Summary of Algorithm

The goal of this project was to predict finger flexion motion captured by the data glove based on subject ECoG data records. Our algorithm pre-processed the data by first removing noisy channels, filtering the data, and subtracting constant noise. We then extracted 6 unique features from each channel for every subject's dataset. The first feature is average time domain signal voltage, and the subsequent five features were the average frequency domain voltage of five different frequency bands determined from consulting literature. A linear decoder was used to create a model for the four fingers with a specific linear filter for each finger. The predicted finger angles were zero clamped and smoothed, and initial predictions were spline interpolated to match the dimensions of the provided finger angles. Methods to develop this algorithm were influenced by published literature, past engineering competitions, and guidance provided by course instructors. Decisions made throughout the process were motivated by trial and error and cross-validation, with further guidance from submissions to the leaderboard and resulting correlations to a portion of the leaderboard dataset. Based on the resulting leaderboard correlations, these predictions achieved a maximum correlation of 0.5072 with the averaged correlations of all four fingers.

Detailed Explanation of Algorithm

Pre-Processing **|** Data was pulled from the iEEG portal for all subjects. Upon consulting the literature, Channel 55 was removed for Subject 1, Channels 21 and 38 were removed for Subject 2, and all channels were kept for Subject 3. For each subject, the mean value across all channels was subtracted from each record to remove constant noise from the signal. Data for all subjects was filtered using a powerline Butterworth filter. Powerline noise present in biomedical signals is additive in nature, so the data was filtered to better determine features using a fourth order Butterworth bandpass filter with cutoff frequencies of 1 and 175 Hz. Fourth order was used as Butterworth filters present less ringing and overshoot with increased order.

Extracting Features **|** The algorithm extracted 6 features using a 100 ms sliding window in the time domain with a 50 ms overlap, applied across all subject data. Features were chosen and adapted based on past machine learning algorithms and published literature. Feature 1, the average time domain voltage, was calculated by taking the mean of the signal over each time window using the anonymous function "MovingWinFeats". The time-series ECoG data was then converted into the frequency domain to calculate spectral amplitudes between 0-200 Hz using 1 Hz bins. Features 2-6 were the average frequency domain magnitudes across the selected frequency bands: 5-15 Hz, 20-25 Hz, 75-115 Hz, 125-160 Hz, 160-175 Hz. Features 2-6 were calculated using MATLAB's spectrogram function, which takes a signal with a specified time window and non-overlap, a vector of cyclical frequencies, and the sampling frequency to output the short time Fourier transform of the signal. The absolute value of the real component of the Fourier transform was used to calculate the features. All six features for each channel of a subject were concatenated into a single feature matrix.

Building the Model **|** The basis of this algorithm is structured around the optimal linear decoder method described in Warland et al. Using six features from N=3 prior time bins, hand position was calculated at a given time bin. A response matrix R was built off of the feature matrix for a given subject. Each column in R corresponds to a feature, and each row corresponds to the successive time bin. Each feature was repeated for N time bins across the total number of time bins for each subject. Data glove datasets corresponding to the appropriate subjects were used as the target vectors. In order to make the data glove data compatible with the feature matrix calculated for each subject, the data glove data was downsampled by a factor of 50 Hz to match the frequency of extracted features from the training ECoG dataset. An optimal linear filter f was built for each finger using the following equation: $\mathbf{f = (R^T R)^{-1} (R^T s)}$.

These filters were then used to calculate optimal predictions. Testing ECoG data was imported and pre-processed according to the aforementioned methods. R matrices were constructed on the testing ECoG data using the same methods as the R matrices constructed on the training ECoG data. Predictions were made on the training data using the following equation: $\mathbf{U = Rf}$.

Post-Processing **|** Post-processing of finger predictions included zero-clamping, smoothing, and spline interpolation. First, predictions were zero-clamped such that negative finger predictions were replaced with zeros to resemble the resting baseline period of the data glove. The reasoning behind zero-clamping our predictions was to better incorporate real-world constraints imposed on finger flexion. Moreover, this method was used successfully in past BCI competitions. Then, predicted finger angles were smoothed using MATLAB’s movmean function, which calculates a moving average over a signal. After trying several values, we calculated a moving average across 7 points on either side of each prediction angle. This smoothing processing was integral in raising our predicted angles’ accuracy as the initial predictions were very noisy. Then, a cubic spline interpolation was used to match the predicted data glove values to the length of the ECoG training data.

Flow Chart of Algorithm

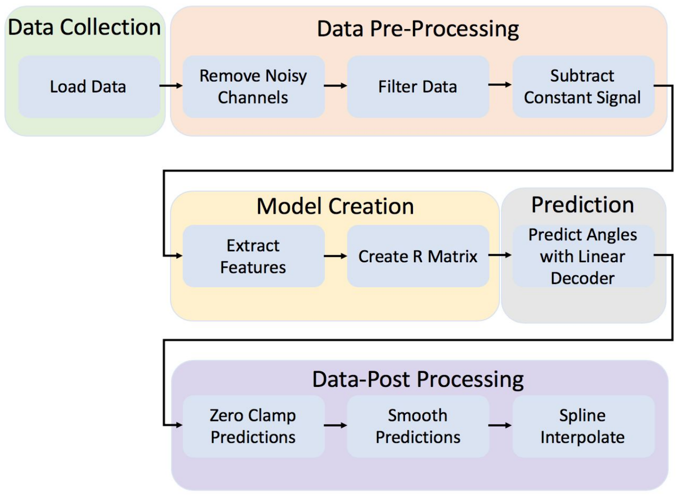

Figure 1. Algorithm Flow Chart. Flow chart details the general steps involved in our algorithm.

Unsuccessful Attempted Methods

***Filtering* |** When trying to filter the raw ECoG data, our group tried different filtering methods before deciding on a bandpass Butterworth filter, as well as experimented with different parameters for the filters themselves. We attempted to use a high-pass elliptic filter in order to reduce noise at 60 Hz, however we did not see as much noise reduction as with Butterworth filtering. Our parameter choices were influenced by the paper by Akwei-Sekyere which compares various methods to try and reduce powerline noise. Our team then tried to smooth the filtered data before feature extraction. However, upon graphing the smoothed data against only filtered data, the smoothing did not make any significant differences. Upon searching literature, we decided to try smoothing after creating the feature and R matrices. When we moved the smoothing function to the post-processing portion of the algorithm, we saw the results we had wanted, and our correlation increased immensely as a result.

***Post-Processing: Zero-Clamping & Smoothing* |** There were a handful of other variables that we experimented with in the post-processing step, namely zero-clamping and smoothing. Originally, we did not set limits for the values of data glove predictions. However, we quickly discovered that our algorithm would predict more accurately if we clamped all negative predictions to zero, as the data glove would output values close to zero if ever negative. Additionally, we had trouble finding the best way to smooth our predictions. Through iterating our algorithm on both a testing subset of the training data and the leaderboard data, we were also able to determine the optimal smoothing window (we tried window sizes of 10^0, 10^1, 10^2, and 10^3 orders of magnitude) to be 7.

***Interpolation* |** Our team attempted to use zero order interpolation to match the dimensions of the provided finger angles for each subject. Zero order interpolation involves copying each predicted angle a given number of times to reach the desired length of data. We had initially thought that zero order interpolation would be more successful than spline interpolation as each finger angle in the provided data repeats several times before the finger angle changes. However, we realized that zero order interpolation resulted in magnified error between the predicted and true finger angle as predicted values far from the true values were repeated, reducing correlation between the predictions and true angles. Instead, our team decided to use spline interpolation to get the dimension of the predicted angles to match the true finger angles. Spline interpolation fits a curve between data points, which effectively reduced the difference between the predicted and true finger angles, resulting in a higher correlation between the two datasets.

Figure of Interesting Portion of Algorithm

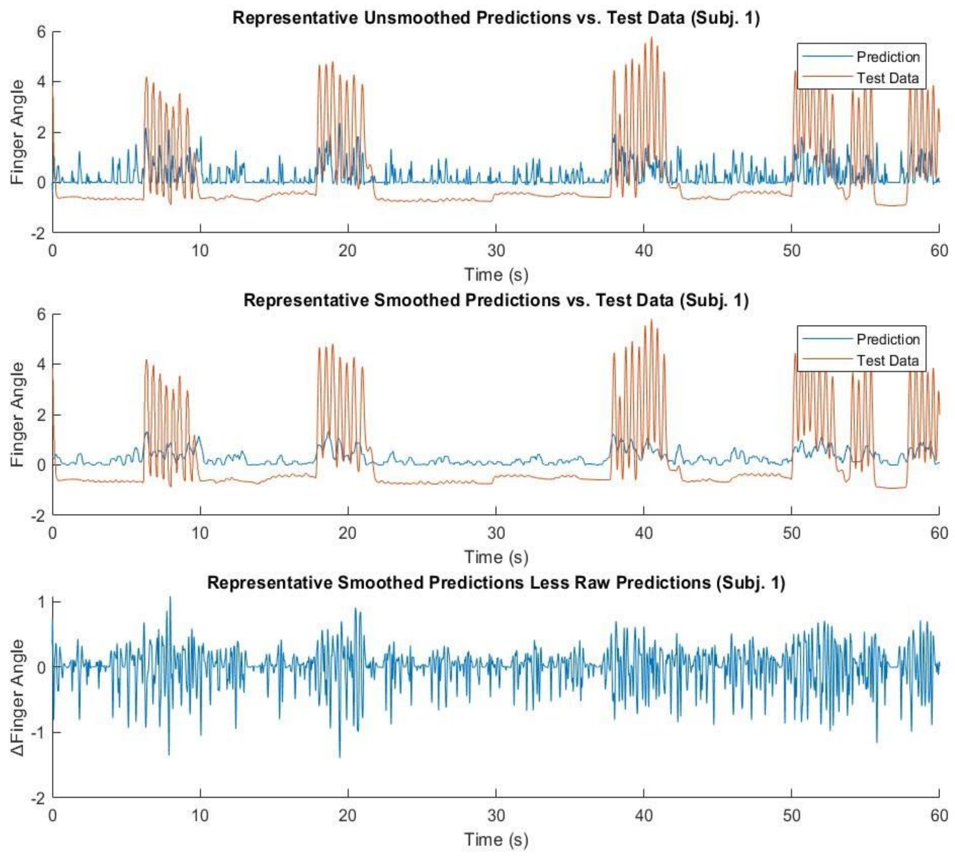

Figure 2. Effects of prediction smoothing, visualized. We discovered that our optimal success lied in smoothing predictions before interpolating our output to compute correlations. In the figure above, a sliding window of 7 samples was used (3 antecedent and 3 subsequent samples to the centerpoint) to compute an average value for each set of predictions. The movmeans function on MATLAB was used to achieve the desired smoothing.

Why the Fourth Finger is Correlated with the Third and Fifth Fingers

The fourth finger is not included in the datasets as it is highly correlated with the third and fifth finger, so it is removed entirely from all datasets and hence not included in the algorithm. The reasoning behind removing the fourth finger is likely due to limitations of neuromuscular control during arc-movements, which have the largest effects on the ring (fourth) finger, followed by the little (fifth) finger. Mechanical coupling further limits the independence of the index (second), middle (third), and ring fingers to the greatest degree, followed by the little finger. For the movements captured by the data glove, mechanical coupling between fingers in conjunction with the neuromuscular control limitations appear to be key factors affecting finger independence. Because the fourth finger has the largest cumulative effect between both motion constraints, it could be inferred that the ring finger would have the least amount of independence. Thus, it is within reason to not consider the fourth finger when testing the dataset or include it in the algorithm.

Conclusion

Overall, this was a very challenging but rewarding project. Initially, our team started by implementing the outlined "Simple Method" adapted from Kubanek et al. We then experimented with different filtering methods to preprocess the data, and ultimately decided on a bandpass Butterworth filter to preprocess the raw ECoG data as the filter yielded the highest cross-validation outputs. While we struggled with the leaderboard submissions initially (our group name had an apostrophe in our team name…), we were able to pass Checkpoint 1 using these methods. And, after changing our name, we were able to submit the leaderboard!

To increase our correlation and improve our algorithm for Checkpoint 2, we explored different frequency bands used to create our features according to published literature. We found our adjustments did not change our cross-validation or our leaderboard correlation significantly. So, to further develop our algorithm, we tried various methods of smoothing our data as it still remained noisy. We ultimately decided on using a moving median average for smoothing, as described earlier, and our correlation increased significantly. We ended the competition having achieved a maximum correlation of 0.5072.

Throughout the project, we enjoyed implementing what we learned this semester into our final algorithm. With trial and error, consulting literature, time and effort, we were able to develop (what we believe to be) a robust algorithm representative of our knowledge. If we had more time, we would have liked to try to explore more methods to further advance our algorithm, such as by optimizing our code for each finger, as well as implementing post-processing filtering.

References

[1] Akwei-Sekyere, S (2015). Powerline noise elimination in biomedical signals via blind source separation and wavelet analysis. *PeerJ*, 3, 1086. doi:10.7717/peerj.1086

[2] Kubanek, J, Miller, KJ, Ojemann, JG, Wolpaw, JR & Schalk, G (2009). Decoding flexion of individual fingers using electrocorticographic signals in humans. *J Neural Engineering*, 6(6):066001.

[3] Liang, N & Bougrain, L (2012). Decoding Finger Flexion from Band-Specific ECoG Signals in Humans. *Frontiers in neuroscience*, 6, 91. doi:10.3389/fnins.2012.00091

[4] T. Hayashi, H. Yokoyama, I. Nambu and Y. Wada (2017). Prediction of individual finger movements for motor execution and imagery: An EEG study. 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, 2017, pp. 3020-3023.

[5] V. G. Kanas, I. Mporas, H. L. Benz, K. N. Sgarbas, A. Bezerianos and N. E. Crone (2014). Joint Spatial-Spectral Feature Space Clustering for Speech Activity Detection from ECoG Signals. *IEEE Transactions on Biomedical Engineering*, vol. 61, no. 4, pp. 1241-1250.

Appendix

filter_data.m

```
function clean_data = filter_data(raw_eeg)
%
%
% Input:    raw_eeg (samples x channels)
%
% Output:   clean_data (samples x channels)
%
%
%% Filter Data

% 60hz
[b,a] = butter(4, [59 61]./(1000/2), 'stop');
% 120hz
[b2,a2] = butter(4, [119 121]./(1000/2), 'stop');

% bandpass
[bband,aband] = butter(4, [1 175]/(1000/2), 'bandpass');

% Apply filters
clean_data = filtfilt(b, a, raw_eeg);
clean_data = filtfilt(b2, a2, clean_data);
clean_data = filtfilt(bband, aband, clean_data);

end
```

getWindowedFeats.m

```
function [all_feats] = getWindowedFeats(raw_data, fs, window_length, window_overlap)
%
% Inputs:     raw_data:         The raw data for all patients
%             fs:               The raw sampling frequency (Hz)
%             window_length:    The length of window (s)
%             window_overlap:   The overlap in window (s)
%
% Output:     all_feats:      All calculated features


%  First, filter the raw data
clean_data = filter_data(raw_data);

% Compute number of windows required
NumWins = @(xLen, fs, winLen, winDisp) floor(((xLen - winLen*fs + winDisp*fs)/(winDisp*fs)));
nWins = NumWins(size(raw_data, 1), fs, window_length, window_overlap);

nFeats = 6; % Number of features being calculated per window per channel
windowStart = 1; % Then, loop through sliding windows
numChannels = size(clean_data, 2); % Number of channels in data

all_feats = zeros(nWins, size(raw_data, 2)*nFeats);
for jj = 1:nWins

   % Within loop calculate feature for each segment (call get_features)
   windowIndices = [windowStart, windowStart + window_length*fs];

   window = clean_data(windowIndices(1):windowIndices(2)-1, :);

   % Apply get_features to window
   features = get_features(window,fs);
   all_feats(jj, 1:numChannels) = features;

   % Update start point of the window for next iteration
   windowStart = windowStart + window_overlap*fs;
end

% Compute spectrogram over windows in 5 frequency bands
f = 1:200;
band1 = zeros(nWins, size(raw_data, 2));
band2 = zeros(nWins, size(raw_data, 2));
```

```
band3 = zeros(nWins, size(raw_data, 2));
band4 = zeros(nWins, size(raw_data, 2));
band5 = zeros(nWins, size(raw_data, 2));

for i = 1:numChannels
    [S,F,~] = spectrogram(clean_data(:, i), window_length*1E3, round((window_overlap)*1E3), f, fs);
    S_real = abs(real(S));

    band1(:, i) = mean(S_real(5:15, :))';
    band2(:, i) = mean(S_real(20:25, :))';
    band3(:, i) = mean(S_real(75:115, :))';
    band4(:, i) = mean(S_real(125:160, :))';
    band5(:, i) = mean(S_real(160:175, :))';
end

i = 1;
% Add frequency bands to feature matrix
all_feats(:, numChannels*i+1:numChannels*(i+1)) = band1;
i = i+1;
all_feats(:, numChannels*i+1:numChannels*(i+1)) = band2;
i = i+1;
all_feats(:, numChannels*i+1:numChannels*(i+1)) = band3;
i = i+1;
all_feats(:, numChannels*i+1:numChannels*(i+1)) = band4;
i = i+1;
all_feats(:, numChannels*i+1:numChannels*(i+1)) = band5;

% Return feature matrix
all_feats = normalize(all_feats);

end
```

get_features.m

```
function [features] = get_features(clean_data, fs)
%
%
% Input:    clean_data: (samples x channels)
%           fs:         sampling frequency
%
% Output:   features:   (1 x (channels*features))
%                       (e.g. Ch1_F1, Ch1_F2 ... Ch2_F1, Ch2_F2 ...)
%
%% Get features

% Feature 1: Average Time Voltage
avg_time_voltage = mean(clean_data, 1);

features = avg_time_voltage;

end
```

create_R_matrix.m

```
function [R] = create_R_matrix(features, N_wind)
%
%
% Input:    features:   (samples x (channels*features))
%           N_wind:     Number of windows to use
%
% Output:  R:       (samples x (N_wind*channels*features))
%
%
%% Create R Matrix

% Initiate R matrix
R = ones(size(features, 1), N_wind*size(features, 2) + 1);

% Populate first N_wind-1 rows of R matrix
for jj = 1:N_wind - 1
    R(jj, 2:end) = repmat(features(jj, :),[1 N_wind]);
end

% Populate the remaining rows of R matrix
for jj = N_wind:size(features, 1)
    temp = features(jj - (N_wind - 1):jj, :);
    R(jj, 2:end) = temp(:)';
end

% Replicate last row and append to bottom to conserve matrix dimensions
R(end + 1, :) = R(end, :);

end
```

make_predictions.m

```
function [predicted_dg] = make_predictions(test_ecog)

% INPUTS: test_ecog - 3 x 1 cell array containing ECoG for each subject, where test_ecog{i}
% to the ECoG for subject i. Each cell element contains a N x M testing ECoG,
% where N is the number of samples and M is the number of EEG channels.

% OUTPUTS: predicted_dg - 3 x 1 cell array, where predicted_dg{i} contains the
% data_glove prediction for subject i, which is an N x 5 matrix (for
% fingers 1:5)

% Run time: The script has to run less than 1 hour.

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

% Remove bad channels
test_ecog{1}(:, 55) = [];
test_ecog{2}(:, 21) = [];
test_ecog{2}(:, 38) = [];

% Subtract average value
test_ecog{1} = test_ecog{1} - mean2(test_ecog{1});
test_ecog{2} = test_ecog{2} - mean2(test_ecog{2});
test_ecog{3} = test_ecog{3} - mean2(test_ecog{3});

% Set parameters
fs = 1e3;                  %Hz
window_length = 0.1;       %s
window_overlap = 0.05;  %s

% Load in Trained Model
load('training_feat_mats.mat')
s1_f_model1 = training_feat_mats{1};
s2_f_model1 = training_feat_mats{2};
s3_f_model1 = training_feat_mats{3};

% Get windowed features
s1_featMat_leaderboard = getWindowedFeats(test_ecog{1}, fs, window_length, window_overlap);
s2_featMat_leaderboard = getWindowedFeats(test_ecog{2}, fs, window_length, window_overlap);
s3_featMat_leaderboard = getWindowedFeats(test_ecog{3}, fs, window_length, window_overlap);

% Calculate R matrix
```

```
N = 3;
s1_R_leaderboard = create_R_matrix(s1_featMat_leaderboard, N);
s2_R_leaderboard = create_R_matrix(s2_featMat_leaderboard, N);
s3_R_leaderboard = create_R_matrix(s3_featMat_leaderboard, N);

% Make predictions
s1_yhat_leaderboard = s1_R_leaderboard*s1_f_model1;
s2_yhat_leaderboard = s2_R_leaderboard*s2_f_model1;
s3_yhat_leaderboard = s3_R_leaderboard*s3_f_model1;

% Zero Clamp
s1_yhat_leaderboard(s1_yhat_leaderboard < 0) = 0;
s2_yhat_leaderboard(s2_yhat_leaderboard < 0) = 0;
s3_yhat_leaderboard(s3_yhat_leaderboard < 0) = 0;

% Smooth
s1_yhat_leaderboard  = movmean(s1_yhat_leaderboard, 7);
s2_yhat_leaderboard  = movmean(s2_yhat_leaderboard, 7);
s3_yhat_leaderboard  = movmean(s3_yhat_leaderboard, 7);

% Spline interpolation
x = linspace(0, 147500, length(s1_yhat_leaderboard))';
xx = 0:147500-1;
s1_yhat_leaderboard_interp = zeros(147500, 5);
s2_yhat_leaderboard_interp = zeros(147500, 5);
s3_yhat_leaderboard_interp = zeros(147500, 5);
for i = 1:5
    s1_yhat_leaderboard_interp(:,i) = spline(x,s1_yhat_leaderboard(:,i), xx);
    s2_yhat_leaderboard_interp(:,i) = spline(x,s2_yhat_leaderboard(:,i), xx);
    s3_yhat_leaderboard_interp(:,i) = spline(x,s3_yhat_leaderboard(:,i), xx);
end

% Assign interpolated predictions to predicted_dg
predicted_dg{1} = s1_yhat_leaderboard_interp;
predicted_dg{2} = s2_yhat_leaderboard_interp;
predicted_dg{3} = s3_yhat_leaderboard_interp;
predicted_dg = predicted_dg.';

end
```