

**Automated 3D brain surface reconstruction with heat maps based on underlying
neurological signals.**

**BE 223A Programming Lab
for Medical and Imaging Informatics I**

Mohammadali Alidoost,
Amy Cummings,
Ge Fang,
Anders Olav Garlid,
James Go,
David Gordon,
Yannan Lin,
Jake Pensa,
Joseph Tsung,
and
Zhaoqiang Wang

Correspondence: All correspondence may be addressed to any of the contributing authors and developers, listed below with their specific contributions to the project.

AC (alcummings@mednet.ucla.edu): Signal Analysis, channel quality control

MA (maalidoost@ucla.edu): Signal Analysis, PAC

GF (guojifang17@gmail.com): Signal Analysis, audio processing

AOG (aogarlid@gmail.com): Database Integration, File I/O, and UI

JG (jamescookgo@gmail.com): ECoG Channel Registration

DG (d.gordon@ucla.edu): Signal Analysis

JP (jake.pensa@gmail.com): Signal Analysis, Band Power over Time

JT (josephsung@g.ucla.edu): ECoG and Talairach Channel Localization

YL (allyn1982@ucla.edu): Channel Plotting

AW (zqwang9@g.ucla.edu): Heatmap Generation

TABLE OF CONTENTS

	<i>pp.</i>
A. Background/Introduction and Rationale	3-7
B. Project Component Chapters:	8-68
I. Database Integration and File I/O	8-12
II. Signal analysis	13-42
A. Channel QC	13-16
B. Channel scores	17-21
C. Band Power over Time	22-28
D. PAC	29-37
E. Audio	38-42
III. Electrode localization	43-50
IV. Electrode channel labeling	51-56
V. Channel plotting and coordinate corrections	57-62
VI. Heatmap generation and final visualization	63-68
C. Discussion, Conclusions, and Future Work	69
D. Reference	70-71

A. INTRODUCTION AND PROJECT SUMMARY

BACKGROUND AND RATIONALE. Traumatic brain injury and neurological disorders such as cerebral palsy and amyotrophic lateral sclerosis (ALS) can impair neural transduction and result in the loss of speech capabilities as well as devastating seizures, dramatically impacting affected individuals and severely limiting their ability to lead a normal life¹. The benefits and improvement in quality of life that these patients would experience from assisted communication devices such as a brain-computer interface (BCI) cannot be overstated. However, significant limitations in our understanding of neural communication and speech formation present barriers that impede the development of such technologies.

We have developed the BrainSweet software platform to aid clinicians and researchers in bridging this knowledge gap by automating the generation of rich, 3D visualizations of neural signals from patient imaging and signal data. The *specific scientific problem we address* in the development of the BrainSweet platform is the lack of an efficient mechanism to integrate brain images with neural signal data. This platform automatically generates 3D representations of the cortical surface of the human brain from magnetic resonance imaging (MRI) and computed tomography (CT) scans and superimposes a heat-map of electrical activity measured during speech experiments or seizure monitoring using electroencephalography (EEG) or electrocorticography (ECoG). A variety of Python packages and tools such as BrainSuite (www.BrainSuite.org) can handle aspects of automated MRI processing, including skull stripping for 3D visualization of the brain, but a tool to automatically

visualize neural signals does not yet exist. An overview of the project workflow is depicted in **Figure 1**.

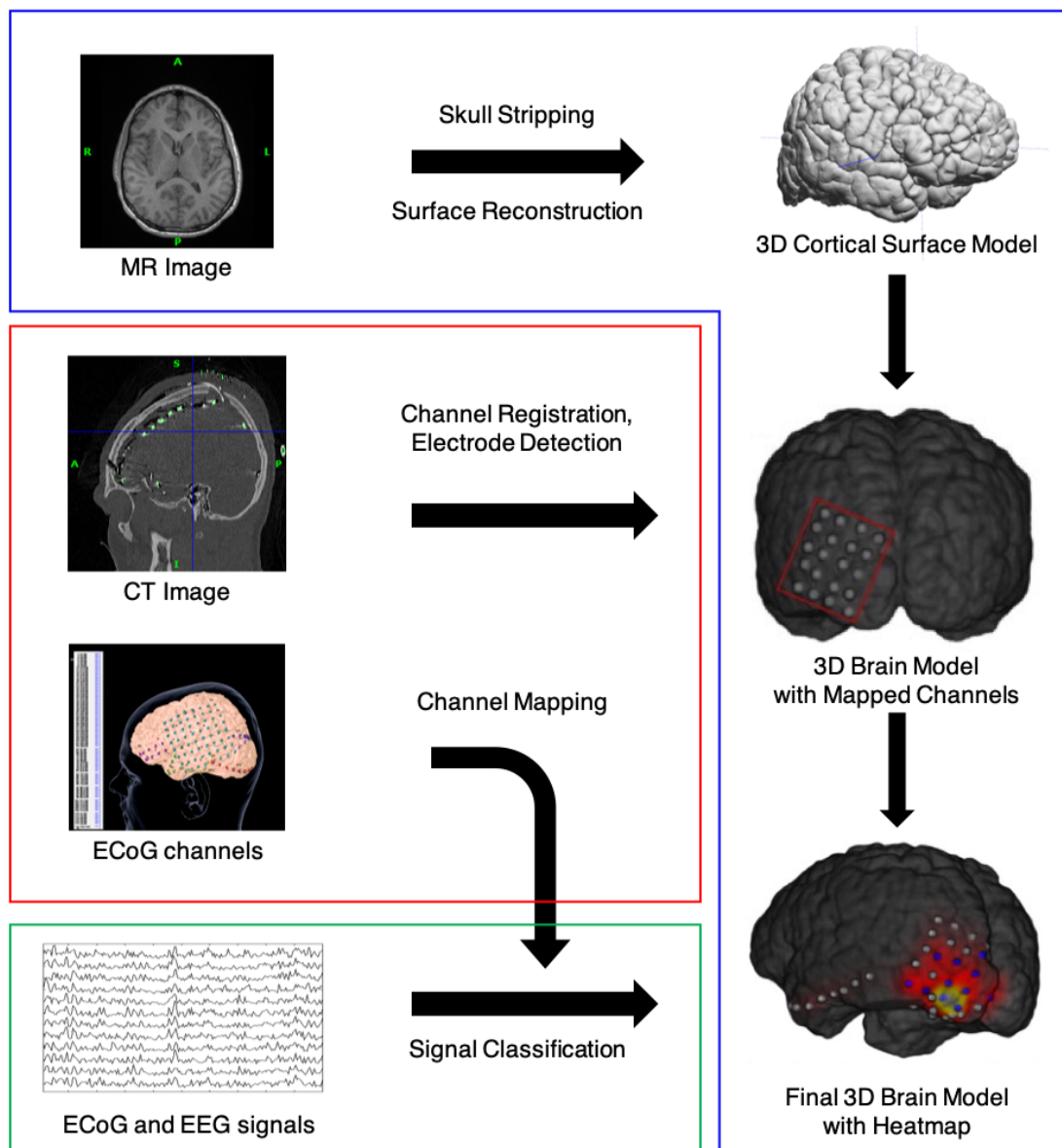


Figure 1: BrainSweet platform overview. The BrainSweet platform takes inputs from the user including the subject MR and CT images, ECoG channel map, and EEG and ECoG signal files. These inputs are then subjected to various processing pipelines to produce intermediate 3D brain models, ultimately leading to the final 3D brain model with a heatmap to display channel scores from signal analysis.

SIGNIFICANCE AND INNOVATION. Automated 3D brain mapping from patient imaging and neural signal recordings will provide tremendous advantages to clinicians and researchers alike by integrating these data into highly informative models of their patients' and research subjects' brains. No method currently exists to efficiently generate such images in an automated fashion, and the current approach is tremendously time-consuming, technically challenging, and tedious. BrainSweet is *significant* in that it establishes a user-friendly platform for efficient image analysis and enhance the utility of patient electrophysiology and imaging data, ultimately facilitating increased understanding of neurophysiology in speech impairment and seizures for improved patient care and further development of assistive technologies and BCIs.

Furthermore, the BrainSweet platform is *technically innovative* in that it makes use of multiple sources of data and combines them in a simple, easy to use interface that requires only the input images, electrical data, and some minor guidance by the user for proper electrode localization.

CHAPTER OVERVIEW. In the following chapters, the authors present a detailed description of each software component and approaches used to develop and run the BrainSweet platform.

Database integration and file I/O, presented in (**Chapter I**), entails the development of the PostgreSQL database, the Python wrapper, and the user interface. The user-facing component of the platform is tasked with acquiring subject and experimental details as well as file locations and desired analysis pathways from the

user. The Python wrapper orchestrates each of the various components of the platform, feeding data inputs and outputs to the database and/or directly between components, as appropriate.

Signal analysis, presented in **Chapter II**, comprises 5 distinct components, including channel quality control, channel scoring, band power over time, phase-amplitude coupling, and audio analyses. Channel quality control (**Chapter II.A**) evaluates EEG channels to determine the proportion of time that their input falls within expected frequency ranges. Channel scoring (**Chapter II.B**) utilizes the onset of the signal for the EEG data to develop a classification algorithm that assesses the importance of each channel. Band power over time (**Chapter II.C**) is used to determine the relative power of each channel at various frequency bands over a specified time period to look for the presence of various neurological signals that are distinguishable by frequency such as the alpha or beta bands. Phase-amplitude coupling (**Chapter II.D**) explores how much the phase of a slow oscillation, e.g. Theta, is coupled with the amplitude of a fast oscillation, e.g. Gamma. This kind of information can be informative in understanding brain functions. Audio analysis tool (**Chapter II.E**) , focus on analyzing the correlation between sound articulation and neurological signals - e.g., combining with visualization methods, such as heatmap, can provide information about the regions on the cerebral cortex that relate to the contraction of vocal cord, which leads to the production of sounds.

Electrode localization, presented in (**Chapter III**), covers the methodology used for locating ECoG electrode coordinates from CT-MR registration and Talairach-MR

registration. Electrode channel labeling, presented in (**Chapter IV**), provides the user with a simple interface and workflow for labeling ECoG electrodes located by CT-MR registration. Channel plotting and coordinate corrections, presented in (**Chapter V**), are two auxiliary components facilitating the understanding of regional brain activities by plotting the electrodes onto the cortical surface. Lastly, heatmap generation and final visualization, presented in (**Chapter VI**), provides a user interface to view the data and analyse the activity of signals in a easy and straightforward way.

CHAPTER I. DATABASE INTEGRATION AND FILE I/O

Author: Anders Olav Garlid

INTRODUCTION AND BACKGROUND. This particular component of the platform focuses on a database integration approach for data management and information transfer between the major components of the platform, from processing images, registering EEG or ECoG electrodes, classifying neural signals, and producing the final 3D models. We have implemented the open-source PostgreSQL object-relational database management system (ORDBMS) to manage data I/O across the platform, chosen in part for its emphasis on extensibility and the potential to accomplish the stretch goal of creating a more advanced user interface with additional deployment options and use cases. We have implemented a Python script to act as the user-interface, posing requests for patient and experiment information as well as file inputs and analysis methods. This program can be accessed through the user's computer terminal or command line interface, where they will be presented with simple requests for the necessary information and files.

The primary benefit of developing this platform with an integrated database and file management wrapper is to provide the end-user with a fully contained experience and the ability to construct the 3D images of the cortical surface without the need to conduct multiple steps in several different programs. Furthermore, complications regarding extensive memory usage by each component can be avoided by running each sequentially. Finally, because these components require specific input and output

file types, the overall project wrapper is able to unify the overall pipeline and deal with these discrepancies within the program, rather than requiring the user to conduct any conversions between different file types.

METHODS AND IMPLEMENTATION.

1. PostgreSQL database. For the database integration component of the BrainSweet modeling platform, we have created an object-relational database to manage data files across the platform, providing appropriate containers for incoming data and hooks for subsequent steps to access the necessary content. We utilized the open-source PostgreSQL, an object-relational database management system (ORDBMS) with an emphasis on extensibility.² As a general-purpose ORDMS, PostgreSQL supports custom functions developed using different programming languages such as C/C++, Java, Python, and MATLAB (REF:PostgreSQL). Because the BrainSweet platform comprises several major components written in different languages, this system is particularly well-suited for the task of managing the data used and produced by these different approaches. As an extensible database management system, this allows definition of custom data types, index types, functional languages, plugins and optimizers.³ Again, this will provide the necessary flexibility to handle the variety of file types managed by this platform, including imaging data, signal data, audio files, and label data, each of which will require a specific data structure for optimum handling. The specific database schema, depicted in **Figure I.1**, contains a 'subjects' relation for subject information, experiment type, and image file paths; a 'signals' relation to house

the signal file paths; a 'channels' relation for the channel coordinates; an 'eeg' relation to store the standard EEG 10-20 system coordinates (provided in a standard data file within the program directory); and a 'scores' relation to house the channel scores derived from the various signal processing methods.

2. File I/O and component orchestration. The overall Python wrapper manages all user input and interaction, orchestration of each component based on the experiment type, and handling of data files utilized by the platform, including images from MRI and CT as well as channel mapping data for EEG and ECoG electrodes; signal data from

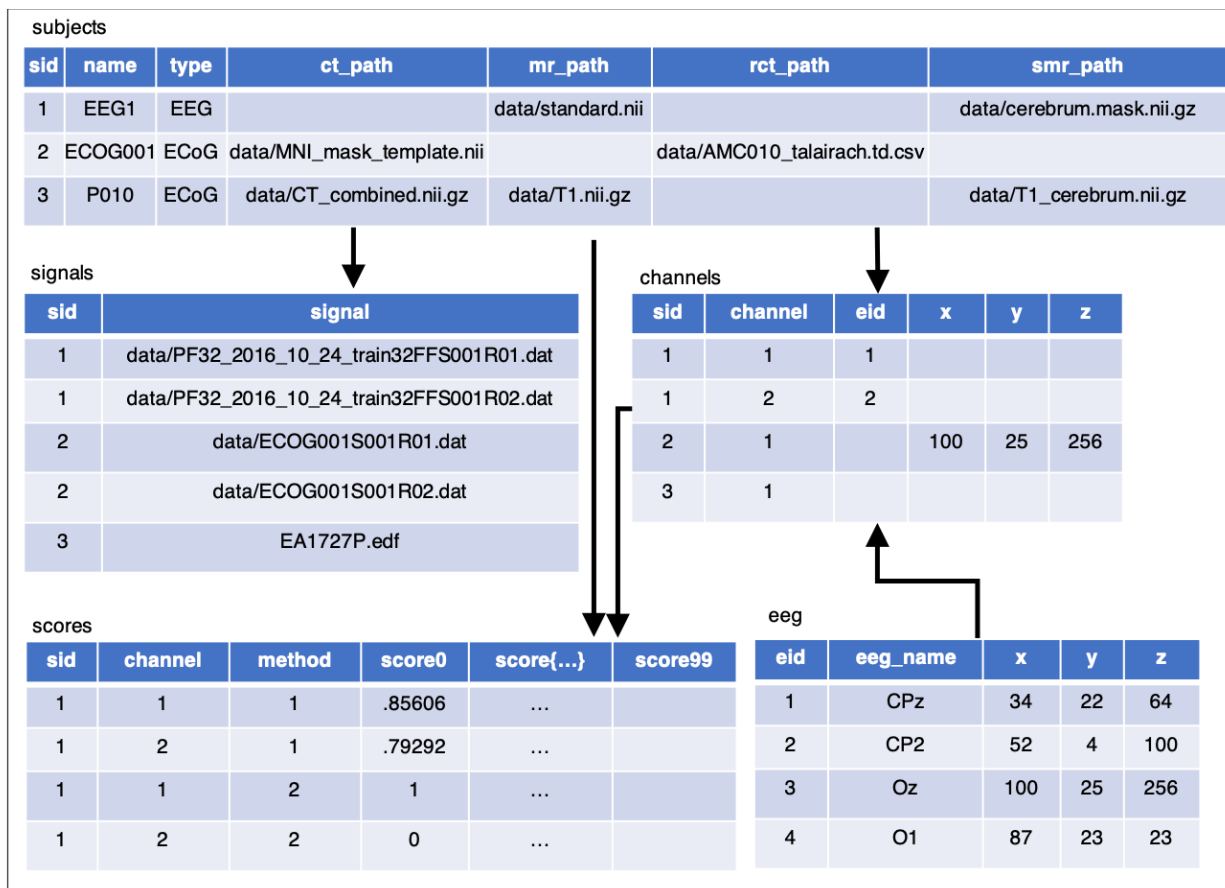


Figure 1: PostgreSQL database schema. contains a 'subjects' relation for subject information, experiment type, and image file paths; a 'signals' relation to house the signal file paths; a 'channels' relation for the channel coordinates; an 'eeg' relation to store the standard EEG 10-20 system coordinates (provided in a standard data file within the program directory); and a 'scores' relation to house the channel scores derived from the various signal processing methods.

EEG and ECoG as well as audio files from speech-related experiments; and label data for channel coordinates and offsets. Specific file formats are detailed in **Table I.1**.

DISCUSSION AND FUTURE DIRECTIONS. The BrainSweet platform semi-automatically generates 3D representations of the cortical surface of the human brain from patient imaging data (including MRI and CT) and superimposes onto this image a heat-map corresponding to the electrical signals recorded via EEG or ECoG during speech experiments or seizure monitoring.

BrainSweet is a reliable and effective method for automated brain modeling through a user-friendly and openly accessible platform. Relative to the current paradigm of manual image generation, our platform will facilitate high-throughput integration of brain imaging and neural signals into 3D models, paving the way for more effective treatments and therapies as well as analysis of large patient cohorts. By enabling clinicians and researchers to visualize the neural signals recorded during

Data type	File format	
Imaging data		
CT	NIFTI	DICOM
MRI	NIFTI	DICOM
Channel map	JPG	XLS
Signal data		
EEG	BCI2000	
ECoG	BCI2000	EDF
Audio files	WAV	
Label data		
Offsets	XLS	
Channel coordinates	XLS	

Table 1: Data and file types utilized by the BrainSweet platform.

speech experiments or seizure monitoring on a 3D reconstruction of their subjects' brains, we hope to facilitate significant discoveries and advances in neuroscience and clinical neurology, including the realms of speech pathology, seizure localization and propagation, and the development of

new technologies for interpreting neural signals and their relation to speech, movement, and beyond.

This framework is well-suited to handle the relatively simple data management demands of the current BrainSweet platform. It is also expected to facilitate implementation of additional features to the platform and further development of the user interface through integration with node.js and react.js. This will allow further instantiations of the program to handle different use cases and production as a web-based application in the future.

CHAPTER II.A. SIGNAL ANALYSIS: Channel Quality Control

Author: Amy Cummings

- Input: path to EEG CSV
- Dependencies: Python 3.6 modules Pandas, SciPy, NumPy
- Output: channel list with scores
 - scores are a number between 0-1 that indicate the proportion of the channel input that falls within an acceptable range
 - 1 indicates a properly functioning channel, 0 indicates no desired input is present (i.e. a bad channel)

INTRODUCTION

Understanding the interplay of large-scale cortical systems such as the human brain with internal and external stimuli relies on functional imaging and source localization. Yet source localization of electric potentials and magnetic fields may be obscured by noise ranging from hardware failure to involuntary muscle contractions. Thus, channel preprocessing and quality control is an important initial step to signal processing to allow for manual electrode adjustment and/or removal of channels that are not providing reliable information.

METHODS AND IMPLEMENTATION

Channel quality control (channel-qc) is a method to identify properly functioning channels in an EEG. It requires Python 3.6 and calls Pandas, SciPy, and NumPy modules. The input required is the path to an EEG CSV file, which is provided by a separate component of this project if the EEG is initially in EDF or DAT format. The output is a list of values between 0 and 1 that correspond to the proportion of the total channel input that falls in the expected range for a normal EEG, which is set at 0.5-30 Hz, consistent with resting wakefulness. This information is then fed to the visualization component for electrode mapping.

The module functions by creating a pandas reading frame (<https://pandas.pydata.org>) that references the EEG CSV and selects the portions of the CSV that contain channel information. It has been set to trim the first row and first two columns of the CSV that tend to provide non-channel information. To manually adjust the CSV reading frame, line 21 may be updated to provide the starting coordinates of the reading frame.

```
21    Only_Channels = eeg.iloc[1:,2:]
```

This reading frame is then passed to the SciPy notch filter (<http://www.scipy.org/>). The SciPy notch filter is set with a sample frequency of 256 Hz, notch filter at 60 Hz to remove electric interference, and quality score of 30, based on usual parameters. After the filter is applied, a matrix with normalized frequencies for each channel is created.

While this section is automated to run based on usual EEG input, if it is so desired, the source code can be manually adjusted to process different frequencies (fs), filters (f0), or quality scores (Q) in lines 24-26.

```
24  fs = 256.0  
25  f0 = 60.0  
26  Q = 30.0
```

A loop function runs each channel through the equivalent of PyEEG's `bin_power` function.⁴ The input for this step is an array of each channel's input from the SciPy notch filter normalized matrix. These arrays are then passed through NumPy's one-dimensional Fourier transform function, then an absolute value function, and then a power function that determines the proportion of the input that is between 0.5 to 30 Hz. An array of channel scores is then printed to the database and visually represented as described later in this document.

Manual adjustment may be made in line 49 to adjust for different band width or frequencies by changing the [0.5,30] input. The replacement value for 0.5 would represent the minimum desired Hertz and the replacement for 30 the maximum desired Hertz. If there is a different frequency than 256, this also may be adjusted in the code below.

```
49  print([x],[((bin_power(Y[:,x], [0.5,30], 256))[1][0])
```

Note there are no input requirements for the length of the CSV or amount of time captured. The input EEG CSV in its entirety is considered. Mapping in real time may enable identification of probes that need to be adjusted before additional analysis is run or inform which channels should be removed from further analysis.

FUTURE DIRECTIONS

Future versions may include the ability to remove channels based on channel_qc score before other analysis is applied and/or salvage portions of channels that may provide reliable information.

CHAPTER II.B. SIGNAL ANALYSIS: Channel Scores

Author: David Gordon

- Input: EEG CSV
- Language: Python Version 3.6
- Python Libraries: Pandas, NumPy, Scikit-learn
- Output: Channel importance scores

Introduction:

Electroencephalography (EEG) signal data was acquired from participants using the P300 Speller brain-computer interface (BCI) with 32 electrodes. Figure 1 shows a representation of the 32-electrode configuration (non-shaded), where the midline represents 0 and as it moves outwards it counts upwards, with even to the right and odd to the left. The system displays a 6x6 character grid with row and column flashes (using the row column paradigm), which has 50-millisecond flash duration and 3.5 second pauses between selections.⁵ The 6x6 character grid is composed of 36 characters including 26 letters and 10 digits. The EEG signal data captured may contain information regarding the importance of the electrode localization. Moreover, by routinely capturing EEG features without knowledge regarding the importance of the electrode localization, we leverage resources that may be more effectively utilized. The specific aim of this project is to apply a classification algorithm to select a subset of important features that discriminate attended from non-attended signals so that we may be able to more effectively target important features, the accuracy of the channel,

and utilize this information in combination with electrode localization to generate heat-maps that may improve BCI near real-time capabilities.

Methods:

To accomplish these tasks, this project was broken down into a series of preprocessing steps that were performed prior to applying supervised machine learning. More specifically, the EEG data provided contains the variables stimulus code and stimulus type, which is in addition to the channel data, and was utilized to create a new variable called onset indices. Stimulus code contains values 1-12, where 1-6 represents the 6 rows in the character grid and 7-12 represents the 6 columns in the character grid.⁵ Moreover, stimulus type is a binary variable, where 1 represents a stimulus and 0 represents no stimulus. Figure 2 below shows a BCI display monitor, highlighting the 6x6 character grid. To further illustrate how stimulus code and stimulus type work together, we offer the following example:

Example Scenario: Patient A is using the BCI and staring at the display monitor, concentrating on the character 'T' in the 6x6 character grid, while rows and columns flash. Patient A has a stimulus type = 1 at stimulus code = 4 (representing the fourth row) and at stimulus code = 8 (representing the second column), which corresponds to character 'T'.

In the example above, Patient A had an attended signal and therefore would have an onset indice = 1 for the above instance. More specifically, to determine the onset of a stimulus for a target character (i.e. 'T') in the character grid, we created a formula that essentially calculates the onset indice for each attended and non-attended signal in the dataset, which we demonstrate in the code below:

```
OnsetIndices = []
count = 0
labels = []
for i in range(len(Stimulus_Code)-1):
    if Stimulus_Code[i+1] > 0 and Stimulus_Code[i] == 0:
        OnsetIndices.append(1)
        labels.append(Stimulus_Type[i+1])
    else:
        OnsetIndices.append(0)
        count = count + 1
```

Note in the code above, we were able to determine if a stimulus was present for a given target character in the character grid. Therefore, we now need to account for the duration of the attended signal, which is 600 milliseconds (0.6 seconds), as well as consider the sampling frequency of the BCI, which is 256 Hz, to determine how to best downsample the data, which is calculated in the formula below:

$$\text{Sampling Frequency (256 Hz)} \times \text{Time After Stimulus (.6 seconds)} = 150$$

Note from the formula above, we could expect the signal to transmit for about 150 time point features. Therefore, we can apply the downsample into the classification algorithm as essentially a temporal dimension reduction step where number of features

= 150, prior to performing classification. The next step is to select an appropriate classification algorithm given our objective to determine channel importance. It is well established that the random forest classifier is effective for classification tasks. As outlined in the Scikit-learn documentation, a random forest classifier fits a number of decision tree classifiers on sub-samples of the data and uses averaging to improve predictive accuracy and control for potential over-fitting.⁶ It is important to note that the sub-sample size is the same as the input sample size, but with replacement when using bootstrap. In this project, we performed a 70/30 split for the training and test set, respectively. Moreover, we selected the number of trees in the forest to be 100. Furthermore, we used the Gini impurity to measure the quality of the split. Additionally, we set the number of features to consider when looking for the best split to be $\sqrt{150}$, which is recommended for classification tasks.

Results and Discussion:

Figure 3 shows the channel importance scores. It seems the random forest classifier performed well for each of the channels, with only two channels having an accuracy below a 0.80 threshold. The two channels below a 0.80 threshold were FC5 and P2. The goal of this project was to develop a classification algorithm to determine an accuracy score for each of the 32 channels using the EEG data. By developing the onset indices variable, we were able to create our label, where 1 = attended signal and 0 = non-attended signal. Moreover, we were able to downsample the features, using the BCI sampling frequency (256 Hz) x time after stimulus (.6s) = 150. In doing so, we

were able to apply a random forest classifier that performed moderately well for each of the channels and output the scores representing EEG channel importance to the scores table in the database.

Figure 1. 32 Electrode Configuration

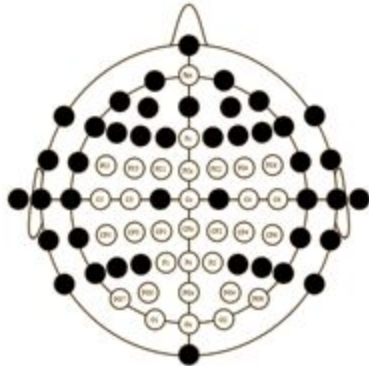


Figure above courtesy of William Speier, PhD.

Figure 2. Brain Computer Interface Display Monitor

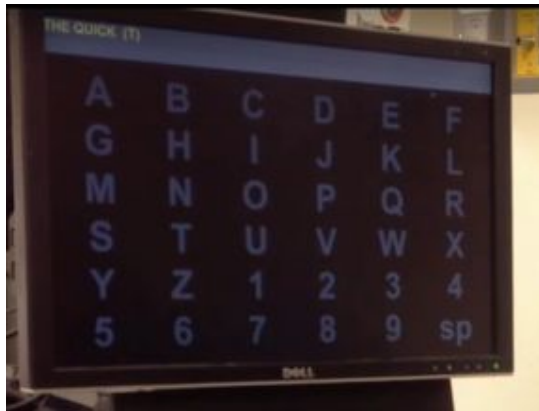


Figure above courtesy of UCLA Neurosurgical Brain Mapping and Restoration Lab

Figure 3. Channel Importance Scores

```
[0.8686868686868687, 0.8484848484848485, 0.8156565656565656, 0.8712121212121212,
0.8510101010101010, 0.8383838383838383, 0.8232323232323232, 0.8510101010101010,
0.8409090909090909, 0.8308080808080808, 0.8535353535353535, 0.7904040404040404,
0.8358585858585859, 0.8409090909090909, 0.8308080808080808, 0.8459595959595959,
0.8106060606060606, 0.8333333333333334, 0.8080808080808081, 0.8484848484848485,
0.8257575757575758, 0.8181818181818182, 0.8282828282828283, 0.7954545454545454,
0.8383838383838383, 0.8207070707070707, 0.8207070707070707, 0.8383838383838383,
0.8181818181818182, 0.8409090909090909, 0.8156565656565656, 0.8535353535353535]
```

CHAPTER II.C. SIGNAL ANALYSIS: Band Power over Time

Author: Jake Pensa

Introduction:

Electroencephalography (EEG) or electrocorticography (ECoG) are vital tools for neuroscience that allow physicians and researchers to gain insight on how the human brain functions. As a result EEG and ECoG systems are used very frequently to both study and diagnose various afflictions affecting the brain. However, due to EEG and ECoG systems utilizing a large number of electrodes sampling at a very high frequency, it can be difficult to quickly discern useful information from the data upon first glance. Combining this with the popularity of EEG and ECoG systems in the medical field results in an excess of data that can be difficult to extract useful information from.

A popular characteristic of EEG and ECoG data to analyze is the relative power of various frequency bands over a period of time for each electrode. This allows researchers to see the presence of various frequency bands of interest such as the alpha or beta bands which are believed to correlate to different forms of brain function. In an attempt to allow for a rapid acquisition of band power calculations, an executable function was generated to take experimental data and quickly output the relative band power of each channel over a desired frequency band for a desired period of time. The goal of this function is allow for easy visualization and presentation of experimental

results when used in conjunction with the rest of the cortical heat map system by automating the necessary signal processing to generate band power.

Function:

The function called `Power_Analysis_2.exe` was created originally as a MATLAB script and turned into a standalone executable function. The function accepts experimental data in the form of either a `BCI2000.dat` or `EDF` file. The user also provides the desired time period, number of intervals, and frequency band for the analysis. The number of intervals is the number of sequential analysis that should be performed and included in the final output. For example, if the time period is set for the first 10 seconds of the data and the number of intervals is set to two, the first interval will be a power analysis on the first 10 seconds of the data and the second interval will be a power analysis on the second 10 seconds of the data.

The function starts by importing the data and extracting the sampling frequency. The function parses the provided data using two different third party MATLAB functions. The first is `load_bcidat.m` and the corresponding `.mex` file, which allows `.dat` type files to be imported into MATLAB, which includes both the data and parameters such as the sampling frequency.⁷ The second function is `edfreadUntilDone.m` which serves a similar function to `load_bcidat.m` except for data that uses `EDF` type files.⁸ Once the data is successfully imported, the size of the data matrix is calculated and it is determined if the desired number of intervals will fit within the data matrix. If not, the number of intervals is decreased to fit within the data range. The data is then confined to the desired time period, and a discrete Fourier Transform is performed using the

MATLAB fft function. The transform is then normalized by dividing the data by the sample size. The magnitude of the transform is then found by taking the absolute value of the transform and multiplying it by the square root of 2. The size of the magnitude data matrix is confined to the size of the folding frequency (the first half of the data points). From there the magnitude of the transform is squared to give the power of each frequency.⁹ Based on the number of data points in the confined time period, the frequency increment size to the folding frequency is found and an array of the corresponding frequencies is generated. From here a trapezoidal approximation is used to integrate the power matrix with respect to the frequency matrix over the desired frequency range. This results in a matrix with one data point for each channel. This data is then normalized from zero to 1 by subtracting the minimum value of the matrix from all points of the matrix and then dividing each value of the matrix by the maximum value. This data is then stored and the process is repeated for the number of additional intervals.

Upon completion of the analysis for all of the intervals a data matrix has been generated with one score per channel per interval. This data matrix is then exported as a CSV file to be read into a database for use with the brain heat map.

Inputs:

In order to use the function, the user needs to provide 6 inputs. The first input is the data file that will be read by the function. This includes the file path, file name, and file extension combined together. For example: "C:\Users\Main\Documents\Data\ECOG_1.dat". This allows the function to find the

required data file, know what file type it is using to determine how to parse the data, and provide a location to save the CSV upon completion of the data analysis.

The second and third inputs the user needs to provide are the start time and stop time in seconds for the analysis. For example if the user wants to analyze the first thirty seconds of data the user would insert “0” and “30” for the second and third inputs. The function then converts these time points into data positions within the data matrix using the sampling frequency extracted from the file, and confines the data matrix to this time period.

The fourth parameter the user needs to provide is the number of sequential intervals desired. This is the number of analysis that will be performed on the data. For example if the user wants to run three consecutive periods of analysis, the user will insert “3”. The function will check to make sure that the three intervals will fit within the full data matrix and if not will decrease the number of intervals to fit within the data. From there the function will run the analysis on the first time period, store the data, move the time range to the second time period, run the analysis, store the data, and so on until all three intervals have been run. All of this data will be exported as a single CSV with each analysis in a separate column.

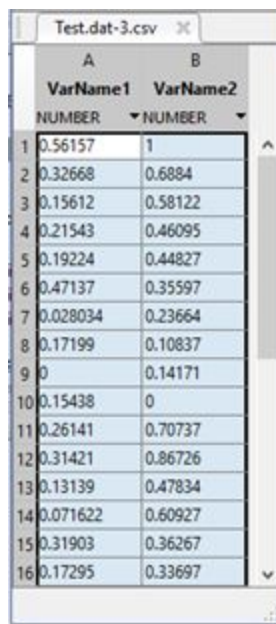
The fifth and sixth parameters are the minimum and maximum frequencies of the desired frequency band. For example if the user wants to evaluate the frequency band of 14 to 30 Hertz, the user would input “14” and “30”. The function would then find the closest match within the array of frequencies based on the frequency

increment size. These positions within the array would be used as the limits for the trapezoidal approximation.

An example function call of the executable in MATLAB would look like the following: `system('Power_Analysis_2.exe C:\Users\Main\Documents\Data\ECOG_1.dat 0 30 3 14 30')`. This would find the relative power for each channel in the file `ECOG_1.dat` for the frequency band of 14 to 30 Hz for the first 30 seconds, the second 30 seconds, and the third 30 seconds.

Outputs:

The Power analysis function outputs a CSV containing the relative power for each channel within each interval. The CSV is saved in the same location the data file is stored with the same file name and extension, but with a “-3.csv” appended on the end. Each row represents a channel and each column represents a different interval. An example of the output data can be seen below. The plots of the power vs frequency can be seen below as well. In terms of the overall visualization project, the executable function is called via a python script within the wrapper code. The CSV is generated and imported into the database to be used for heat map generation.



	A	B
	VarName1	VarName2
	NUMBER	NUMBER
1	0.56157	1
2	0.32668	0.6884
3	0.15612	0.58122
4	0.21543	0.46095
5	0.19224	0.44827
6	0.47137	0.35597
7	0.028034	0.23664
8	0.17199	0.10837
9	0	0.14171
10	0.15438	0
11	0.26141	0.70737
12	0.31421	0.86726
13	0.13139	0.47834
14	0.071622	0.60927
15	0.31903	0.36267
16	0.17295	0.33697

Figure 1: Sample CSV output from the Power_Analysis_2.exe function. The rows represent channels and the columns represent intervals. The new filename with “-3.csv” appended can also be seen at the top.

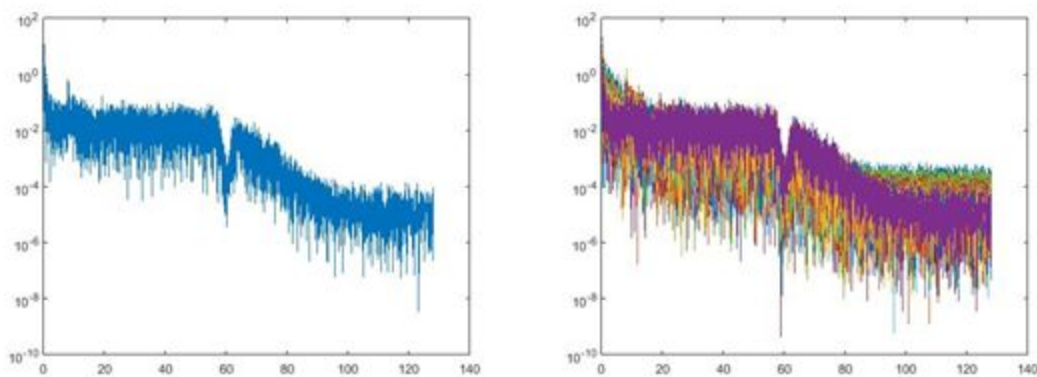


Figure 2: Power vs frequency [Hz] plots for one channel (left) and all 32 channels (right) of an EEG. The y-axis is on a logarithmic scale for easier visualization.

Install:

To install the function the executable `Power_Analysis_2.exe` needs to be downloaded along with MATLAB. The function should also be able to run using only MATLAB runtime library ¹⁰ and the `load_bcidat` mex file. If it is preferable to run the MATLAB script the function `Power_Analysis.m` file needs to be downloaded along with `load_bcidat.m`, the corresponding mex file for the computer (such as `load_bcidat.mexw64`), and `edfreadUntilDone.m`. These dependencies apart from the mex file should not be required for the executable. The python script `Jake.py` is used to call the `Power_Analysis_2.exe` function in the wrapper script, so it will need to be downloaded as well.

Potential Issues:

If the folder arrangement is changed from how it is organized in on github, the python script `Jake.py` will need to be changed to include the location of the `Power_Analysis_2.exe` function, otherwise the script will not call the function. Other issues include: if a large data set is attempted to be analyzed there may be a memory issue. This issue can be fixed by decreasing the size of the time interval so it does not exceed memory limitations. Furthermore the function `edfreadUntilDone` can take a long time to complete due to the large data set it is attempting to import. This process can be sped up by decreasing this value within the script on the line that sets the number of records "`hdr.records = ...`" under the RECORD DATA REQUESTED section.

CHAPTER II.D. SIGNAL ANALYSIS: Phase Amplitude Coupling (PAC)

Author: Mohammadali Alidoost

This chapter particularly aims to show how the phase amplitude coupling can be calculated as a part of the classification of brain signals.

Introduction

There are oscillatory activities in multiple frequency bands in the brain from micro-scale to macro-scale. Studies have shown that brain functions are achieved with simultaneous oscillations in different frequency bands; they can also interact with each other and the rhythms in each of them are linked to perception and cognition.¹¹ It is revealed that not only examining brain activity in each single frequency band, but also the relation and interaction between oscillations in different bands, can be informative in understanding brain function.¹² This has been proposed as a mechanism that different brain regions may use to communicate. Thus, this concept increasingly received interest especially in the field of cognitive neuroscience.¹³

The interaction between several oscillations is known as cross-frequency coupling (CFC). Two forms of CFC in brain rhythms are: phase-amplitude coupling (PAC), and phase-phase coupling (PPC). PAC between low and high-frequency components of signals has received great interest in neuroscience. In this case of coupling, the phase of the lower frequency oscillation drives the power of the coupled higher frequency

oscillation that results in synchronization of the amplitude envelope of faster rhythms with the phase of slower rhythms.¹⁴

Method

There are several methods for measuring PAC values. Each of those methods has certain limitations and also advantages over the others. One of these methods is called the Mean Vector Length modulation index (MVL MI) described in [3]. A time series defined in the complex plane could be used to extract a phase-amplitude coupling measure. Therefore, we will have the envelope of fast oscillation and the phase of slow oscillation.¹⁵

Therefore, to calculate PAC based on MVL MI method, a code has been written using MATLAB programming language. Signal Processing Toolbox of MATLAB along with some functions from Brainstorm software¹⁶, which is an application dedicated to the analysis of brain recordings, have been used.

To calculate PAC, at first, raw data is filtered in the frequency bands of interest. Second, two pieces of information are needed; the instantaneous phase and amplitude of signals. Both of them can be extracted by converting the signal into an analytic signal. Applying the Fourier transform or the Hilbert function to the filtered signal, we have a complex-valued analytic signal.¹⁷ All these steps can essentially be implemented in MATLAB; the instantaneous phase can be derived with the “angle function”, while the amplitude envelope is derived using the “abs function”. With these signals in hand, we can call the main function that calculates PAC values.

Our raw data (neural signals) have been acquired from electroencephalography (EEG: is a noninvasive test that records electrical patterns in the brain) and Electrocorticography (ECOG: consists of the intraoperative recording of electrical activity directly from the exposed cerebral cortex) methods^{18, 19}; Furthermore, there is another file in the European Data Format (EDF) to process. Those raw data can be transferred to MATLAB in a matrix format with nChannels and nTime size. Finally, the maximum and average values of PAC for each channel will be calculated indicating the level of coupling between each pair of low and high frequencies.

Validation

In the beginning, for validation purpose, a dataset of synthesized data with the preferred parameters was generated. Afterward, using the available tool in Brainstorm, phase-amplitude coupling was estimated. The sampling frequency of the simulated signal was also considered 1000 Hz with a duration of 6 seconds. It should be noted that frequencies ranges for slow oscillation (frequency for phase) and high oscillation (frequency for amplitude) were considered as 2-14 Hz and 40-150 Hz respectively. As a result, the maximum PAC level was obtained 0.28 by Brainstorm.

Using the same parameters, the same signal was generated in MATLAB. The result for MaxPAC value was 0.27996 showing the accuracy of the procedure for the current code.

Documentation

This section can be used as a guide to run the code properly. This Code, written in MATLAB programming language, seeks to show how maximum and average PAC values (MaxPAC and MeanPAC) can be calculated for each channel as one of the signal processing methods. As it was mentioned before, it accepts BCI2000 data files, acquired by EEG and ECOG methods, as well as data with EDF files.

It should be noted that data obtained by ECOG method and EDF files are too large. Thus, depending on the computer configuration, a few signals and a part of their length can be analyzed to find the MaxPAC and MeanPAC values; otherwise, the computer may run out of the memory.

The necessary functions to run the code properly are as follows.

- Main Function:

pac.m

- Required Functions:

load_bcidat.m

load_bcidat.mexw64

edfread.m

bst_pac.m

bst_get.m

bst_chirplet.m

bst_bsxfun.m

Bst_bandpass_fft.m

The current code also asks for 9 different inputs to be able to give results in different desired situations. The details have been explained below.

- `file`: It includes the path name, file name, and extension for the specified file.
- `low_freqs`: It is the candidate frequency band (range) of slow oscillation, e.g. [1 40] Hz.
- `high_freqs`: It is the candidate frequency band (range) of fast oscillation, e.g. [40 200] Hz.
- `num_sig`: It checks whether you have not chosen the wrong signal number (not more than all the available signals).

Note: It should be bigger than the "last_sig"!

Note: Use 0 if you want to calculate all the available signals; in this case, the values of "first_sig" and "last_sig" are not important!

- `first_sig`: It is the number of the first signal to calculate PAC.
- `last_sig`: It is the number of the last signal to calculate PAC.
- `t_total`: It is the total time period (minute) that you want to calculate PAC for each signal.

Note: Use 0 if you want to consider the whole length of the chosen signals!

- `t_window`: It is the time period for the moving window (minute) that you want to calculate PAC for each signal.
- `t_step`: It is the time step (minute) between the moving windows.

As it has been explained before, the outputs of the program are the maximum and the average values of PAC for each channel. Thus, you may have different (more than one)

MaxPAC and MeanPAC values for each of the chosen signals depending on the time window and the time step values you choose. In addition, the results can be accessed through a file with CSV format.

- MaxPAC: It is the maximum value of PAC measures for each channel; (or/and)
- MeanPAC: It is the average value of PAC measures for each channel.

Therefore, the MATLAB Function would be as follows.

- `[MaxPAC, MeanPAC] = pac(file, low_freqs, high_freqs, num_sig, first_sig, last_sig, t_total, t_window, t_step)`

This Function can also be run in python using the commands below.

- ```
import matlab.engine

eng = matlab.engine.start_matlab()

pac = eng.pac (r'file', matlab.double(low_freqs), matlab.double(high_freqs),
float(num_sig), float(first_sig), float(last_sig), float(t_total), float(t_window),
float(t_step), nargout=2)

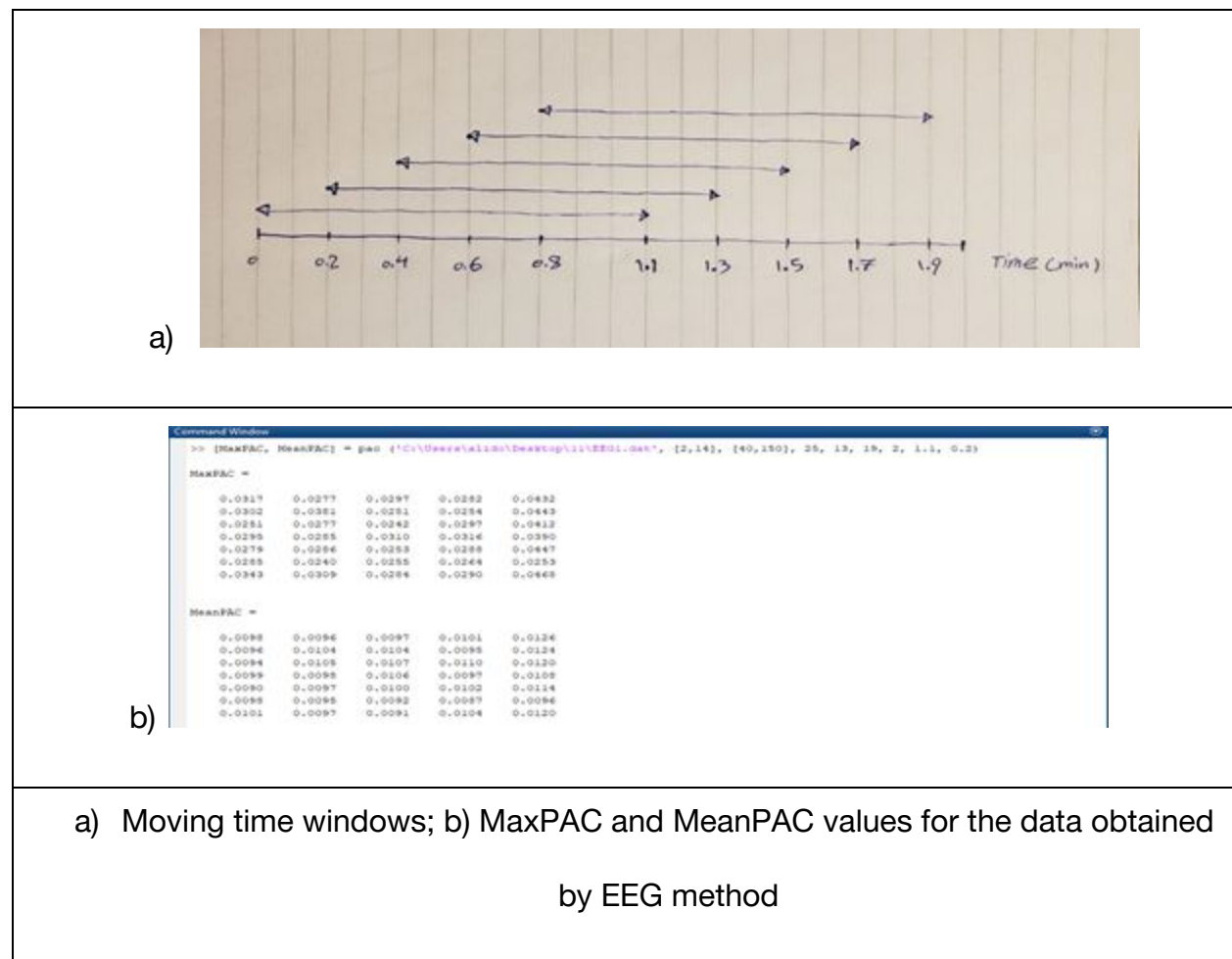
print(pac)
```

## **Results**

In this section, two kinds of simulation results, including pac values in MATLAB command and a standard coupling map will be shown.

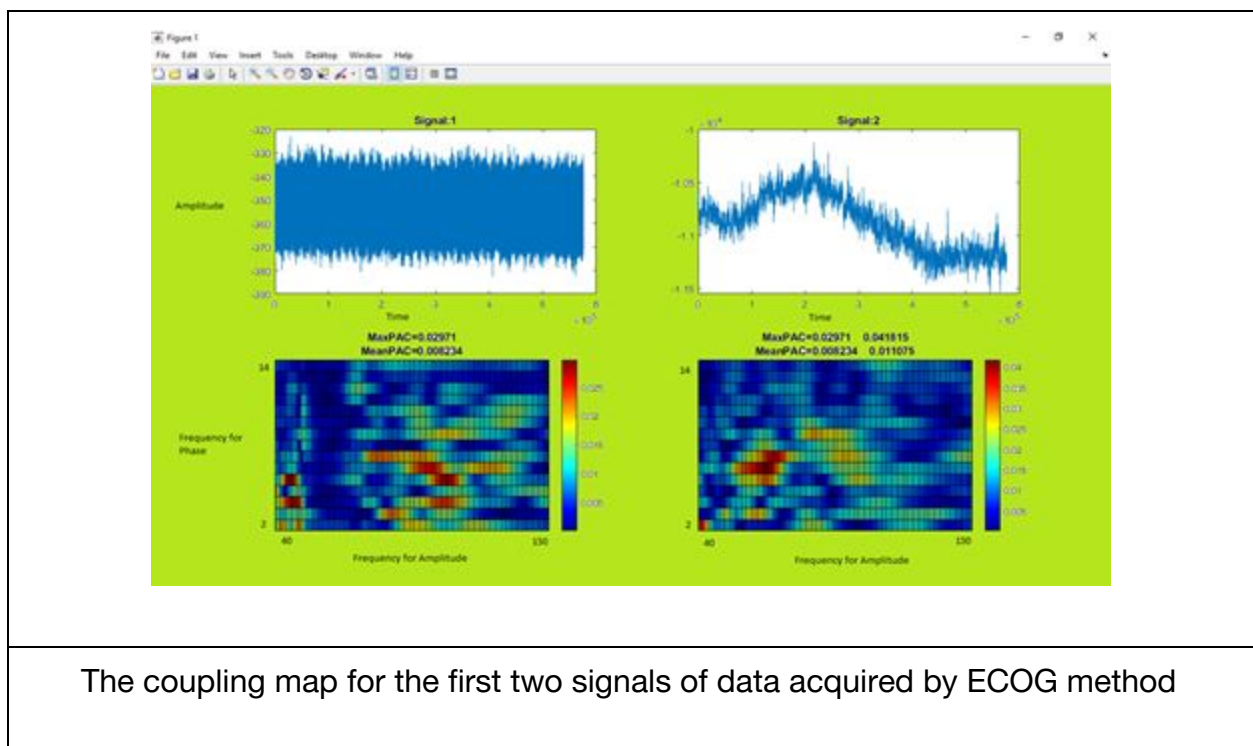
## A. Simulation result in MATLAB command

The result of using the data acquired by EEG method in the MATLAB command window is represented. It should be noted that one of the capabilities of this code is that PAC can also be calculated in specific moving time windows over the chosen time which is evident in the figure below.



## B. Coupling Map

The following coupling map, also known as comodulogram, is the result of the first 2 signals acquired by ECOG method and shows the cross frequency coupling in the frequency domain. The maximum and average PAC values have also been calculated. Frequency ranges for slow and fast oscillations are 2-14 Hz and 40-150 Hz respectively.



## Conclusion

In this chapter, a program in MATLAB was written to calculate phase-amplitude coupling (PAC). Data files obtained by EEG and ECOG methods were used as different

inputs. As a result, maximum and average PAC values between low and high-frequency components of chosen signals were calculated showing how much the amplitude of a fast oscillation is related to the phase of a slower oscillation. Finally, the pac values for each channel, called channel scores, along with the position of each channel are used to generate a colormap.

## **CHAPTER II.E. SIGNAL ANALYSIS: Audio Analysis**

**Author:** Ge Fang

This chapter particularly aims to show how the delay in sound production can be calculated as a part of the classification of brain signals.

**Introduction and background:** This signal processing method mainly focuses on finding the relevance between neurological signals and speech production. Neurological signals, such as electrocorticography (ECoG) signals which are collected by placing electrodes directly on the intracranial surface, provides decent sensitivities, are widely used for neurological studies.

The regions on cortex related to speech can be simply divided into two groups: (1)Regions related to speech production; (2)Regions related to speech feedback processing. Speech production, anatomically, is related to the contraction and relaxation of muscle within vocal cord and oral cavity, which is controlled by the motor cortex, a region of the cerebral cortex <sup>20</sup>. When a sound is planned to produce, different regions within the motor cortex will be activated, which pass through the central nervous system and peripheral nervous system, to change the state of muscles in different part of the oral cavity and vocal cord involving in this specific sound, or phoneme. Speech feedback processing, Speech feedback processing occurs after speech production, when the states of muscles, as well as the sound information, will be fed back into the cortex, activate or suppress another group of neurons on the cortex.

Finding the correlation between these two systems not only helps the understanding of the activities of motor cortex but also, it may be a useful tool to investigate the diseases related to speaking.

This tool focus on calculating the possible regions related to vocal cord, contraction, which leads to the production of a voiced sound (a sound can be heard/record), as well as classifying the regions into either group above.

### **Input, Output of This Method and Relationship to Overall Project Workflow:**

Input: the method receives the path of ECoG file that was recorded when a person was speaking, as well as the simultaneous audio file (which in some cases is already included in the ECoG file). Also, the users can also select the time period, channel numbers, the signal compressing rate by inputting the numbers under the instruction of the interface, here, could be python IDLE, or command window.

Output: the method will generate a score table of the relative time difference between the neurological signal collected by each channel, and the audio record.

Relationship to overall project workflow: this method feeds scores to the visualization methods (here, the heat map method programmed by Zhaoqiang)

### **Required Platforms and Packages**

#### **Platforms:**

**Matlab** R2017a or later.

**Python** 2.7, 3.4, 3.5 if installing Matlab R2017a

**Python** 2.7, 3.4, 3.5, 3.6 if installing versions after Matlab R2017a

**Package and Command Based on Win10**

For Matlab: load\_bcidat.m

For python:

Csv - 'pip install csv'

Scipy - 'pip install scipy'

Matlab - 'pip install matlab'

Os - 'pip install os'

Math - 'pip install math'

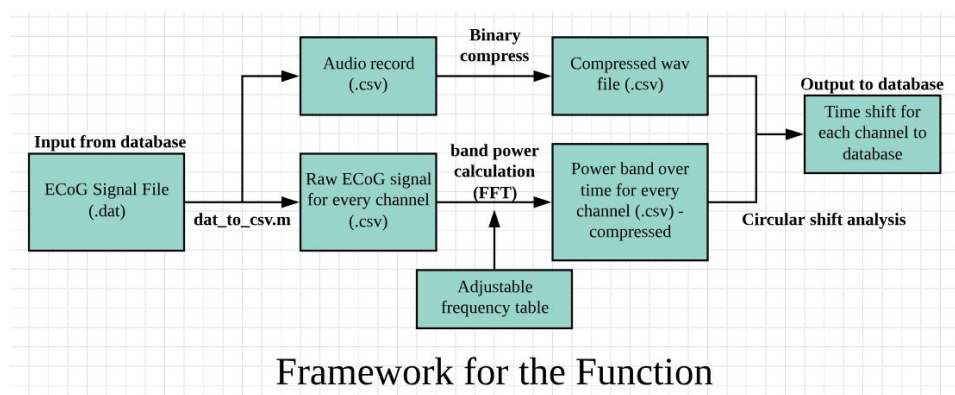
Sys - 'pip install sys'

## Method & Framework

The principle of this method: Applying multiple time shifts on the time-related powerband of ECoG signals and calculate the correlation between audio record signal and the shifted ECoG signals, to find the highest correlation and the time-shift applied to it.

The program is based on Python, but it will automatically run Matlab to convert necessary files.

The framework of this method is shown below:



Framework for the Function



1. Pull ECoG file path from the database, and then convert the ECoG file(including a channel carrying audio record) into CSV format, which is readable to python.
2. Convert the audio record into binary form and compress it based on the input parameters that the user provides.
3. Apply band power calculation to the channels that the user willing to analyze, while saving band power table in CSV format locally.
4. Applying different time shifts to band power data, and calculating the correlation between audio record and the shifted powerband data.
5. Searching for the timeshift giving the best correlation result, and upload it to the database.

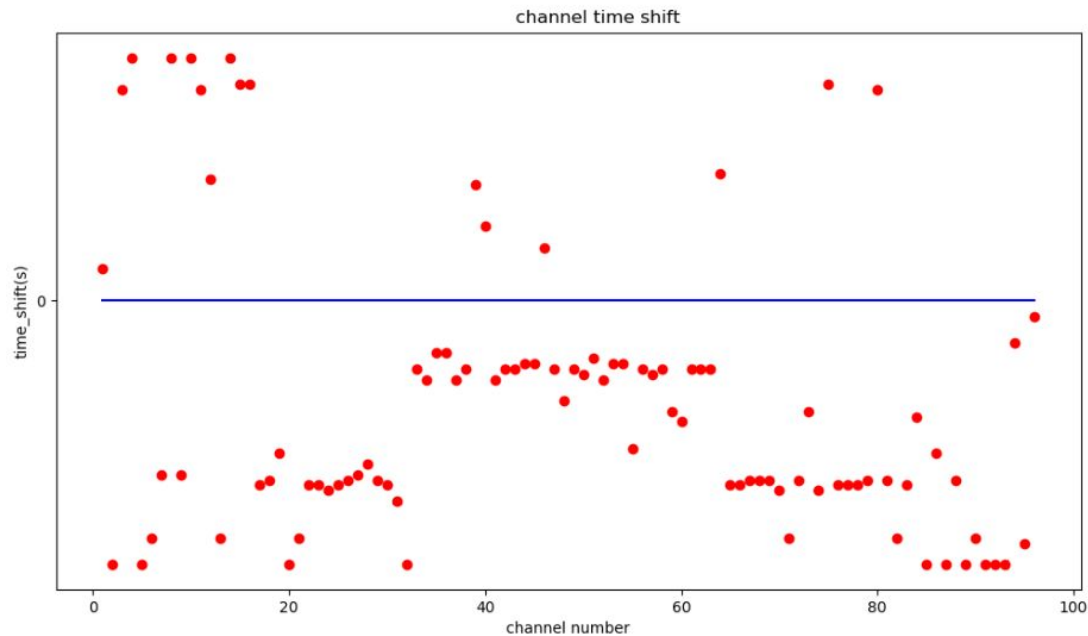
**Example of output**

| A              | B               | C          | D        |
|----------------|-----------------|------------|----------|
| Channel Number | Time Shift(sec) | Correation | p-value  |
| 1              | -0.14           | 0.022565   | 9.57E-13 |
| 2              | -0.6            | 0.21917    | 0        |
| 3              | -0.58           | 0.200534   | 0        |
| 4              | -0.52           | 0.225724   | 0        |
| 5              | -0.52           | 0.249915   | 0        |
|                |                 |            |          |

The first two column (Channel number and time shift) will be uploaded to the database, while all four column will be saved locally, providing useful information for further investigation.

## Further Investigation of the Output Information

Here is an attempt I made to interpret the result.



As we can see, the channels are grouped together based on their location (if we assume that the channel are numbered based on their location), so it's an indication that the regions of speech production and speech processing are located separately, or at least, relevant in their location.

## **CHAPTER III. ELECTRODE LOCALIZATION**

**Author:** Joseph Tsung

**INTRODUCTION AND BACKGROUND.** In this platform, there are currently two types of available inputs: EEG and ECoG data. Unlike EEG electrodes which have static mapping using the standard 10/20 system, ECoG electrodes require spatial alignment between preoperative T1-weighted MRI image and the postoperative CT image to locate the electrodes, also known as registration. The purpose of this electrode localization module is to demonstrate the implementation of rigid-body registration including the following components:

- *CT-MR registration:* Align and overlay the CT image with the MRI image in order to determine anatomically-precise locations for the electrodes.
- *Electrode detection:* Determine the locations of electrodes to be used in the visualization elements of this platform.
- *Talairach-MR registration:* Translate the surgically-defined electrode positions in Talairach coordinates to MNI-MR coordinate space to allow for visualization.

**Relationship to Overall Project Workflow:** The CT-MR registration and electrode detection components are sequential components that aid in ECoG electrode localization. Similarly, the Talairach-MR registration represents a parallel pathway to accommodate variability in available ECoG data in instances where patient CT and MRI images are not available.

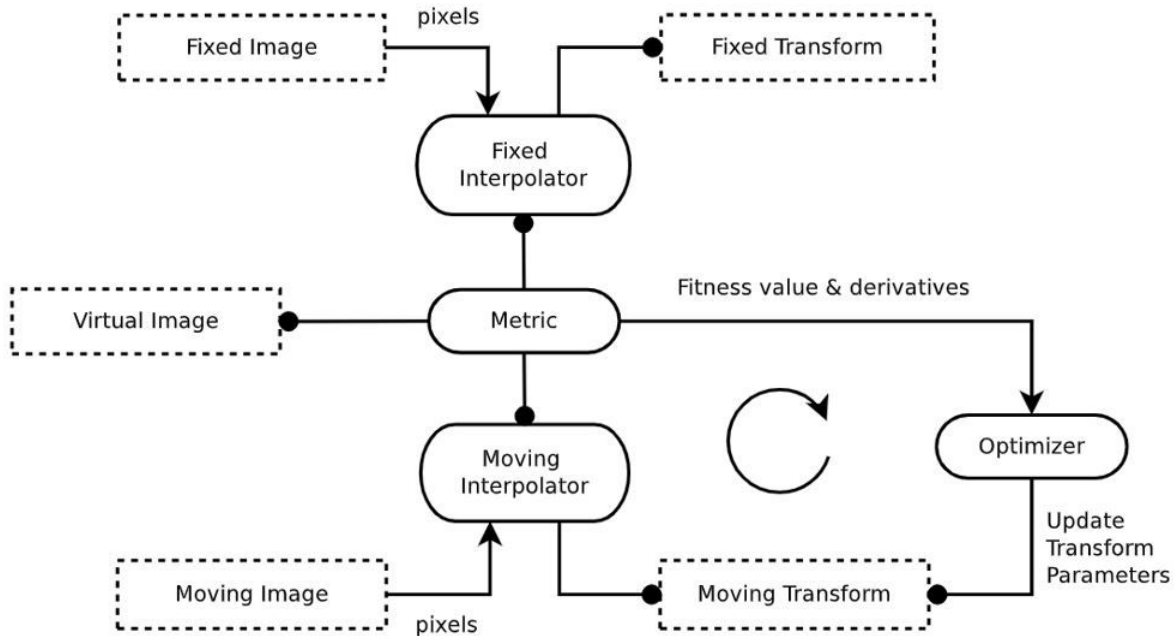
The inputs of the CT-MR registration are the pre-operative CT image and post-operative MR image, both in the standard NIfTI format, producing an output of the registered CT image also in the standard NIfTI format. The registered CT image is then used as the input to the electrode detection component in order to find and output the electrode coordinates. Lastly, the Talairach-MR registration component uses the provided Talairach coordinates as input to generate and output the electrode coordinates in MNI-MR.

**METHODS AND IMPLEMENTATION:** The electrode localization component of this platform will use open-source Python modules and libraries to determine electrode coordinates.

**Component goal:** Locate ECoG coordinates to allow for visualization in subsequent steps of this platform, taking into account of availability of different patient data (i.e. CT image and MR image or surgical Talairach coordinates).

**Approach:** The CT-MR registration component utilizes the established framework of the Insight Segmentation and Registration Toolkit (ITK) toolbox, interfaced for use in multiple programming languages including Python, R, and Java as SimpleITK. Specifically built with digital images in mind, SimpleITK uses object-oriented design in organizing the variety of tasks into recognizable Classes and Objects. For the purposes of registration, the following framework (**Figure 3.1**) is used <sup>21</sup>:

- *Fixed\_image*: The stationary image used during registration for the moving\_image to overlay and align to. In the context of the platform, the stationary image or the image being registered to is the patient MRI image.
- *Moving\_image*: The mobile image used during registration to overlay on top of the fixed\_image. In the context of the module, the moving image or the image that is registering is the patient CT image.
- *Similarity metric*: The comparison statistic between the moving image and the fixed image that controls the number of iterations that the moving transform will be run or optimized for. SimpleITK supports various metric types, including mean squares, demons, correlation, ANTS neighborhood correlation, and the most commonly used metric, joint histogram mutual information. In this platform, joint histogram mutual information is used.
- *Optimizer*: The comparison optimizer types supported by SimpleITK include exhaustive, Nelder-Mead downhill simplex, Powell optimizer, gradient descent, and limited memory Broyden, Fletcher, Goldfarb, Shannon-bound constrained. In this platform, gradient descent is used as the default optimizer.
- *Interpolator*: Algorithmic attempts to construct new points in each iteration of the optimization process. Linear, nearest neighbor, spline, gaussian, Hamming windowed sinc, cosine windowed sinc, Welch windowed sinc, Lanczos windowed sinc, and Blackman windowed sinc. The default setting for this module is the linear interpolator.



**Figure 3.1:** Overarching framework used in the registration component from [SimpleITK](#) (Yaniv Z et al, 2017) to construct an instance of the *ImageRegistrationMethod* class.

As the framework above illustrates, SimpleITK's object-oriented approach is powerful in allowing for modularity, customizing the registration process using a range of available options to carry out the same overall task. A wide number of possible registration frameworks can quickly be built and compared, tailoring to specific needs.

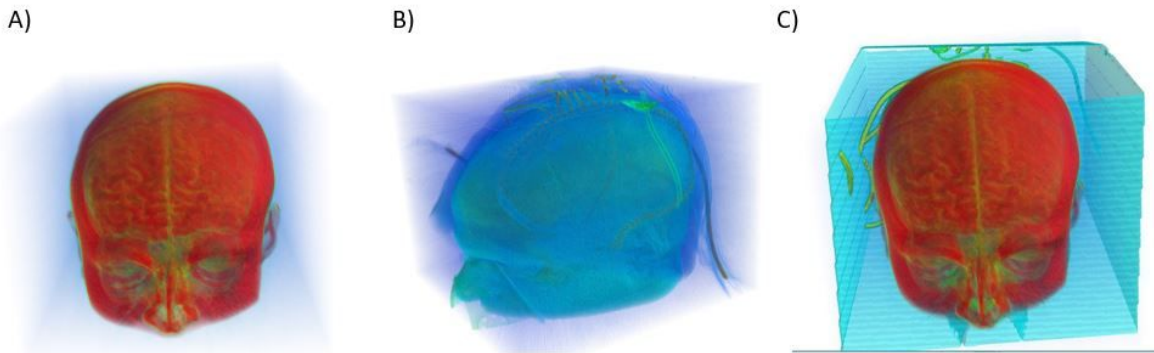
In electrode detection, the module begins with image segmentation that filters the registered CT image into only the relevant parts, namely the electrodes. Similarly, SimpleITK supports a wide variety of filters including Otsu filtering. In this specific instance, a simple binary thresholding technique is used fortunately due to the high intensity of electrodes as they appear in the CT. Having delineated the regions of

interest in the image, a Gaussian blurring or smoothing was performed to eliminate high-resolution noise or single-pixels in the image. This step is crucial in that it necessarily affects the performance of the connected component function. This is especially important since there can be presence of artifact in the post-operative CT image. In finding connected components, this module is attempting to find independent clusters of connected pixels that likely represent electrodes of interest in the image. Filtering for sizes of connected components yield the most likely candidates for the locations of electrodes. Lastly, the centroids of all the connected components is outputted.

The Talairach-MR registration component uses an open-source tool from the Yale BioImage Suite <sup>22</sup> to convert Talairach coordinates into MNI coordinates via automated web browser interactions. The resulting MNI coordinates can then be plotted onto the MNI/ICBM 152 template brain <sup>23</sup>, an averaged anatomic MRI template registered onto the MNI template.

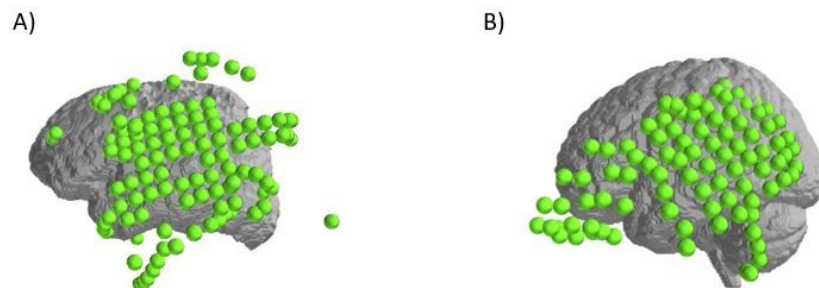
## **RESULTS:**

*CT -> MR registration:* Using volume rendering from the visualization steps, the results of the CT-> MR registration component can be seen below (**Fig. 3.2**). After performing the CT-> MR registration component, volume rendering shows a great deal of overlap between the original patient MRI image with the registered CT image.



**Figure 3.2:** Volume rendering from A) pre-operative MRI image B) post-operative CT image and C) registered CT image after running the registration component.

*Electrode detection and Talairach registration:* **Figure 3.3** shows the electrode locations layered on top of rendered patient MRI and MNI/ICBM 152 template.

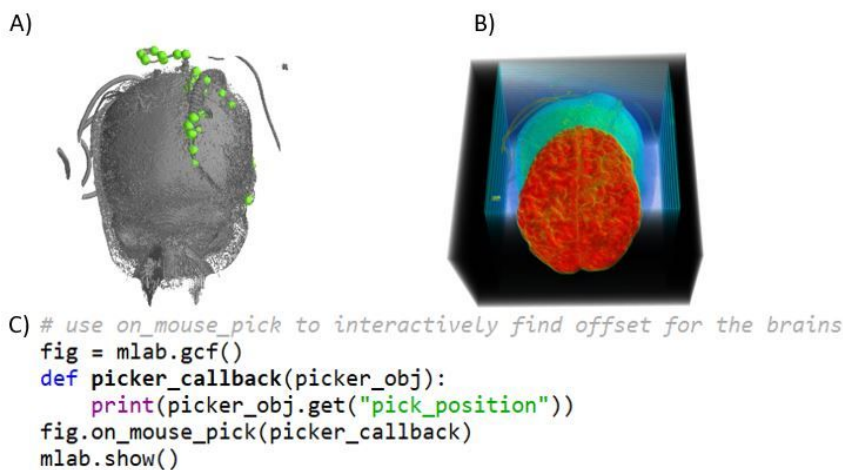


**Figure 3.3:** Volume rendering to visualize electrode locations. A) Electrode locations from electrode detection of the registered CT image plotted along with the skull-stripped MRI image B) Electrode locations obtained from Talairach registration, mapped onto the MNI/ICBM 152 MRI template.

**DISCUSSION:** The aim of the electrode localization component was to automate the detection of the ECoG electrode coordinates, given either the pre-operative CT image and post-operative or the Talairach coordinates. Overall, the component was able to successfully detect electrode locations. Additionally, using SimpleITK in Python avoided the need to download auxiliary neuroimaging software or Linux terminals in order to access such programs such as SPM or Freesurfer that are routinely used for



registration and other imaging-related tasks. Despite the sparse documentation available for SimpleITK, it was relatively simple to implement and set up the module in Python. However, there were a few problems encountered and potential solutions are suggested below.



**Figure 3.4:** Volume rendering to visualize coordinate mismatch. A) Electrode locations plotted on volume rendering of the un-skull-stripped registered CT image shows promising cortical adherence B) Mismatch between the registered CT image (blue) and volume-rendering for the skull-stripped brain used in visualization (red) C) Code snippet showing interactive functionality as a possible approach to fixing coordinate mismatch.

#### *Problems and potential solutions:*

- **Conflicting coordinate systems:** Electrode coordinates shown in **Figure 3.3** were generated after uniform manual correction. As such, some appear to be “floating” off of the surface. The displacement of the electrodes away from the brain mask used for visualization was found to be due to coordinate mismatch.

As **Figure 3.4** shows, the registered CT image (*in blue*) and the skull-stripped

brain used for visualization (*in red*) are mirror images with some offset. An additional complicating factor include the differences between image representation in array format between SimpleITK and Numpy. Consequently, future work to better electrode localization will focus on correcting the coordinate mismatch. A transformation (reflection) matrix could be applied to the current electrode coordinates and the *on\_mouse\_pick* function could be used to find the offset between the two brains to achieve better cortical-surface fitting with an additional translation.

- *Algorithmic order in electrode detection:* In the electrode detection component, binary thresholding was implemented prior to Gaussian smoothing. Theoretically, implementing Gaussian smoothing followed by binary thresholding would help to further reduce noise from non-electrodes. However, attempts at reversing the algorithmic order resulted in over-reduction of connected component pixel sizes. In addition, new threshold intensity values would have to be determined after smoothing.

## **CHAPTER IV. ELECTRODE CHANNEL LABELING**

**Author:** James Go

### **Introduction**

ECoG electrodes identified by CT scanning are given x,y,z coordinates. These electrodes are scored by various methods, but they need to have channel labels in order to match them to the scoring methods. Previous methods of labeling required each individual point to be plotted and manually labeled which is very tedious and time consuming. We developed a graphical user interface (GUI) to aid the user in labeling ECoG electrode channels.

### **Methods**

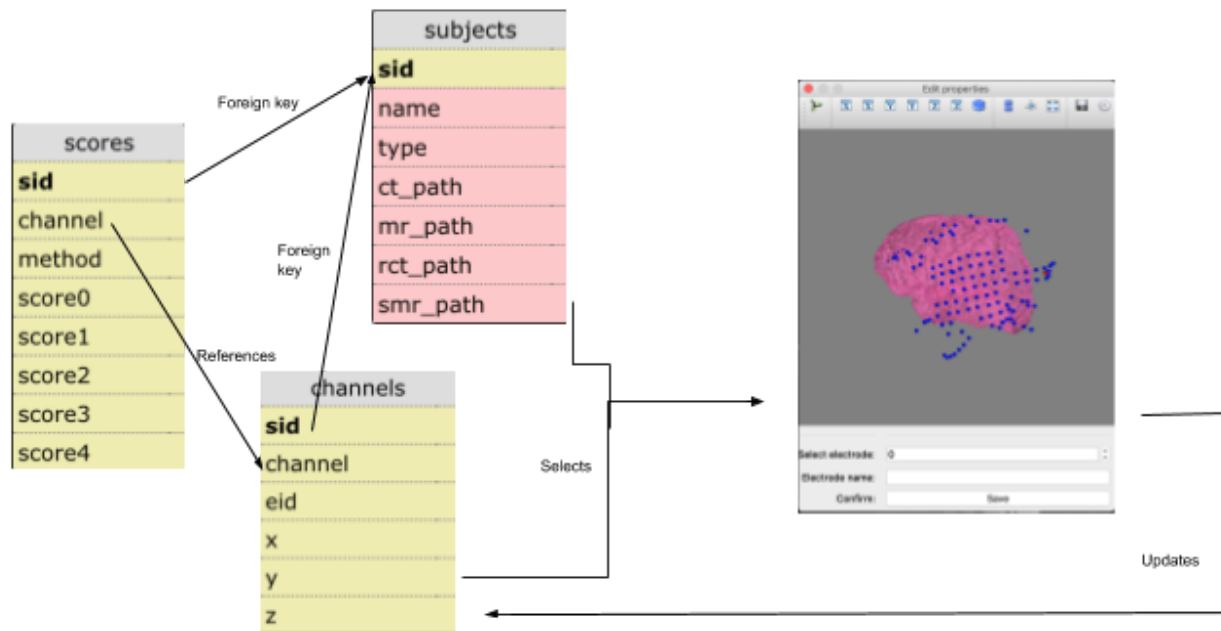
This module requires the following software and libraries:

- Python 3.6. Python 3.7 is not used because currently Mayavi cannot be installed with that version.
- Mayavi <sup>24</sup>, a library used for displaying 3D visualizations.
- Traits UI <sup>25</sup>, a GUI library that displays and interacts with Mayavi visualizations.
- NiBabel <sup>26</sup>, a library that reads nii files and loads them into Mayavi.

The GUI is called from the wrapper function and is passed a database cursor object and the subject ID. The file path to the skull stripped MR nii file is pulled from the subjects table and the coordinates for each channel are pulled from the channels table

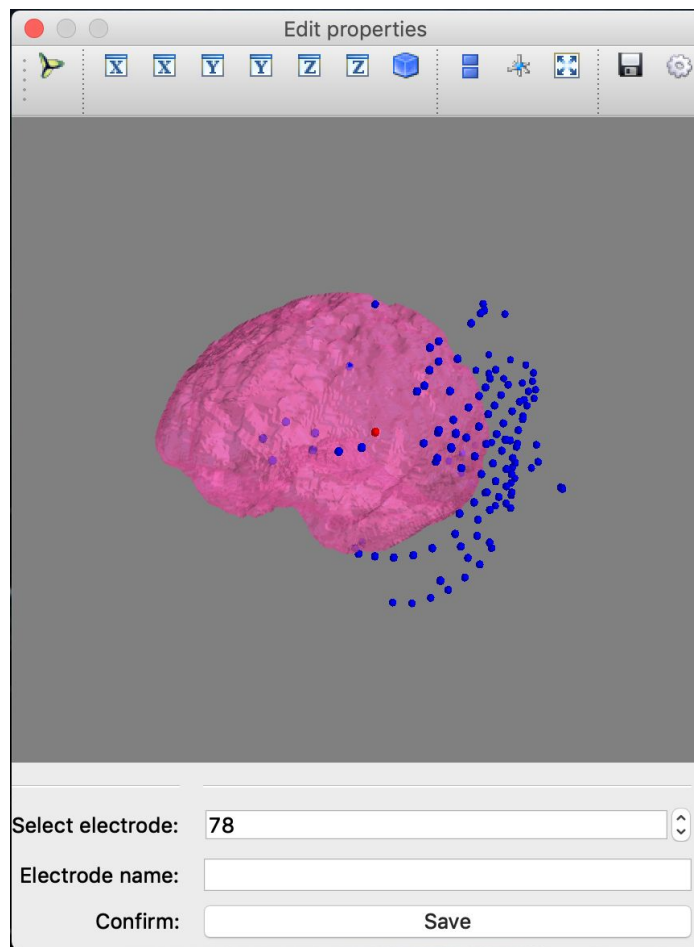
where they match the subject ID. Only rows where the eid column is NULL in the channels table are selected since the eid column is a foreign key to the eeg table and eeg electrodes do not need to be labeled. The skull stripped brain is loaded into memory and plotted by Mayavi. The plot is made 80% transparent so that electrodes with coordinates inside the brain are visible. Each electrode is plotted as a blue point, except the currently selected electrode which is red. The user can scroll and select different electrodes, input the name of the selected electrode or save the name of the selected electrode. Saving the name automatically selects the next electrode. When the user exits out of the GUI, the electrode channel names are saved back into the channels database table. A warning is then given to the user displaying how many channels could not be updated in the database.

Below is a diagram of how the database tables and GUI interact.



## Results

A simple GUI window is created with a transparent brain image and electrodes as seen below. The selected electrode is given a red color. Note that several electrodes are embedded inside the brain but are still visible. The user is able to rotate the brain as needed and capture images.



After exiting the GUI, the results channel labels are automatically saved to the database in the channel column.

| sid | channel | eid | x  | y  | z   |
|-----|---------|-----|----|----|-----|
| 1   | 13      |     | 40 | 47 | 90  |
| 1   | 1       |     | 83 | 52 | 69  |
| 1   | 2       |     | 43 | 60 | 64  |
| 1   | 3       |     | 20 | 52 | 73  |
| 1   | 4       |     | 41 | 91 | 34  |
| 1   | 5       |     | 39 | 69 | 60  |
| 1   | 15      |     | 53 | 28 | 114 |
| 1   | 16      |     | 39 | 73 | 69  |

The user is warned if all the records are not updated.

**Could not update 94 out of 128 channels**

### **Discussion**

Although this module should help the user more efficiently label electrodes, there are still some improvements that can be made. Future work may include automating the labeling of electrodes. To accomplish this, the user would select an electrode to be labeled as electrode 1. Then the script would try to figure out what the next electrode will be based on the geometry of the electrodes. Multiple iterations can be made and the user could select the iteration that is most accurate. The user can then adjust the electrodes that were incorrectly labeled.

Another improvement could be integrating this module into the heat map generation module. The current workflow in the overall program is to open the GUI for this module, label the electrodes, close this module, then open the GUI for the heatmap module. Both modules use Mayavi and TraitsUI for 3D plotting and GUI generation respectively. These two steps could be combined to provide an easier workflow for the user.

The Mayavi framework is not well documented for the functions we wanted to build in this module. For example, we wanted to allow the user to select electrodes by clicking on the image. With Mayavi, events can be registered to the image to get the

coordinates of where the user clicked. However, click events could not be registered in conjunction with the TraitsUI GUI library, which is used for providing input boxes and buttons. It wasn't clear from the documentation of a way to allow GUI elements and click events together. The Kivy <sup>27</sup> GUI framework was considered as an alternative to Mayavi. Kivy is a mature framework for building similar applications. Unfortunately, it has poor support for 3D drawing and could not be used. Future work may include looking into other GUI frameworks such as Qt to provide more flexibility in adding additional features.

## **Conclusion**

This module allows the user to label electrode channels in order to label them more efficiently. It interacts with other modules by pulling in the skull stripped MR brain file generated by the skull stripping module and the ECoG electrode channel coordinates generated by electrode localization from the database. It then provides an easy workflow for the user to label each electrode. Further improvements can be made to speed up the workflow for the user.



## **CHAPTER V. SURFACE RECONSTRUCTION, CHANNEL PLOTTING, AND COORDINATE CORRECTION**

**Author:** Yannan Lin

### **SKULL-STRIPPING & SURFACE RECONSTRUCTION**

#### **INTRODUCTION**

Skull-stripping is the first step towards surface reconstruction. BrainSuite 18a<sup>28</sup>, which is a collection of tools that can help automate MRI processing of the human brain, is used in this program to aid the skull-stripping process. Only ECoG subjects need to do skull-stripping as the end users will provide their own CT and MR image data. For EEG subjects, the end users would not have their own MR data, thus, a standard brain will be used for plotting. The skull-stripped standard MR data is pre-stored in the subjects table in the database and can be requested from the database directly when plotting.

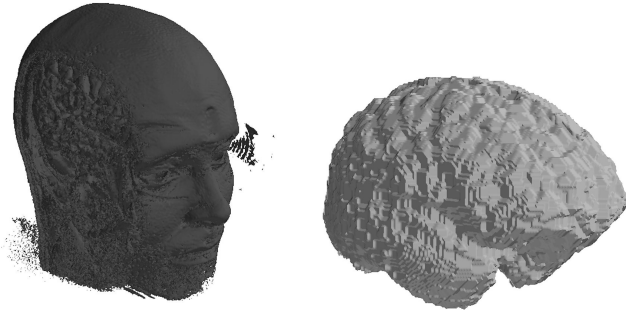
#### **METHODS**

- Language: Python 2.7

Since BrainSuite supports command line execution<sup>29</sup>, skull-stripping is done through command line surface extraction. The command line code is executed in python 2.7. The input data is the preoperative head MRI in the NIfTI format and the output data is the skull-stripped cortex mask (not volumetric cerebrum). The output files will be saved to the same path as the head MR file. There are a dozen of output files and the one ending with .cerebrum.mask.nii.gz will be stored to the subjects table in the database.

#### **RESULTS**

The following two figures help demonstrate how the head (left) and skull-stripped (right) MR data look like after 3D rendering using mayavi package <sup>24</sup>.



## **PROBLEMS AND POTENTIAL SOLUTIONS**

BrainSuite also provides tools to manually adjust the automatically generated brain mask, making the cortical surface extraction more accurate. However, command line execution does not support mask tools. One potential way to solve this problem is by building an integrated user interface within which all functionalities of BrainSuite application are embedded.

## **CHANNEL PLOTTING**

### **INTRODUCTION**

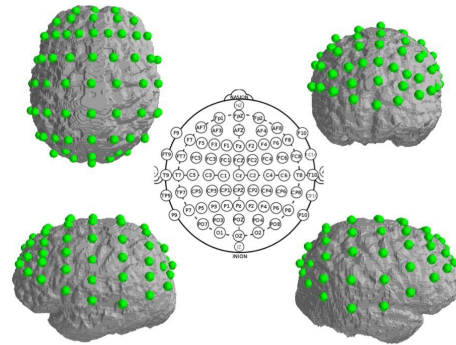
After the coordinates of the electrodes are obtained, they can be plotted on the cortical surface to reflect either the location of the electrodes implanted on the cortical surface of a subject or the projection of the scalp electrodes in the 10-20 system onto the cortical surface.

### **METHODS AND RESULTS**

- Language: Python 2.7

### a. EEG

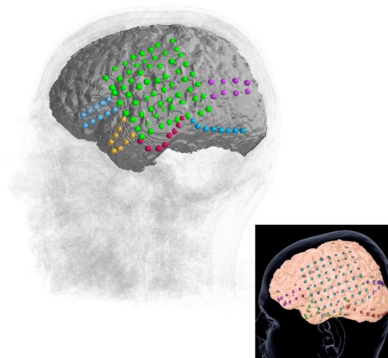
For EEG subjects, the coordinates of the scalp electrodes in the 10-20 system are located manually through a mouse picker function from mayavi package in Python. The coordinates of 63 scalp electrodes are found and prestored in the eeg table in the database. In this program,



instead of plotting the EEG electrodes on the scalp, they are projected onto the cortical surface to demonstrate the regional brain activity. When the end user has an EEG subject, the program will request the prestored standard brain mask data from the subjects table and EEG coordinates data from the eeg table for plotting. The figure below shows a the standard brain mask with 10-20 EEG electrodes on it.

### b. ECoG

The method to get the coordinates for ECoG subjects is different from the method used for EEG subjects. Since the end user will provide their own preoperative MRI and postoperative CT (with electrodes implanted on the cortical surface) data, the coordinates of the electrodes are obtained from the CT→MR/Talairach→MNI→MR



registration step. The coordinates from the registration step is uploaded to the channels table in the database. When plotting, the program requests the skull-stripped MR data along with the electrode coordinates data and output the cortical surface with

electrodes on it. The figure below demonstrates the ECoG electrodes on the cortical surface.

## **PROBLEMS AND POTENTIAL SOLUTIONS**

Because the EEG electrode coordinates are found manually, some of them may seem buried deeper or shallower in the mask than others depending on the region-specific degree of convolution. This could be fixed by manual adjustment through eyeballing the placement of all of the electrodes. A more complex approach is to develop some algorithms to detect the depth of each electrode plotted on the cortical surface and automatically fix the electrodes that sit very deep or shallow on the surface.

## **ELECTRODE POSITION CORRECTION**

### **INTRODUCTION**

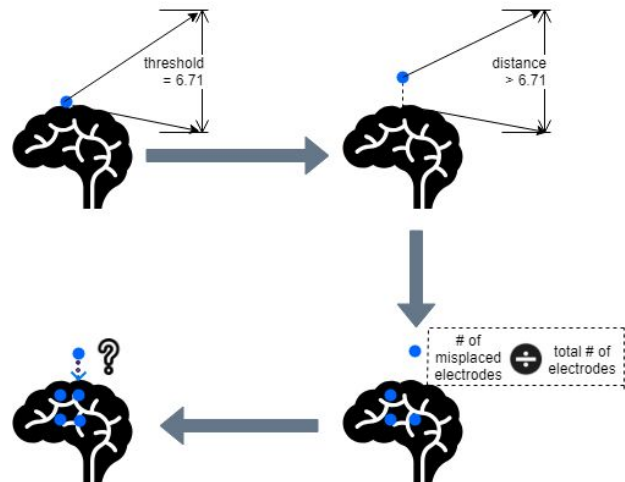
The coordinates of the electrodes are from the CT→MR registration for ECoG subjects. The two studies used for the registration are the preoperative MR study and the postoperative CT study. On the postoperative CT, the electrodes appear bright and can be detected through filtering the signals. However, several factors can affect the registration from one space to another. The electrodes would seem off of the cortical surface if there are inconsistencies between the two studies. Another source of electrode coordinates is from the Talairach→MNI→MR registration step. It basically has the same issue as the CT→MR step. Therefore, an algorithm is in place to help fix the electrodes that are not sitting on the cortical surface. This correction function would only be executed when less than 5% of all the electrodes are misplaced. If a

large proportion of the electrodes are off, they would be left as they are because it makes no sense to pull them all back to the surface for too many electrodes. This function works to fix the minor misplacement issue of the electrodes occurred during registration process, if any.

## **METHODS**

- Language: Python 2.7

- Find the misplaced electrodes: The manually found scalp coordinates for EEG subjects is used to determine a threshold value. The coordinates of the nearest point on the brain mask to every electrode is



calculated. The distances between the center of each EEG electrode (voxel) and the nearest point is calculated for all EEG electrodes and a threshold value of 6.71 is determined as the largest distance is 6.708. The definition of a misplaced electrode is the distance between its center and the nearest point on brain surface is greater than 6.71.

- Calculate the proportion of the misplaced electrode: After getting all the distances from the center of each electrode to the nearest point, the proportion of misplaced electrode is calculated.
- Determine if correction will be performed: If the proportion of the misplaced electrodes is greater than 5%, it will not change the coordinates of any

electrodes. However, if the proportion is less than 5%, the coordinates of the nearest point on brain surface will be used as the coordinates in plotting.

## **RESULTS**

Results from one CT→MR registration and one Talairach→MNI→MR registration were used to test the performance of this function. The electrode coordinates from those two testing cases showed that more than 90% of the electrodes were not on the cortical surface. Therefore, the function did not update the coordinates as the <5% off requirement was not met.

## **PROBLEMS AND POTENTIAL SOLUTIONS**

There is always an argument about how to find the right coordinates of a floating electrode on the cortical surface after registration. Others have tried different algorithms. One suggested method is to draw a line between the off electrode and center of the brain mask, and use the point of intersection between the line and cortical surface as the suggested coordinates. Another concern of doing coordinate correction is that if an electrode is very far away from the cortical surface, is it really a misplaced electrode or something else such as noise? The aforementioned issues depict the difficulty to verify these approximation methods for an automated program like this one. Because the program is not able to double-check if an electrode is truly off by looking into the postoperative CT study. One potential solution is to implement a look back function, which helps compare the electrodes on the cortical surface and CT. The look back function will be able to determine if the floating object is an electrode, and if it is, give the right coordinates to place it on brain surface.

## **CHAPTER VI. HEATMAP GENERATION AND FINAL VISUALIZATION**

**Author:** Zhaoqiang Wang

- Input: scores, coordinates, brainmask
- Language: Python 3.6
- Python Libraries: Mayavi, Traits and others
- Output: Plot window

### **Introduction**

Given skull stripped brain images and dynamic signals recorded by ECoG (or EEG), the final step is to visualize it interactively. We want to deliver a high-quality rendering of cortex texture with heatmap on the surface showing channel's activity. By consulting a colormap, user will observe regions where signal is intense or mild (the specific interpretation will depend on the analysis methods such as Phase Amplitude Coupling, Band Power etc.). And for the interaction, user shall be able to select different time windows to see the fluctuation of signal, as well as manipulating the view freely.

There exist many mature platforms for 3D data visualization. With these packages, a volumetric rendering could be as simple as calling a function. For example, VisPy is a high-performance interactive 2D/3D data visualization library leveraging the computational power of modern Graphics Processing Units (GPUs) through the OpenGL library to display very large datasets. It has straightforward functions to handle colors, geometry, data IO and filters. There's even a package named Visbrain that's built on VisPy exclusively for brain visualization.

However, existing package often restricts the form of demonstration such as a fixed rendering mode and sometimes it's incompatible with our preferred UI tools. Finally, after practice and experience, we chose a combination of Python, Traits and Mayavi to design this module.

## **Methods**

The proposed schema is illustrated in Fig.6-1 which constitutes three parts: Database Interface, Mayavi Rendering and Traits UI. As a final presentation of all the results from the other modules, we pull and sort all the brain images, scores and channel coordinates from the database and generate a plot window.

For the core function, heatmap generation, we chose a gaussian function to simulate the diffusion as the signal spreads across the cortex. We first define a electrodes space which assigns specific value to each channels' positions. These values are normalized to the maximum to accommodate different signal analysis methods. And we apply a gaussian kernel to convolve the entire space. Here, in order to filter out a cortex surface shape of heatmap, we also need to use the corresponding brain mask to define the region of interest. Next, we set a colormap such as RGB colormap to assign the largest value red while the lowest value blue. To store the heatmap matrix (3D + time) into RAM for quick query, we have to do a little extra data compression. In specific, we tick out those zeros in the matrix and rearrange those values left with their index. Because that the region of interest is only a surface so zeros make up a large portion. This simple compression will decrease the size of matrix dramatically.



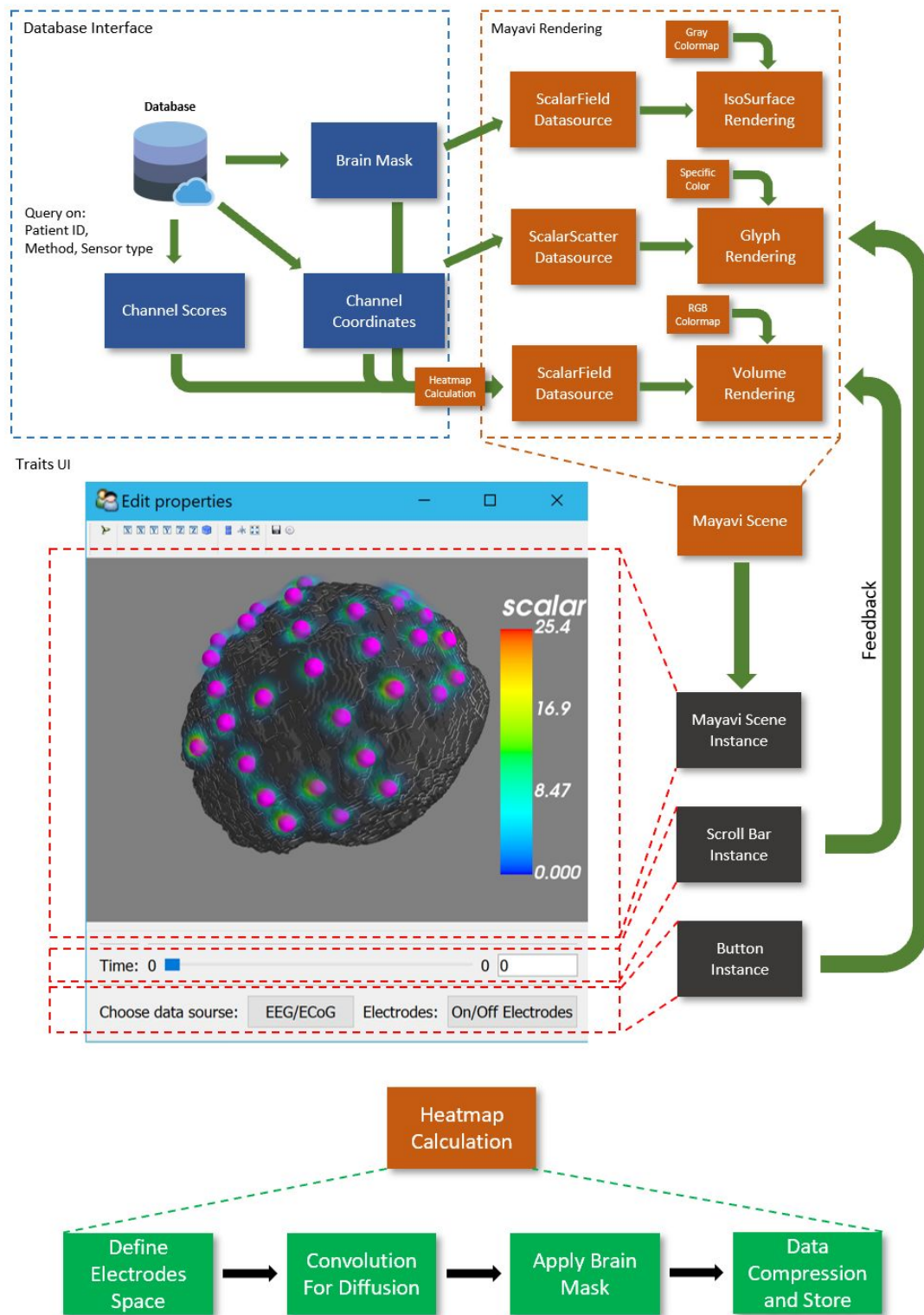


Figure 6-1. Schema of the entire module

The 3D rendering is facilitated by the powerful Mayavi engine. It provides a pipeline to structure customized scene. The pipeline starts from the Datasource, Filter to the Module. For example, we can import a ScalarField object which represents a scalar matrix. This object will contain many filters to apply and with the final output method including iso-surface, volume rendering, etc. The iso-surface will create a smooth contour while the volume rendering will, on the other hand, polish a transparent body according to colormap. Each pipeline can be combined to make a Mayavi scene.

The UI design is based on Traits that is very similar to Matlab GUI. It provides a very straightforward way to define button, dialogs, slide bar, text, etc. as well as their callback function. Most importantly, it supports Mayavi as a scene that can be directly integrated with other UI components.

## **Results**

The final output is also shown in Fig.6-1. Three objects are created. The brain is rendered in gray ios-surface to provide a solid background. Heatmap is rendered in colorful volumn and projected onto the brain. And the electrodes are rendered as uniform sphere, marking out the channel position. User can use the scroll bar to select different time frames. And the buttons are to decide to show/hide different objects.

Fig.6-2 is an example in which we map the Band Power from ECoG signals onto the brain and the red region demonstrates those intense activities. Those purple dots in Fig.6-2-a represents those electrodes used in ECoG experiment.

There are several visualization parameters that I would like to discuss. One of them is

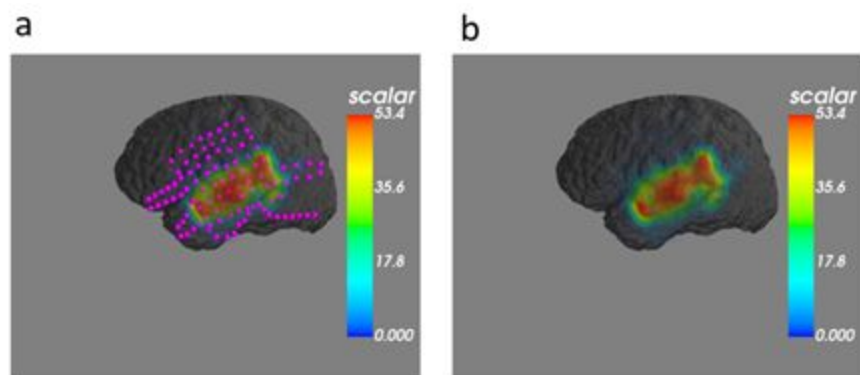


Figure 6-2 Heatmap Visualization

the convolution kernel size. Physically, it means that how concentration the signal on the cortex is. The larger the kernel is, the wider the signal is going to spread. We can tell the difference from Fig.6-3-a and b where b has a double kernel size as a's. The

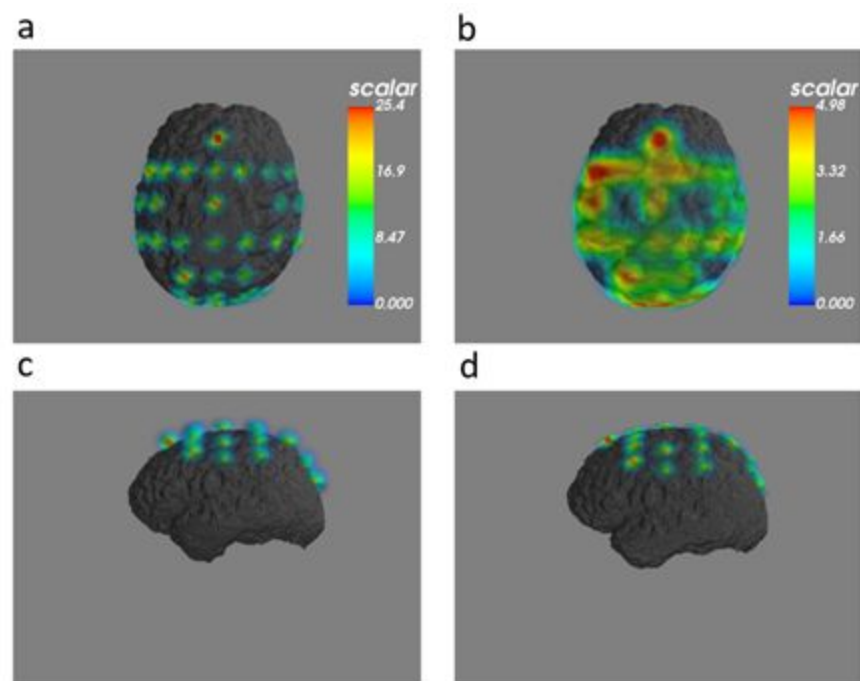


Figure 6-3 Different parameters and their visual effect. a and b differ in convolution kernel size. c and d differ in brain mask dilution size

other one parameter is brain mask's dilation size. This is purely visualization-oriented with no physical meaning. We set it because a better result will be generated if the brain mask is enlarged a little to keep more heatmap (other than a single surface layer). Fig.6-3-c shows a larger dilation size compared to Fig.6-3-d.

### **Discussion and Conclusion**

So we built a module dedicated to visualizing the brain activity with interactive heatmap. A gaussian kernel is applied to simulate the diffusion of signal on the brain cortex and user shall observe the activities of cortex according to the colors.

This might be able to facilitate the user to have a direct sense about signal processing result but is still a very preliminary and crude presentation. A further version will include more numeric analysis tools. For example, user needs to see the statistics (mean value, variance, correlation etc. ) besides the heatmap and also have the ability to hide or integrate parts of data.

Now what we render is simply a brain cortex surface. We can add options to show brain in sectioning plane or more complex way (like half of brain rip open while the other half stays solid as reference) in order to extend this module's application.

## C. DISCUSSION, CONCLUSIONS, AND FUTURE WORK

Creating a platform such as this one involves unifying many components together as well as overcoming compatibility issues, both software (Matlab v. Python) and hardware (Windows v. Mac OS). We hope that the development of such a pipeline will streamline and automate related tasks in the near future.

### Future Work:

- *Incorporating electrode knowledge into the electrode localization module:* knowing the shape and multiplicity of ECoG electrodes can help correct or fill-in electrode visualization by interpolating where the electrodes should be with the assumption of a specific electrode geometry or arrangement.
- *Utilizing depth electrodes as an input source:* currently, both ECoG and EEG are both surface inputs. Incorporating depth electrodes would involve modifications on a couple of fronts: modifying the brain template used for visualization in order to see inside the brain as well as modifying signal processing algorithms in order to account for the higher frequency input.
- *Real time visualization accompanying the P300 speller:* since the P300 speller uses EEG to acquire signals and the EEG uses a standard 10-20 system with known electrode configurations without needing to provide patient imaging data, real-time visualization is feasible.

## D. REFERENCES

1. Speier, W., Arnold, C., & Pouratian, N. (2016). Integrating language models into classifiers for BCI communication: a review. *Journal Of Neural Engineering*, 13(3), 031002. doi: 10.1088/1741-2560/13/3/031002
2. PostgreSQL. (2018). Retrieved from <https://en.wikipedia.org/w/index.php?title=PostgreSQL&oldid=864684407>
3. Procedures, S., PHP, P., Python, P., JDBC, P., Functions, A., & Functions, D. et al. (2018). What is PostgreSQL. Retrieved from <http://www.postgresqltutorial.com/what-is-postgresql/>
4. Bao, F., Liu, X., & Zhang, C. (2011). PyEEG: An Open Source Python Module for EEG/MEG Feature Extraction. *Computational Intelligence And Neuroscience*, 2011, 1-7. doi: 10.1155/2011/406391
5. Speier, W. (2018). Toward real-time communication using brain-computer interface systems. Retrieved from <https://escholarship.org/uc/item/8j79w06v>
6. 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.1 documentation. (2018). Retrieved from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
7. gmicros/MATLAB. (2018). Retrieved from <https://github.com/gmicros/MATLAB/tree/master/BCI%20Initial%20Assignment>
8. edfRead - File Exchange - MATLAB Central. (2018). Retrieved from <https://www.mathworks.com/matlabcentral/fileexchange/31900-edfread>
9. The Excel FFT Function v1.2. (2018). Retrieved from <http://physics.oregonstate.edu/~grahamat/COURSES/ph421/lec/ExcelFFT.pdf>
10. MATLAB Runtime - MATLAB Compiler. (2018). Retrieved from <https://www.mathworks.com/products/compiler/matlab-runtime.html>
11. voytekresearch/tutorials. (2018). Retrieved from <https://github.com/voytekresearch/tutorials/blob/master/Phase%20Amplitude%20Coupling%20Tutorial.ipynb>
12. Tutorials/TutPac - Brainstorm. (2018). Retrieved from <https://neuroimage.usc.edu/brainstorm/Tutorials/TutPac>
13. Özkurt, T., & Schnitzler, A. (2011). A critical note on the definition of phase–amplitude cross-frequency coupling. *Journal Of Neuroscience Methods*, 201(2), 438-443. doi: 10.1016/j.jneumeth.2011.08.014
14. Seymour, R., Rippon, G., & Kessler, K. (2017). The Detection of Phase Amplitude Coupling during Sensory Processing. *Frontiers In Neuroscience*, 11. doi: 10.3389/fnins.2017.00487
15. Cross Frequency Interactions – Attention Circuits Control Lab. (2018). Retrieved from <http://accl.psy.vanderbilt.edu/resources/analysis-tools/cross-frequency-interactions/>
16. Introduction - Brainstorm. (2018). Retrieved from <https://neuroimage.usc.edu/brainstorm/>
17. Huelsemann, M., Naumann, E. and Rasch, B. (2018). Quantification of Phase-Amplitude Coupling in Neuronal Oscillations: Comparison of Phase-Locking

Value, Mean Vector Length, and Modulation Index. bioRxiv, 290361. doi: <https://doi.org/10.1101/290361>

18. EEG (Electroencephalogram): Purpose, Procedure, and Risks. (2018). Retrieved from <https://www.healthline.com/health/eeg>
19. Electrocorticography - an overview | ScienceDirect Topics. (2018). Retrieved from <https://www.sciencedirect.com/topics/neuroscience/electrocorticography>
20. Hall, J. (2016). Guyton and Hall textbook of medical physiology. Elsevier, 551-560.
21. Yaniv, Z., Lowekamp, B., Johnson, H., & Beare, R. (2017). SimpleITK Image-Analysis Notebooks: a Collaborative Environment for Education and Reproducible Research. *Journal Of Digital Imaging*, 31(3), 290-303. doi: 10.1007/s10278-017-0037-8
22. Lacadie, C., Fulbright, R., Rajeevan, N., Constable, R., & Papademetris, X. (2008). More accurate Talairach coordinates for neuroimaging using non-linear registration. *Neuroimage*, 42(2), 717-725. doi: 10.1016/j.neuroimage.2008.04.240
23. Fonov, V., Evans, A., McKinstry, R., Alml, C., & Collins, D. (2009). Unbiased nonlinear average age-appropriate brain templates from birth to adulthood. *Neuroimage*, 47, S102. doi: 10.1016/s1053-8119(09)70884-5
24. Ramachandran, P. and Varoquaux, G. (2011). "Mayavi: 3D Visualization of Scientific Data." *IEEE Computing in Science & Engineering*, 13 (2), 40-51.
25. enthought/traitsui. (2018). Retrieved from <https://github.com/enthought/traitsui>.
26. Brett, M., Gerhard, S., Haselgrove, C., & Hanke, M. (2018). nipy/nibabel. Retrieved from <https://github.com/nipy/nibabel>
27. Virbel, M., Pettier, G., Arora, A., Kovac, J., Taylor, A., & Einhorn, M. et al. (2018). Kivy: Cross-platform Python Framework for NUI. Retrieved from <https://kivy.org>
28. Shattuck, D., & Leahy, R. (2002). BrainSuite: An automated cortical surface identification tool. *Medical Image Analysis*, 6(2), 129-142. doi: 10.1016/s1361-8415(02)00054-3
29. Shattuck, D. (2018). BrainSuite Batch Processing. Retrieved from [http://brainsuite.org/wp-content/uploads/2017/06/BrainSuite\\_Workshop\\_2017\\_Batch\\_David\\_Shattuck.pdf](http://brainsuite.org/wp-content/uploads/2017/06/BrainSuite_Workshop_2017_Batch_David_Shattuck.pdf)