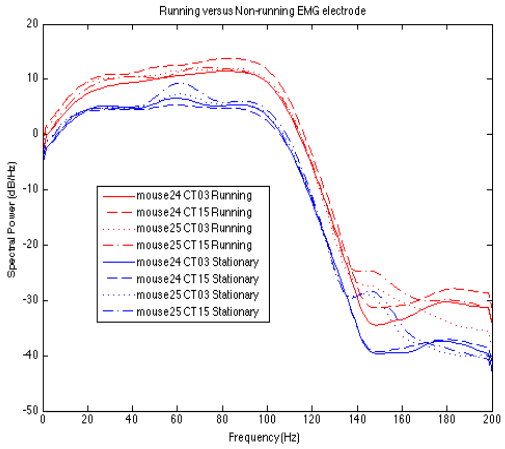


Figure 7: Spectral power of the EMG channel shows definitive detection of running versus non-running events, and is used in the filter for determining running events.

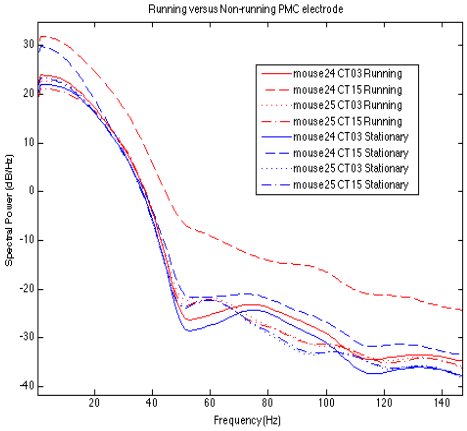


Figure 8: Spectral power analysis on running versus non-running in the ECoG channels shows change in power when compared at the same recording period.

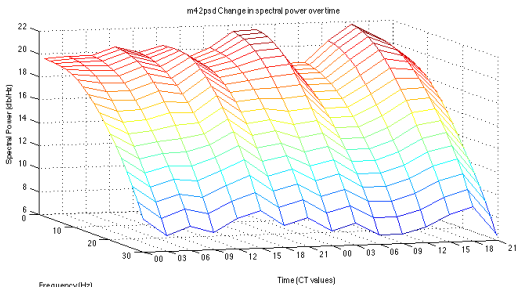
The filter was built so as to favor a very low false positive rate (FPR) while allowing for a reduced true positive rate (TPR). The rational being that in the recordings there was an excess of events needed for analysis, so the focus was to catch only running events and missing a few while not accidentally including errant stationary events. The filter chosen in the Circadian modeling analysis had a FPR of 0% and TPR of 66% on the training data set. We wanted to best eliminated false positives and make up for the lower true positive rate by taking longer recording sessions as that was under our control and would give us the quality data set to draw our later inferences on.

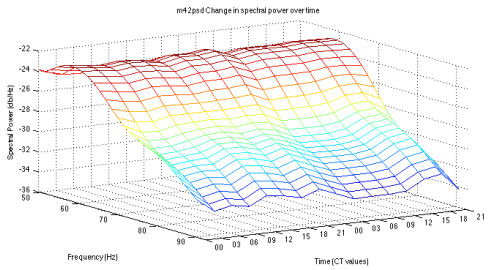
Time of Day – 48 hour cycles

After using the Running Filter on the data to focus on the single event, changes over the circadian cycle became apparent. As there is a large range of behavior during the recording, the prior, averaged spectral power over the whole recording was not able to isolate a single activity as well and show how it was altered by the circadian cycle. On the first batches of mice (m01 – m34) only one full, 24-hour cycle was recorded. This meant 8 recordings separated by 3 hours on the mouse's circadian, subjective time performed over about a week's time. From this data, included in Appendix C, we saw oscillatory changes over a single circadian cycle, but the oscillatory pattern differed for each mouse. To confirm that each mouse's oscillatory pattern could be replicated a second circadian period was recoded for some mice. Two different models for the ECoG data are shown, both the spectral power taken over the cycles and the autoregression analysis used to find the poles of the system. Both techniques demonstrated repeated, periodic behavior that occurred over the circadian period.

3D Spectral power over time

To compare the spectral power of the ECoG signal over time, we computed the power spectral density (PSD) using the Welch method with a Blackman-Harris window so that we could look at the whole 1 – 200 Hz frequency components. A PSD was computed for each running instance in each recording set. Each of these PSDs were then averaged together per each recording, a recording here being a single CT time point. This led to 16 averaged spectral power plots, as we had two recordings for each CT point in these 48-hour cycle recordings. Finally, each of these averaged spectral power plots were plotted as a 3-dimensional mesh plot in Matlab to show the changes across frequencies over the subjective circadian cycles of the mice.





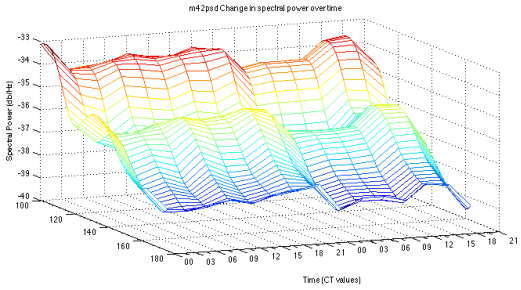


Figure 9: m42 spectral power changes over a 48-hour cycle, ordered by subjective time of day. The three plots show the low, middle, and high frequency bands separately to minimize the dynamic range of the power so that differences are evident in each frequency band.

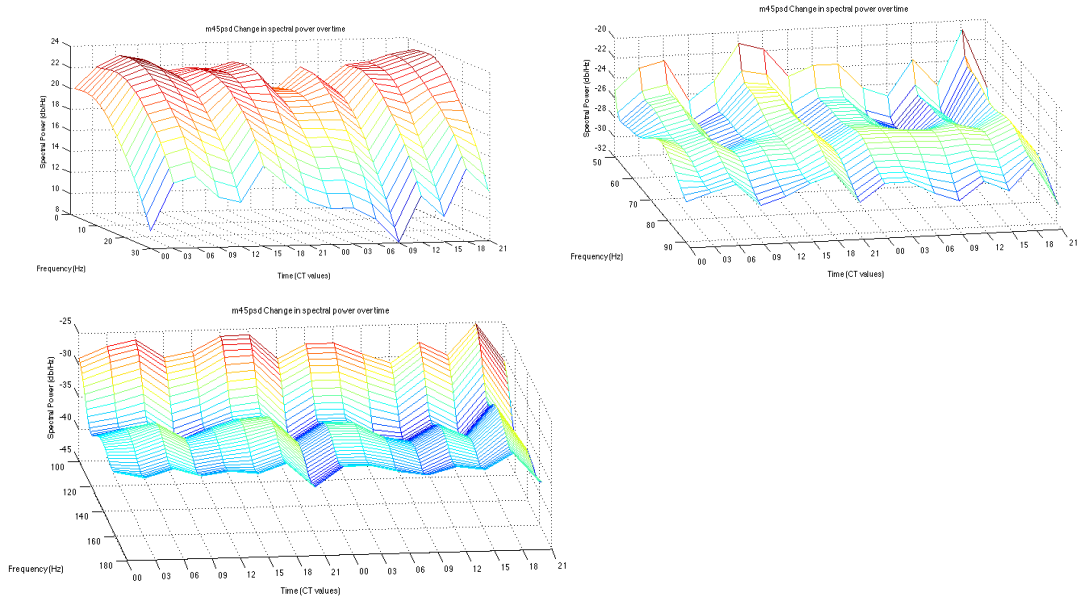


Figure 10: m45 spectral power changes over a 48-hour cycle, ordered by subjective time of day

More trials are planned to obtain a larger sample size, to obtain statistical significance as well as to find if there are clusters to the circadian pattern for how the mouse's PMC running signal changes over its subjective time of day.

As the 3D plot of the entire spectral power helps to visualize the change, binning the power in a certain frequency band shows with more certainty the effects of the circadian cycle on spectral power. Using the same mice and calculations in Fig. 9 and 10, the total power in gamma band was added up and displayed how it varies over circadian time because this aggregate gamma PSD measure is used in current BCIs. The value in the bin

was taken as a relative value, the sum of the magnitudes for all the frequencies within the bound limits of 50 to 100 Hz.

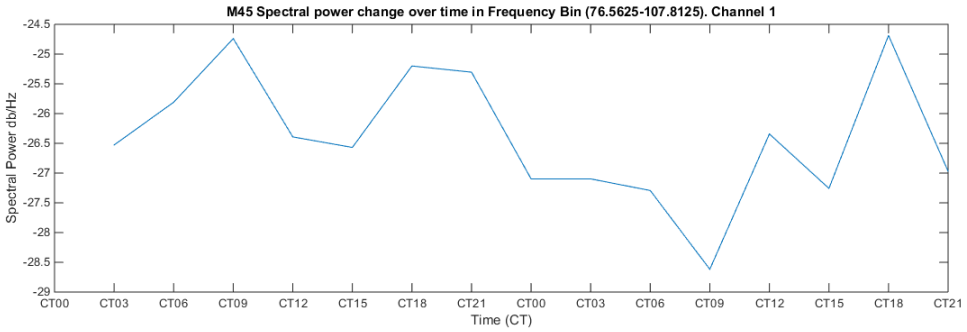


Figure 11: Average power within the low-gamma frequency bin taken from the averaged PSD of all running events for each CT point. Demonstrates the significant change in dB for a frequency bin that would alter the baseline for a threshold in a BCI control signal.

Conclusions

We have identified the existence of oscillatory patterns in the PMC electrical signals associated with running behavior that appear to be influenced by the endogenous circadian system. However, the results have not yet achieved either statistical significance for individual cyclical behavior or a similar cyclic pattern shared across all mice.

While future experiments will confirm the presence of a circadian pattern as a source of noise in PMC, we have built the tools and the experimental groundwork to use the mouse as an experimental model to rigorously test the involvement of the circadian system in the modulation of PMC signals associated with motor tasks. The experimental setup adapted from other sleep studies and adding the ECoG measurement gave us an animal model and process that gives quality data where comparisons across CT time can be assessed.

Unlike other animal models like the monkey or rat, mice have not yet been studied to exemplify changes in the gamma band of ECoG spectral power in PMC during moments of running versus stationary behavior. After confirming our ability to detect these changes with the smaller mouse brain, we were able to exploit this experimental model to assess circadian modulation of the PMC signals.

Each mouse demonstrated a variance over its recordings set that was most pronounced and had visual repetition over a 48-hour cycle. Nevertheless, the circadian phase of the oscillation differed between mice, and it is possible that, under our experimental conditions, this circadian pattern of PMC activity modulation is repeatable within each animal but not between animals.

Future Work

Next Steps – Genotype Study to Support a Circadian-based Variance

As the results have shown promise to a circadian-based variance in the PMC electrical signal, more data is in the process of being collected with more mouse trials to improve statistical significance of the differences found.

Additionally, there are strains of mice that do not exhibit circadian rhythmicity because components of their genetic clockwork are missing a functional circadian clock[10]. These mice are denoted as BMAL1 knock-out (KO) strains and will help to isolate the possible mechanisms that are the root of the electrical signal differences seen in our results. The lack of an underlying circadian rhythm in an otherwise healthy subject will evidence any oscillations in the wheel running signature across times of day in the genetic knockouts while PMC electrical signal changes should be intact in the wild type (WT) controls. This would confirm our hypothesis that the canonical circadian molecular clockwork modulates the motor programs within the PMC.

Figure 12 shows early results of the BMAL1 KO mouse compared to a WT mouse below demonstrating the substantial change in PSD for running events in the mouse with a functioning circadian clock versus the fairly unchanging levels seen on the mouse without a functioning circadian cycle. While more trials with the BMAL1 knockouts are needed as well, these initial results help to support the hypothesis that the circadian rhythm is the source of variance.

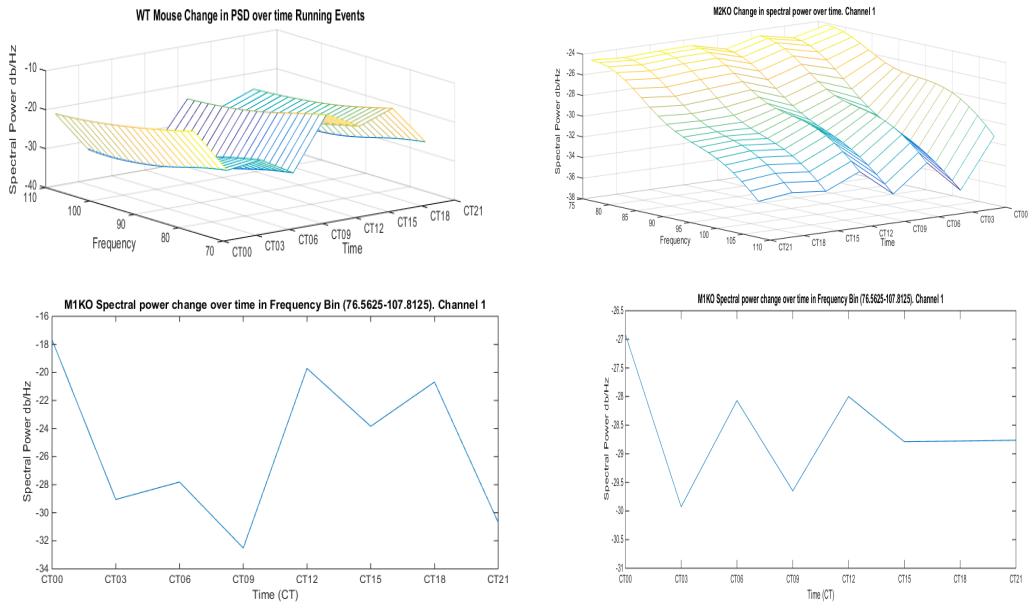


Figure 12: PSD changes for run events in a WT (left) versus a BMAL1 knockout (right). Notice that as time-of-day changes the power levels for the same frequency change quite drastically for the WT mouse while the BMAL knockout remains at about the same power for each frequency. The above two plots shows each frequency's average PSD, the plot below shows the average power level for the entire frequency bin from the above plot to simplify showing the drastic variance changes seen in the y-axis range in the WT and BMAL1 mice.

Applications – Long-term implanted BCIs

Characterizing the variance due to an animal's endogenous circadian rhythm needs to be accounted for in next generation, chronic brain computer interfaces that will be coming to market in the near term. While prior BCIs have been calibrated and then used immediately in the lab setting, chronic BCIs will be worn on the person and will utilized throughout the day to perform critical tasks. One such chronic BCI would be a Deep Brain Stimulator (DBS) that uses an implanted ECoG array to make cortical readings to guide when it should stimulate to control essential tremor in Parkinson's patients. DBS is currently used to reduce essential tremor but the current input does not take into account the measure of tremor or the person's state or intended action[11]. A chronic BCI prototype was demonstrated as a way to create a closed-loop control system that would measure motor intent from the person's PMC to then only run the neurostimulation when movement was desired[12]. This system used healthy subjects and EEG in place of the

implanted ECoG with Parkinson's patients, but used the same sensed movement intentions via EEG to determine when stimulation was required to mock when the stimulation would occur in the proposed closed-loop DBS system. The closed-loop DBS prototype to calculate motor intent were using a similar frequency-based power threshold, as we have demonstrated there to likely be a time-of-day variance in mice, then compensating for the variance could lead to performance improvements over long term use of the BCI.

References

1. Instant Neural Control of a Movement Signal. Serruya Hatsopoulos NG, Paminski L, Fellows MR & Donoghue JP (2002). *Nature* 416, 141-142.
2. Direct Cortical Control of 3D Neuroprosthetic Devices. Taylor DA, Helms Tillery SI & Schwartz AB (2002). *Science* 296, 1829–1832.
3. Learning to control a brain–machine interface for reaching and grasping by primates. Carmena JM, Lebedev M, Crist RE, O'Doherty JE, Santucci DM, Dimitrov DF, Patil PG, Henriquez CS & Nicolelis MAL (2003). *Plos Biol* 1, 1–16.
4. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. Hochberg LR, Serruya MD, Friehs GM, Mukand JA, Saleh M, Caplan AH, Branner A, Chen D, Penn RD & Donoghue JP (2006). *Nature* **442**, 164–171.
5. The Role of High-Frequency Features in Movement Classification. Miller, K. J. et. al, Beyond the Gamma Band: IEEE Transactions in Biomedical Engineering, 2008
6. Transplanted Suprachiasmatic Nucleus Determines Circadian Rhythm. Ralph, Foster, Davis, Menaker. Science vol. 247, 1990. p 975-978
7. http://el-temps.com/principal.html, Diez-Noguera, University of Barcelona, Barcelona, Spain
8. Pinnacle 8200 series – Tethered EcoG for mice. http://www.pinnaclet.com/3-channel/tethered-for-mice.html
9. *Discrete Time Signal Processing, Third Edition*. Oppenheim, Schafer. Prentice Hall Signal Processing, p.538.
10. *Circadian Modulation of Neuromotor Control*. Jennifer Gile. University of Washington Honors Thesis. 2014.
11. Adam O Hebb, Jun Jason Zhang, Mohammad H Mahoor, Christos Tsiokos, Charles Matlack, Howard Jay Chizeck, and Nader Pouratian. Creating the feedback loop: Closed-loop neurostimulation. *Neuro- surgery Clinics of North America*, 25(1):187–204, 2014.
12. *Closed-Loop DBS with Movement Intention.* Herron, Denison, Chizeck. 7th Annual International IEEE EMBS Conference on Neural Engineering. Montpellier, FR. April 2015.

Appendix A: Surgery Protocol for 2 EEG/1 EMG in Mice

This guide provides instructions on performing a 2EEG/1EMG mouse surgery for cortical studies (including sleep and seizure studies) using Pinnacle's standard mouse headmount and screws. ***It is recommended that all surgeries be performed at least 5 days before connecting animal to Pinnacle hardware.***

Materials:

- Stereotaxic
- Scalpel/Scissors
- Tweezers
- Curved forceps
- Dental cement and solvent
- Cotton Swabs/Q-tips
- Surgical thread and needle
- Prepared mouse headmount
- Solder gun and solder
- 4 .10" mouse screws
- Screwdriver for EEG screws
- 23-Gauge Needles
- Sharpie
- Puralube (eye ointment)

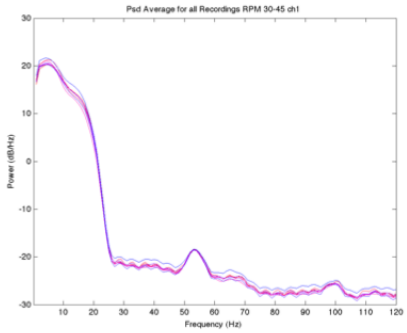
Surgical Procedures:

1. Align the mouse in the stereotaxic apparatus for surgery and prepare scalp for incision by applying betadine and ethanol alternatively three times. At this time, you should apply the puralube to the eyes so that they do not dry out.
2. Using the scalpel or scissors make a 1.5 cm rostral-caudal incision in the skin to expose the skull surface. Use a Q-tip to clear away any excess tissue.
3. Mark the skull with the appropriate coordinates you would like to use with the stereotaxic. Use bregma as the (0,0) coordinate. Mark the coordinates with a sharpie on the skull. *The coordinates for motor cortical recordings are as follows: (+/-1.6,.35) for bilateral motor cortex areas. The ground coordinates are (-1.5, -3.5) and the reference coordinates are (1.5, -6.0). ***Important! Your coordinates will vary depending on what area of the brain you want to record from!***
4. Tapping screw holes should be accomplished using a fresh 23 gauge needle. Place the needle tip where you wish to make the hole and gently push until the needle tip penetrates the skull surface. Slowly rotate the needle and advance it into the hole until the entire tip diameter has penetrated. This will create a correctly sized pilot hole for the EEG screws.
5. If bleeding occurs, hold a Q-tip over the hole and absorb the excess liquid until the bleeding has stopped.
6. Using a screwdriver, advance the screws ½ way into the holes. There should be some slight resistance as the screw threads grab the skull surface. (This corresponds to about 2 ½ screw turns once the screw has caught in the skull).
7. Once all four screws are in place, they need to be soldered to their individual wires connected to the headmount that you have previously prepared.

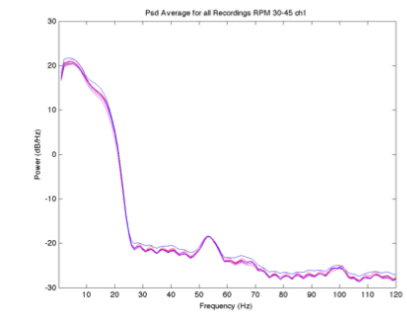
8. To insert the EMG wires, makes a small pocket in the nuchal muscles using a pair of forceps. Insert the EMG wires into the opening and straighten the wires as needed. The EMG wire should be positioned straight in the cavity. A single suture may be needed, depending on the size of the incision and if the wire is not staying in place.
9. The epoxy is important to ensure a solid electrical connection between the screws and headmount. It cures quite quickly and it is recommended to make it only as you need it. Once the epoxy has been made up (it should have a liquid consistency) cover the wires completely with it. You should not be able to see any wires or screws once the epoxy has been applied. When complete, the entire base of the headmount should be coated. You can use a wet brush or a wet spatula to smooth and shape the epoxy better. **Note: make sure to avoid getting epoxy on the sides of the headmount. This area is used during plugging and unplugging the mouse to the preamplifier unit and excess epoxy will make it difficult to maintain a firm hold.**
10. To finish, give the mouse the booster shot of buprenorphine and keep on a warmer until the mouse has woken up from the anesthesia. You should not leave the mouse unattended until it has started cleaning the eye ointment off of itself.

Appendix B: Algorithm parameters effects on analysis

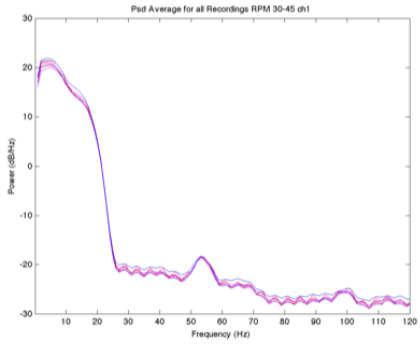
Trailing Average Period



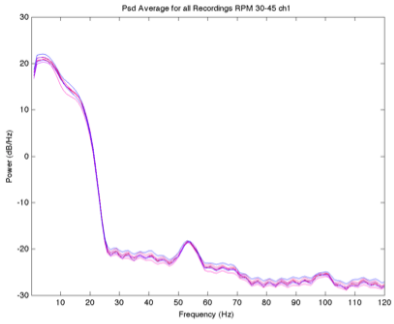
1 Second Trailing Average



2 Second Trailing Average



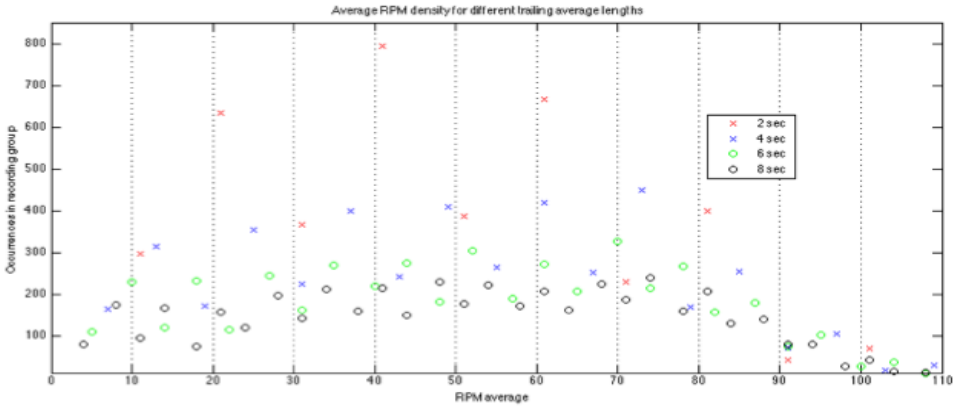
4 Second Trailing Average



10 Second Trailing Average

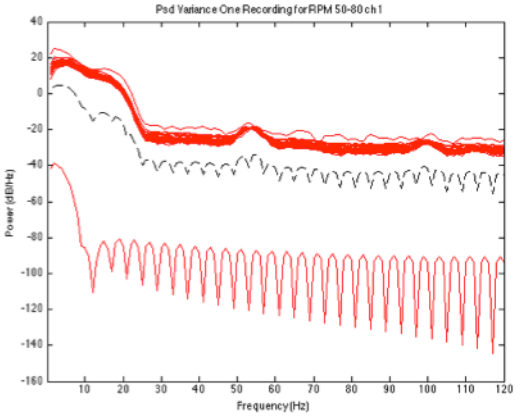
The plots above are all for m03, plotting the same progression of subjective day times (8 plots for ct00 - ct21) only changing the length of the trailing average used in calculating the avg rpm and the PSD. There are many different plots for multiple mice that had good running profiles (m03, m06, and m07) using several different rpm thresholds, all of these are stored in the .Dropbox/Ecog/MousePlots folder. The plots above show the characteristic trend, which are small changes between the averaged Psds, but I don't notice any trends occurring as the window is lengthened. The only real effect observed was a performance drain for the longer trailing average lengths.

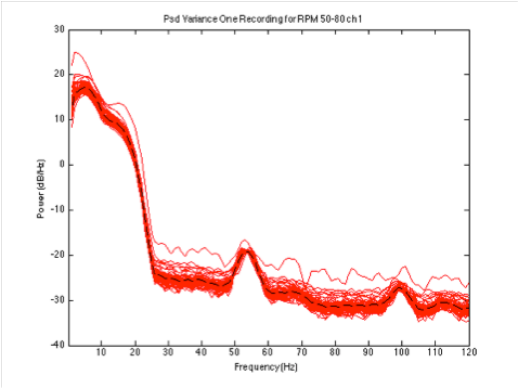
RPM distribution for trailing averaging length



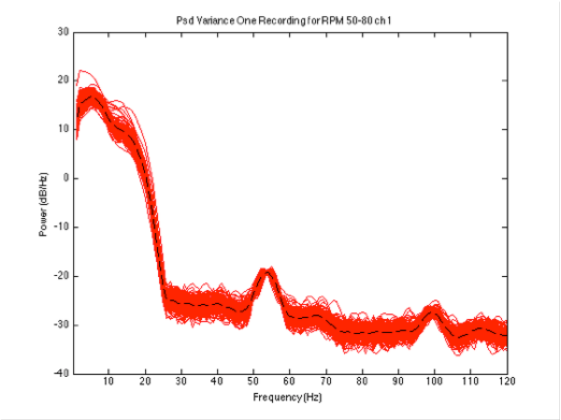
Looking at the RPM distributions, we see that the longer the time period the average rpm density (number of occurrences for a given rpm in the recording) is smoother and more consistent, which allow for smaller rpm thresholds. This plot suggests that we would not want to go below about 4 sec. trailing average lengths as the resolution of the average rpm gets too big.

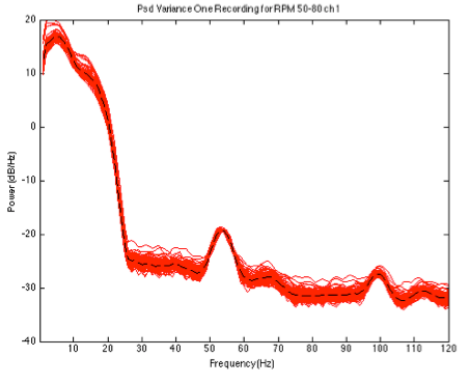
PSD Variance within an average





Left, the Psd average considering all Psd calculations within an Rpm bin can be drastically misleading if there is just one bad measurement (although sometimes there can be several, for 40-50 recordings, typically at least one unreasonably low power recording would occur, the cause is unknown, whether it's a bug in the computation or some physical event (like a wire touching the wheel, etc.). The red plots are each individual Psd plot and the dotted black line is the average of them all.





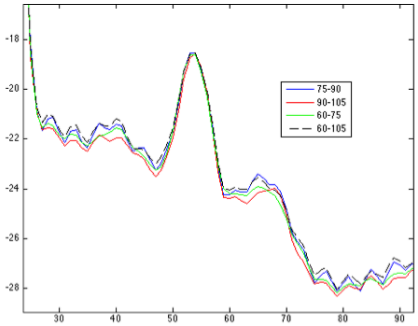
Above, left the Psd variance with a trailing avg of 2 sec, on right a trailing avg of 8 sec. The variance appears to be smaller for the longer trailing average period. All the recordings seem to sit in tighter to the average than in the shorter averaging of 2 seconds in the other plot. We are likely averaging out to see the larger trend with the longer period, we may or may not want this.

Narrow vs. Wider Rpm bin effect on PSD

Confirmed that smaller rpm bins can show smaller differences, but the issue ran up against was more one of number of samples.

Found that for a good running mouse, like m03, the smallest range was 15 rpm that would still get around 40 samples.

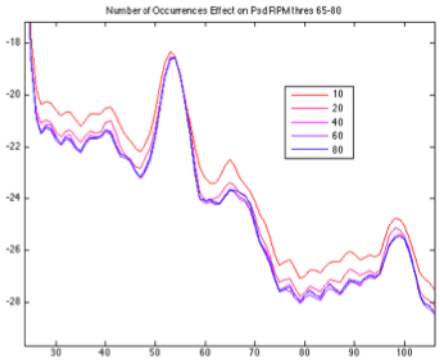
Note, this took more processing to get to work out for this demonstration, the limits were chosen to be able to make the comparison.



Then, choosing an rpm thresholds noted in legend of varying ranges and only plotting the first 40 occurrences.

Number of occurrences effect on Psd average

The sample size N should exceed 40 samples based on analyzing the data for rpm size, different mice and psd variance.



m03 -

For recording 6 with the rpm threshold of 50-80 (pictured right) the average Psd converges pretty closely to around N = 40 and is the case in the other recordings and rpm thresholds as well.

Appendix C: Chronological effects on recordings, order of recording.

To check to see if there were effects due to the time the recordings were taken, as this was one factor we could not account for, we wanted to look at if there were effects due to time progression such as electrode degradation or scar tissue build up that are common in these types of procedures. Care was taken to randomize ordering of the sequence of CT points taken over time, not in straight order where effects over days would be tightly correlated with the subjective time of day changes. In the plots below, we are not looking for periodic behavior so much as rather a moving trend, of a time-varying effect depending on how far into the trial the recording was made. We see no such effect in the plots below, where the same analysis used in the circadian cycle analysis, but when the recordings are ordered by the time they were taken after the surgery.

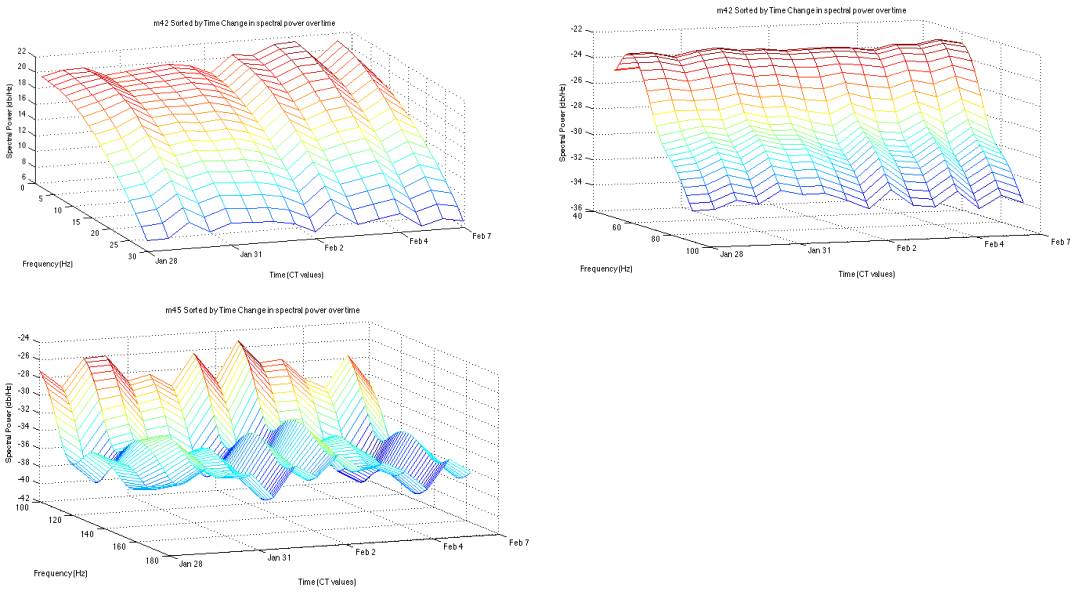


Figure 13: m42 Chronologically ordered spectral power changes. No linear trending behavior is apparent.

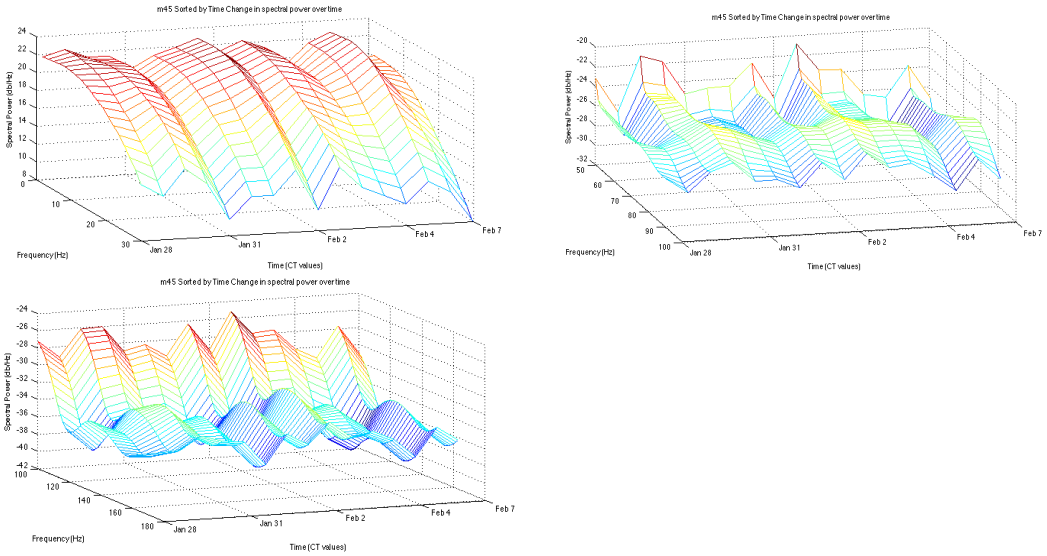


Figure 14: m45 Chronologically ordered spectral power changes. No linear trending behavior is apparent.

Appendix D: Core Matlab Code

Mouse Object:

```
classdef rMouse

    properties
        mouseIdentity
        mouseRecording %array of each recording
        trailingAverageLength
        hasRunningInstancesComputed
        hasStationaryInstancesComputed
        hasPsdOfRunningInstancesComputed
        hasPsdOfStationaryInstancesComputed
        hasDynamicPolesComputed
        areDynamicPolesNormalized
    end

    methods
        function obj = rMouse(excelFilename)

        %needs checks for sizes
            sheetName='Sheet1';
            [numbers, strings] = xlsread(excelFilename, sheetName);
            if ~isempty(numbers)
                newData1.data =  numbers;
            end
            if ~isempty(strings)
                newData1.mouseRecordingFiles =  strings;
            end

            % Create new variables in the base workspace from those
fields.
            vars = fieldnames(newData1);
            for i = 1:length(vars)
                assignin('base', vars{i}, newData1.(vars{i}));
            end
            obj.trailingAverageLength = 1; %We have always been using a
trailingLength of one second

            %READ off the mouse identity, will be specified in same
cell
            obj.mouseIdentity = newData1.mouseRecordingFiles{2, 1};
            %consider changing this to 8 or 16 for different cycles
            numberRecordings = size(newData1.mouseRecordingFiles, 1) -
1; %length of matrix values, first row is Headers, not recordings
            iter = 1:1:numberRecordings;
            tempEcogFiles{numberRecordings} = 0;

            for i = iter
               %Make sure to skip first Row which is just the labeling
               %headers
               tempEcogFiles{i} = newData1.mouseRecordingFiles{i + 1,
2};
               tempRpmFiles{i, 1} = newData1.mouseRecordingFiles{i + 1,
3};
```

```
                tempRpmFiles{i, 2} = newData1.mouseRecordingFiles{i + 1,
4};
            end

            obj.mouseRecording{numberRecordings} = 0;
            for i=iter
                obj.mouseRecording{i} =
rMouseRecording(tempEcogFiles{i}, tempRpmFiles{i,1}, tempRpmFiles{i,2},
obj.trailingAverageLength);
            end

            obj.hasRunningInstancesComputed = false;
            obj.hasPsdOfRunningInstancesComputed = false;
            obj.hasPsdOfStationaryInstancesComputed = false;
            obj.hasStationaryInstancesComputed = false;
            obj.hasDynamicPolesComputed = false;
            obj.areDynamicPolesNormalized = false;
            %all the recordings must be read in first
            rpmAvg = obj.calculateAvgRpmAllFiles();
            clear mouseRecordingFiles;
        end

        %% Notes on this method, of what it is searching for, criterion
to
        %  be considered a running signal for all the recordings
        function [mouse] = computeRunningInstances(mouse)

            iter = 1:1:length(mouse.mouseRecording); %change this to be
the size of the signal...
            for i = iter
                %Check if it has a recording...
                mRecording = mouse.mouseRecording{i};
                if(~isempty(mRecording))
                    %Write out the Recording it is on,,,,
writeln(Computing
                    fprintf('Computing Running instances for recording:
%i of %i\n', i, length(iter));
                    sigLength = 1200; %data points used for the AR
model, keeping consistent across
                    mouse.mouseRecording{i} =
FindRunningInstances(mRecording, sigLength);
                else
                    fprintf(['Error: Recording from index ' num2str(i)
' is empty, running instance could not be computed']);
                end
            end
            mouse.hasRunningInstancesComputed = true;
        end

        function [mouse] = computeStationaryInstances(mouse)

            iter = 1:1:length(mouse.mouseRecording); %change this to be
the size of the signal...
            for i = iter
                %Check if it has a recording...
```

```
                mRecording = mouse.mouseRecording{i};
                if(~isempty(mRecording))
                    %Write out the Recording it is on,,,,
writeln(Computing
                    fprintf('Computing Stationary instances for
recording: %i of %i\n', i, length(iter));
                    sigLength = 1200; %data points used for the AR
model, keeping consistent across
                    mouse.mouseRecording{i} =
FindStationaryInstances(mRecording, sigLength);
                else
                     fprintf(['Error: Recording from index ' num2str(i)
' is empty, stationary instance could not be computed']);
                end
            end
            mouse.hasStationaryInstancesComputed = true;
        end

        function [ mouse ] = computePsdOfRunningInstances( mouse )
           if (~mouse.hasRunningInstancesComputed)
               mouse = computeRunningInstances(mouse);
           end
           %Then iterate and compute the Avg Psd
           iter = 1:1:length(mouse.mouseRecording);
           for i = iter
               fprintf('Computing PSD of running data for recording: %i
of %i\n', i, length(iter));
               %make into a separate var...
               mouse.mouseRecording{i} =
mouse.mouseRecording{i}.computePsdOfRunningInstances();
           end
           mouse.hasPsdOfRunningInstancesComputed = true;
        end

        function [ mouse ] = computePsdOfStationaryInstances( mouse )
           if (~mouse.hasStationaryInstancesComputed)
               mouse = computeStationaryInstances(mouse);
           end
           %Then iterate and compute the Avg Psd
           iter = 1:1:length(mouse.mouseRecording);
           for i = iter
               fprintf('Computing PSD of stationary data for recording:
%i of %i\n', i, length(iter));
               %make into a separate var...
               mouse.mouseRecording{i} =
mouse.mouseRecording{i}.computePsdOfStationaryInstances();
           end
           mouse.hasPsdOfStationaryInstancesComputed = true;
        end

        function [ mouse ] = computeEEGdifferential( mouse )
            if (~mouse.hasRunningInstancesComputed)
               mouse = computeRunningInstances(mouse);
           end
           iter = 1:1:length(mouse.mouseRecording);
```

```
            for i = iter
                fprintf('Computing Ecog Channel differences for
Recording: %i of %i\n', i, length(iter));
                mouse.mouseRecording{i} =
mouse.mouseRecording{i}.computeChannelDifferenceRunning();
            end
        end

        %% Computes the poles from the dynamic system modeled from the
Burg
        %  Autoregression method.
        function [ mouse ] = computePolesDynamicSystem( mouse,
modelOrder, shouldNormalize )
            mouse.areDynamicPolesNormalized = shouldNormalize;
            if (~mouse.hasRunningInstancesComputed)
               mouse = computeRunningInstances(mouse);
            end
            numRecordings = length(mouse.mouseRecording);
            %then take overall max & normalize the other recordings
            %Get Max values from all returned AR Roots.
            maxARCoefficients(numRecordings, modelOrder) = 0;
            iter = 1:1:numRecordings;
            for i = iter
                fprintf('Computing PSD of running data for recording:
%i of %i\n', i, length(iter));
                [mouse.mouseRecording{i}, maxARCoefficients(i)] =
mouse.mouseRecording{i}.computePolesFromAutoRegressionModel(modelOrder)
;
            end

            globalMax(modelOrder) = 0;
            orderIter = 1:1:modelOrder;
            if (shouldNormalize)
                for j = iter
                    for oI = orderIter
                       %if, more than max, store the value.
                       if (maxARCoefficients(j, oI) > globalMax(oI))
                           globalMax(oI) = maxARCoefficients(j, oI);
                       end
                    end
                end
                mouseRecording.polesDynamicModel

                %Now compute this.!
                %iterate again and normalize value and store to the
rec.

            end

            %Run through the calculations.
            mouse.hasDynamicPolesComputed = true;
        end

        %% Freq low index and the high index are the frequency limits
to plot
```

```
        %  the PSD between (what the actual frequencies would be are in
the
        %  psdFrequency file in the Recording, the limits must be
between 1
        %  and 129 (the array size the PSD is being calculated with.
        %  EEGchannel can be either 1 or 2 depending on which channel
to
        %  plot, if neither of those values it will default to 1.
        function [ mouse ] = plotRunningAveragePsdAsMeshOverTime(
mouse, freqLowIndex, freqHighIndex, eegChannel)
            %Checks, check that PSD has been calculated & that index
values
            %are in the correct range.
            if (eegChannel ~= 1 && eegChannel ~= 2)
                fprintf('Invalid Channel for ECog, must be either 1 or
2, defaulting to 1');
                eegChannel = 1;
            end
            if (~mouse.hasPsdOfRunningInstancesComputed)
                fprintf('Computing PSD for all Running Instances');
                mouse = mouse.computePsdOfRunningInstances();
            end
            if (freqLowIndex < 1)
                fprintf('Lower bound for frequency index cannot be
below 1, defaulting to 1Hz');
                freqLowIndex = 1;
            end
            if (freqHighIndex > 129)
                fprintf('Upper bound for frequency threshold cannot be
above 129 (index 129 is fs/2 Hz), defaulting to 1Hz');
                freqHighIndex = 129;
            end

            iter = 1:1:length(mouse.mouseRecording);
            psdData(freqHighIndex - freqLowIndex + 1, length(iter)) =
0;
            %just taking the frequency values from the first recording.
            %This will have had to be computed from theu above,
            %computePsdOfRunningInstances check, and the frequencies
HAVE
            %to be the same for the Psd is calculated the same for each
            %recording
            w =
mouse.mouseRecording{1}.psdFrequencyValues(freqLowIndex:freqHighIndex);

            for i=iter
                %Loop through the signal, form array in calling pwelch
                data =
mouse.mouseRecording{i}.psdRunningInstancesData{eegChannel};
                psdData(:,i) =  data(freqLowIndex:freqHighIndex);
            end

            s = size(psdData);
            py = w;
            px = 1:1:s(2);
```

```
            mesh(px, py, psdData);
            %TODO: Figure out how to get the ticks labeled to show ct
name, for
            %now, will have to stick with doing it manually on final
plots
            xlabel('Time');
            ylabel('Frequency');
            zlabel('Spectral Power db/Hz');
            title([mouse.mouseIdentity ' Change in spectral power over
time. Channel ' num2str(eegChannel)]);
        end

        function [ mouse ] =
plotRunningAveragePsdFrequencyBinAverageOverTime( mouse, freqLowIndex,
freqHighIndex, eegChannel)
            %Checks, check that PSD has been calculated & that index
values
            %are in the correct range.
            if (eegChannel ~= 1 && eegChannel ~= 2)
                fprintf('Invalid Channel for ECog, must be either 1 or
2, defaulting to 1');
                eegChannel = 1;
            end
            if (~mouse.hasPsdOfRunningInstancesComputed)
                fprintf('Computing PSD for all Running Instances');
                mouse = mouse.computePsdOfRunningInstances();
            end
            if (freqLowIndex < 1)
                fprintf('Lower bound for frequency index cannot be
below 1, defaulting to 1Hz');
                freqLowIndex = 1;
            end
            if (freqHighIndex > 129)
                fprintf('Upper bound for frequency threshold cannot be
above 129 (index 129 is fs/2 Hz), defaulting to 1Hz');
                freqHighIndex = 129;
            end

            iter = 1:1:length(mouse.mouseRecording);
            psdData(freqHighIndex - freqLowIndex + 1, length(iter)) =
0;
            %just taking the frequency values from the first recording.
            %This will have had to be computed from theu above,
            %computePsdOfRunningInstances check, and the frequencies
HAVE
            %to be the same for the Psd is calculated the same for each
            %recording
            w =
mouse.mouseRecording{1}.psdFrequencyValues(freqLowIndex:freqHighIndex);
            pAveragePsdInBin(length(mouse.mouseRecording)) = 0;

            tempAvgAdder = 0;
            freqBinIter = 1:1:(freqHighIndex - freqLowIndex);
            for i=iter
                tempAvgAdder = 0;
```

```
                %Loop through the signal, form array in calling pwelch
                data =
mouse.mouseRecording{i}.psdRunningInstancesData{eegChannel};
                psdData(:,i) =  data(freqLowIndex:freqHighIndex);
                for fI = freqBinIter
                   tempAvgAdder = tempAvgAdder + psdData(fI,i);
                end
                pAveragePsdInBin(i) = tempAvgAdder /
length(freqBinIter);
            end

            s = size(psdData);
            px = 1:1:s(2);
            plot(px, pAveragePsdInBin);

            %TODO: Figure out how to get the ticks labeled to show ct
name, for
            %now, will have to stick with doing it manually on final
plots
            xlabel('Time (CT)');
            ylabel('Spectral Power db/Hz');
            title([mouse.mouseIdentity ' Spectral power change over
time in Frequency Bin (' num2str(w(1)) '-' num2str(w(length(w))) ').
Channel ' num2str(eegChannel)]);
        end

        function [ mouse ] =
plotRunningAveragePsdSingleRecording(mouse, freqLowIndex,
freqHighIndex, eegChannel, recordingIndex)
            if (eegChannel ~= 1 && eegChannel ~= 2)
                fprintf('Invalid Channel for ECog, must be either 1 or
2, defaulting to 1');
                eegChannel = 1;
            end
            if (recordingIndex < 1 || recordingIndex >
length(mouse.mouseRecording))
                fprintf('Invalid recording index passed for mouse
recording, defaulting to 1.');
                recordingIndex = 1;
            end
            if (~mouse.hasPsdOfRunningInstancesComputed)
                fprintf('Computing PSD for all Running Instances');
                mouse = mouse.computePsdOfRunningInstances();
            end
            if (freqLowIndex < 1)
                freqLowIndex = 1;
                fprintf('Lower bound for frequency index cannot be
below 1, defaulting to 1Hz');
            end
            if (freqHighIndex > 129)
                freqHighIndex = 129;
                fprintf('Upper bound for frequency threshold cannot be
above 129 (index 129 is fs/2 Hz), defaulting to 1Hz');
            end
```

```
            data =
mouse.mouseRecording{recordingIndex}.psdRunningInstancesData{eegChannel
};
            psdData =  data(freqLowIndex:freqHighIndex);
            plot(psdData);
            xlabel('Frequency');
            ylabel('Spectral Power db/Hz');
            f
            title([mouse.mouseIdentity 'Average Spectral Power,
Recording' num2str(recordingIndex) ', Channel ' num2str(eegChannel)]);
        end

        function [ mouse ] =
plotRunningChannelDifferencePsdMeshOverTime (mouse, freqLowIndex,
freqHighIndex)
            if (~mouse.hasPsdOfRunningInstancesComputed)
                fprintf('Calculating PSD for running instances in all
recordings');
                mouse = mouse.computePsdOfRunningInstances();
            end
            if (freqLowIndex < 1)
                fprintf('Lower bound for frequency index cannot be
below 1, defaulting to 1Hz');
                freqLowIndex = 1;
            end
            if (freqHighIndex > 129)
                fprintf('Upper bound for frequency threshold cannot be
above 129 (index 129 is fs/2 Hz), defaulting to 1Hz');
                freqHighIndex = 129;
            end

            iter = 1:1:length(mouse.mouseRecording);
            psdData(freqHighIndex - freqLowIndex + 1, length(iter)) =
0;
            %just taking the frequency values from the first recording.
            %This will have had to be computed from theu above,
            %computePsdOfRunningInstances check, and the frequencies
HAVE
            %to be the same for the Psd is calculated the same for each
            %recording
            w =
mouse.mouseRecording{1}.psdFrequencyValues(freqLowIndex:freqHighIndex);

            for i=iter
                %Loop through the signal, form array in calling pwelch
                data =
mouse.mouseRecording{i}.psdECogChannelDifference;
                psdData(:,i) =  data(freqLowIndex:freqHighIndex);
            end

            s = size(psdData);
            py = w;
            px = 1:1:s(2);
            mesh(px, py, psdData);
            %TODO: Figure out how to get the ticks labeled to show ct
```

```
name, for
            %now, will have to stick with doing it manually on final
plots
            xlabel('Time');
            ylabel('Frequency');
            zlabel('Spectral Power db/Hz');
            %put in mouse name to form plot name correctly
            title([mouse.mouseIdentity ' Change in spectral power over
time ECog channel difference']);
        end

        function [ mouse ] =
plotStationaryVersusRunningAveragePsd(mouse, recordingIndex,
ecogChannel)
            %Ensure all the input parameters and values have been
computed
            if (~mouse.hasPsdOfRunningInstancesComputed)
                fprintf('Computing PSD for all Running Instances');
                mouse = mouse.computePsdOfRunningInstances();
            end
            if (~mouse.hasPsdOfStationaryInstancesComputed)
                fprintf('Computing PSD for all Stationary Instances');
                mouse = mouse.computePsdOfStationaryInstances();
            end
            if (ecogChannel ~= 1 && ecogChannel ~= 2 && ecogChannel ~=
3)
                fprintf('Ecog Channel must be either 1, 2, or 3,
setting it to the default of 1');
                ecogChannel = 1;
            end
            if (recordingIndex < 1 || recordingIndex >
length(mouse.mouseRecording))
                fprintf('Invalid recording index passed for mouse
recording, defaulting to 1.');
                recordingIndex = 1;
            end

            %Plot the PSD avg for Running vs Stationary
            psdRunning =
mouse.mouseRecording{recordingIndex}.psdRunningInstancesData{ecogChanne
l};
            psdStationary =
mouse.mouseRecording{recordingIndex}.psdStationaryInstancesData{ecogCha
nnel};
            figure;
            hold on;
            p(1) = plot(psdRunning, 'r');
            p(2) = plot(psdStationary, 'b');
            %need to know how to show the label....
            str{1} ='Running';
            str{2} = 'Stationary';
            legend(p,str)
            xlabel('Frequency');
            ylabel('Spectral Power db/Hz');
            title([mouse.mouseIdentity ' Running versus Stationary
```

```
Average PSD, Recording ' num2str(recordingIndex) ', Channel '
num2str(ecogChannel)]);
        end

        %% set the recording number to be used, the startTime (second)
and the length of recording
        %  to show. Consider that time may need to be in ms.
        function [ mouse ] = plotEMGatTime(mouse, recordingIndex,
timeStart, timeLength)
            %Compute the PSd of the EMG sig3 for the given time
length...
            % do a two-panel plot of the PSD from start, and the raw
            % signal.
            if (recordingIndex < 1 || recordingIndex >
length(mouse.mouseRecording))
                fprintf('Invalid recording index passed for mouse
recording, defaulting to 1.');
                recordingIndex = 1;
            end

            [emgRawSignal, emgPsd, psdFreq] =
mouse.mouseRecording{recordingIndex}.computeEmgRawSignalAndPsd(timeStar
t, timeLength);
            fs = 400; %For mouse
            timeVec = 0:1/fs:(timeLength - 1/fs);
            %plot with thse results
            subplot(2,1,1)
            plot(timeVec, emgRawSignal);
            xlabel('Time (s)');
            ylabel('EMG raw signal (mV)');
            title([mouse.mouseIdentity ' EMG plot at time '
num2str(timeStart) ' on Recording ' num2str(recordingIndex)]);
            hold on
            subplot(2,1,2)
            plot(psdFreq,emgPsd)
            xlabel('Frequency');
            ylabel('Spectral Power db/Hz');
            title([mouse.mouseIdentity ' EMG Spectral Power, Recording'
num2str(recordingIndex)]);
        end
        %get the stationary vs. non-stationary comparison in here...

        %%%%%%% RPM %%%%%%%%%

        %%Plots a continuous stream of avg Rpm, currently not
delineated by
        %%recording number. This is mostly to get a feel of each
mouse's
        %%running characteristics and trends
        function [] = plotAvgRpmAllFiles(mouse)
            data = calculateAvgRpmAllFiles(mouse);
            %do a stem plot to show more cleanly
            plot(data,'ro'); %plots red o's with no line
            xlabel('time (  s)');
            ylabel('averaged wheel speed (rpm)');
```

```
        end

        %Concatenates all the avg Rpm recordings together into one file
        function [avgRpmAllFiles] = calculateAvgRpmAllFiles(mouse)
            num_files = length(mouse.mouseRecording);
            file_iterator = 1:num_files;
            avgRpmAllFiles = 0;
            for i = file_iterator
                mr = mouse.mouseRecording{i};
                avgRpmAllFiles = [avgRpmAllFiles mr.rpmAvgData{1}];
            end
        end

        function [rpmDensity] = calculateAvgRpmDensityAllFiles(mouse)
            rpmX = 0:1:140; %choose 200 as our upper limit that we'll
care to look at
            rpmY = zeros(1,length(rpmX));
            num_files = length(mouse.mouseRecording);
            iter = 1:num_files;
            for i = iter
                mr = mouse.mouseRecording{i};
                rpmAvg = mr.rpmAvgData{1};
                jter = 1:length(rpmAvg);
                for j = jter
                    %neat O(1) op to count rpm, uses rpm value for the
index to count
                    rpmY(round(rpmAvg(j)) + 1) = (rpmY(round(rpmAvg(j))
+ 1)) + 1; %index off by 1
                end
            end
            rpmDensity{1} = rpmX;
            rpmDensity{2} = rpmY;
        end
    end
end
```

Mouse Recording object

```
classdef rMouseRecording

    properties
        rpmFile_ch1
        rpmFile_ch2
        ecogFile
        rpmData %cell 1 corresponds to ch1, cell 2 to ch 2
        rpmAvgData %cell 1 is the avg rpm value computed from two
channels, cell 2 is a binary array for if the two rpm channels agreed,
or were significantly different.Values are 0 if they ARE significantly
different, or 1 if the values are within an acceptable range.
        ecogEdf
        trailingAveragePeriod
        %% runningInstancesData{channelNum, RecordingInstance}
        %  This is two channels, the first array is ch1 of the EEG
signal, and the second array is ch2 from the recordings
        runningInstancesData
        %%The filter used for selecting running instances by.
        runningInstancesFilterMethod
        %% Collection of the instances where the mouse was deemed
stationary by FindStationaryInstances filter
        stationaryInstancesData
        %%Difference signal PSD
        psdECogChannelDifference
        %%psdRunningInstancesData, first array is the ch1 of EEG signal
        %%from running instances, second cell array is ch2.
        psdRunningInstancesData
        %%psdStationary is the averaged PSD for the recording for the
        %%stationary instances
        psdStationaryInstancesData
        %%the array of w from the psd computation
        psdFrequencyValues
        %%poles from Dynamic AR model
        polesDynamicModel
        %%Offset in time between the Pinnacle and Clocklab systems
        recording_offset
        %%Handles which of the systems were started first to handle
offset
        recording_offset_Pinnacle_StartedFirst
    end

    properties (Access = private)
        %PSD variables, local only, change values here for calculation
        %parameters
        psdFs = 400;
        psdOverlap = 40;
        psdWindow = blackmanharris(50);
    end

    methods
        function obj = rMouseRecording(ecogFile, rpmFile_ch1,
rpmFile_ch2, trailingAvgPeriod)
            obj.rpmFile_ch1 = rpmFile_ch1;
```

```
            obj.rpmFile_ch2 = rpmFile_ch2;
            obj.ecogFile = ecogFile;
            obj.trailingAveragePeriod = trailingAvgPeriod;
            obj.ecogEdf = sdfopen(ecogFile);
            obj.runningInstancesData = [];
            obj.runningInstancesFilterMethod = '';
            obj.polesDynamicModel = [];
            %edf wrt cl
            %need check for negative... ie ecog was started before cl
on the first
            %recording
            %also, offset is same for both rpm channels
            initialOffset = findTimeDiff(ecogFile, rpmFile_ch1) %make
this into a static function
            if (initialOffset < 0)
                obj.recording_offset_Pinnacle_StartedFirst = false;
                obj.recording_offset = -initialOffset;
            else
                obj.recording_offset = initialOffset;
                recording_offset_Pinnacle_StartedFirst = true;
            end
            rpmTotal_ch1 = ReadCLfile(rpmFile_ch1);
            rpmTotal_ch2 = ReadCLfile(rpmFile_ch2);

            recording_length = obj.getRecordingLength()

            %Max length,


            %if (obj.recording_offset_Pinnacle_StartedFirst) %pad it if
it was second
                obj.rpmData{1} =
rpmTotal_ch1(obj.recording_offset:(obj.recording_offset +
recording_length))
                obj.rpmData{2} =
rpmTotal_ch2(obj.recording_offset:(obj.recording_offset +
recording_length));
            %else
            %    obj.rpmData{1} =
rpmTotal_ch1(obj.recording_offset:(recording_length));
            %    obj.rpmData{2} =
rpmTotal_ch2(obj.recording_offset:(recording_length));
            %end
            %obj.rpmData{2} = obj.rpmData{1};

            %Create new method, computeRunningAverageTwoChannels to set
            %rpmAvgData
            avg1 = runningAverage(obj.rpmData{1},
obj.trailingAveragePeriod);
            avg2 = runningAverage(obj.rpmData{2},
obj.trailingAveragePeriod);
            obj.rpmAvgData{1} = (avg1 + avg2) / 2; %merging avgs
together
            sig_diff = 60/(trailingAvgPeriod + 1); %considered to be
significant if two channel differ by more than one count, case where
```

```
wheel is not doing full rotations, just rocking back and forth only
registering on one channel
            sdArray(length(avg1)) = 0;
            avgIter = 1:1:length(avg1);
            for a = avgIter
                if(abs(avg1(a) - avg2(a)) > sig_diff)
                    sdArray(a) = 0;
                else
                    sdArray(a) = 1;
                end
            end
            obj.rpmAvgData{2} = sdArray;

        end

        %% Calculates the averaged PSD for each recording's set of
running instances.
        %
        function [mouseRecording] =
computePsdOfRunningInstances(mouseRecording)
           runningInstancesArray = mouseRecording.runningInstancesData;
           if(~isempty(runningInstancesArray))
               [mouseRecording.psdRunningInstancesData{1},
mouseRecording.psdFrequencyValues] = ...

rMouseRecording.computeAveragedPsd(runningInstancesArray, 1,
mouseRecording.psdFs, mouseRecording.psdOverlap,
mouseRecording.psdWindow);
               [mouseRecording.psdRunningInstancesData{2},
mouseRecording.psdFrequencyValues] = ...

rMouseRecording.computeAveragedPsd(runningInstancesArray, 2,
mouseRecording.psdFs, mouseRecording.psdOverlap,
mouseRecording.psdWindow);
           else
               fprintf('Compute Psd in mouseRecording called before
Running Instances were calculated');
               mouseRecording.psdRunningInstancesData = [];
               mouseRecording.psdFrequencyValues = [];
           end
        end

        %% Calculates the averaged PSD for each recording's set of
stationary instances.
        %
        function [mouseRecording] =
computePsdOfStationaryInstances(mouseRecording)
           stationaryInstancesArray =
mouseRecording.stationaryInstancesData;
           if(~isempty(stationaryInstancesArray))
               [mouseRecording.psdStationaryInstancesData{1},
mouseRecording.psdFrequencyValues] = ...

rMouseRecording.computeAveragedPsd(stationaryInstancesArray, 1,
mouseRecording.psdFs, mouseRecording.psdOverlap,
```

```
mouseRecording.psdWindow);
                [mouseRecording.psdStationaryInstancesData{2},
mouseRecording.psdFrequencyValues] = ...

rMouseRecording.computeAveragedPsd(stationaryInstancesArray, 2,
mouseRecording.psdFs, mouseRecording.psdOverlap,
mouseRecording.psdWindow);
            else
                fprintf('Compute Psd in mouseRecording called before
Stationary Instances were calculated');
                mouseRecording.psdStationaryInstancesData = [];
                mouseRecording.psdFrequencyValues = [];
            end
        end

        %% For the analysis to try to subtract out the reliance on the
Reference node
        %  from EEG1 - EEGref = Channel 1, and EEG2 - EEGref = Channel
2,
        %  so then this signal will be Channel1 - Channel2 = EEG1 -
EEG2.
        %  Which gives at least the EEG only signal without the
potentially
        %  skewed results from where the EEGref was implanted in muscle
        %  from the bad diagram. Then from this difference signal, we
take
        %  the PSD and average together for an overall signal that
could
        %  show subtle circadian patterns.
        %  THIS COULD be modified to store each of the differences if
there
        %  is a different analysis than the averaged PSD.
        function [ mouseRecording ] =
computeChannelDifferenceRunning(mouseRecording)
            runningInstancesArray = mouseRecording.runningInstancesData;
            if(~isempty(runningInstancesArray))
                %build array from diffing two running signal channels.
                iter = 1:1:length(runningInstancesArray);
                diffRawSig{length(runningInstancesArray)} = 0;
                for i = iter
                   %Subtract signals
                   diffRawSig{i} = runningInstancesArray{1,i} -
runningInstancesArray{2,i}; %Check operation is correct
                end
                [mouseRecording.psdECogChannelDifference,
mouseRecording.psdFrequencyValues] = ...
                    rMouseRecording.computeAveragedPsd(diffRawSig, 1,
mouseRecording.psdFs, mouseRecording.psdOverlap,
mouseRecording.psdWindow);
            else
                fprintf('Recording Instances of Running array is empty
when compute Channel difference called, Run Compute first');
                mouseRecording.psdECogChannelDifference = [];
            end
        end
```

```
        %% Computes the EMG signal and shows the raw signal at a single
instance in time...
        %
        function [ emgRawSignal, emgPsdSignal, psdFreq ] =
computeEmgRawSignalAndPsd(mouseRecording, timeOffset, timePeriod)
            %PSD parameters
            fs = 400;
            overlap = 40;
            win = blackmanharris(50);

            %Verify that the timeOffset and timePeriod are valid
inputs.
            edf = mouseRecording.ecogEdf;
            lengthOfEcogRecording = edf.Dur * edf.NRec;
            if(timeOffset < 1)
                frintf('Time Offset passed in is less than 1, setting
timeOffset to 1');
                timeOffset = 1;
            elseif (timeOffset >= (lengthOfEcogRecording - timePeriod))
                timeOffset = lengthOfEcogRecording - timePeriod;
                frintf(['Time Offset and time period go beyond the
length of recording, moving time back to, ' num2str(timeOffset)]);
            end

            rawSig = sdfread(edf, timePeriod, timeOffset);
            emgRawSignal = rawSig(:,3);
            %PSD for the emg signal
            [pxx,psdFreq] =
pwelch(emgRawSignal,win,overlap,[],fs,'onesided');
            emgPsdSignal = 10*log10(pxx);
        end

        %% Models all the running instances by looking at the poles of
the dynamic system using
        %  the AR Burg method. Set the Model order to use, and if the
model
        %  should normalize the poles based on the largest pole value
from
        %  all recordings.
        function [ mouseRecording, maxValueRoots ] =
computePolesFromAutoRegressionModel( mouseRecording, modelOrder )
            %TODO: May need to separate out the Normalizing part...
Pull
            %out the normalizing code up one level to the Mouse level,
and
            %put the straight computation of the poles at this level.
TODO.
            numRunningInstances =
length(mouseRecording.runningInstancesData);
            %Make the PolesOfDynamicSystem (below is rootsResult), and
the GetOrderOfDynamicSystem
            %methods on the mouseRecording.
            mouseRecording.polesDynamicModel(numRunningInstances,
modelOrder) = 0;
```

```
            maxValueRoots(modelOrder) = 0; %holds the max value to
normalize to for each order of the Model
            orderIter = 1:1:modelOrder; %iteration to go through all
the orders poles to get max value
            %First run through and compute all the poles
            if(numRunningInstances >= order) %order rather than 0
                iter = 1:1:numRunningInstances;
                for i = iter
                    arCoefficients =
arburg(mouseRecording.runningInstancesData{i},order);
                    arModelRoots = roots(arCoefficients);
                    mouseRecording.polesDynamicModel(i) = arModelRoots;
                    %Then compute the max values to return if
normalization
                    %is done later.
                    for oI = orderIter
                       rootAbsVal = abs(arModelRoots(oI));
                       if (rootAbsVal > maxValueRoots(oI))
                           maxValueRoots(oI) = rootAbsVal;
                       end
                    end
                    %Hold onto for now, had been taking the max as the
95%
                    %largest value.
                    %mVal = maxArray(topFivePercentileMaxLength);
%topFivePercentileMaxLength);
                    %if(n_mouse_rec1(recNum) > mVal)
                    %maxArray(topFivePercentileMaxLength) =
n_mouse_rec1(recNum);
                    %maxArray = sort(maxArray, 'descend');
                    %end
                end
            else
                fprintf(['The model order needs to be less than the
number of instances in the data set, there are only '
                    num2str(numRunningInstances) ' running
instances']);
            end
        end

        %% Note: Don't forget that the rpm signal and ECog signal are
at different sampling Rates (If plotting on the same
        %        plot, you'll need to expand out the number of data
points
        %        to be on the same axis. This is just to get a segment
of
        %        raw data to simplify access to easily get raw data to
        %        experiment with from the command line.
        %  Note2: It is also possible (if going to the end of the
        %        recording) that there will only be one dataStream (ie
RPM
        %        only or ECOG/EMG only) because they don't end
        %        simultaneously. So stay away from the final seconds of
a
        %        recording.
```

```
        function [rpmOutput, rawEcog1, rawEcog2, rawEmg] =
getRawSignalAndRpmAtTime(mouseRecording, timeOffset, timePeriod)
            edf = mouseRecording.ecogEdf;
            lengthOfEcogRecording = edf.Dur * edf.NRec;
            if(timeOffset < 1)
                frintf('Time Offset passed in is less than 1, setting
timeOffset to 1');
                timeOffset = 1;
            elseif (timeOffset >= (lengthOfEcogRecording - timePeriod))
                timeOffset = lengthOfEcogRecording - timePeriod;
                frintf(['Time Offset and time period go beyond the
length of recording, moving time back to, ' num2str(timeOffset)]);
            end
            if (timePeriod < 1)
               timePeriod = 1;
               fprintf('TimePeriod has to be longer than 1 second,
defaulting to 1');
            end

            rawSig = sdfread(edf, timePeriod, timeOffset);
            rawEcog1 = rawSig(:,1);
            rawEcog2 = rawSig(:,2);
            rawEmg = rawSig(:,3);
            %And then RPM...
            rpmOutput(timePeriod) = 0;
            rpmCh1 = mouseRecording.rpmData{1};
            rpmCh2 = mouseRecording.rpmData{2};
            iter = 1:1:timePeriod;
            for i = iter
                rpmOutput(i) = (rpmCh1(timeOffset + i - 1) +
rpmCh2(timeOffset + i - 1))/2;
            end
        end

        %%returns value in seconds, size guarentees recording for both
rpm and ecog values
        function recording_length = getRecordingLength(mRecording)
            edf = mRecording.ecogEdf;
            lengthOfEcogRecording = edf.Dur * edf.NRec
            %okay to call ReadCLfile again, it's fast and code simpler
this way
            lengthOfRpmRecording =
length(ReadCLfile(mRecording.rpmFile_ch1)) -
mRecording.recording_offset

            %use the lesser of the two, drop anything that was recorded
after the
            %other (rpm or ecog) was stopped recording
            if(lengthOfEcogRecording < lengthOfRpmRecording)
                recording_length = lengthOfEcogRecording;
            else
                recording_length = lengthOfRpmRecording;
            end
        end
    end
```

```
    methods (Static = true)
        %% Compute and Store the averaged PSD for multiple instances to
average together
        function [avgPsd, w] = computeAveragedPsd(rawSignalArray,
channel, fs, overlap, windowingFunction)
            if (channel == 1 || channel == 2)
                numInstances = length(rawSignalArray);
                avgPsd(129, 1) = 0;
                if (numInstances > 1)
                    numIter = 1:1:numInstances;
                    for nI = numIter
                        sig1 = rawSignalArray{channel, nI};
                        [pxx,w] =
pwelch(sig1,windowingFunction,overlap,[],fs,'onesided');
                        avgPsd = avgPsd + 10*log10(pxx);
                    end
                    avgPsd = avgPsd / numInstances;
                end
            else
                fprintf('Invalid channel passed for computing PSD,
passing back empty results');
            end
        end
    end
end
```