Abstract

This chapter presents the implementation of several electroencephalogram (EEG)-based brain–computer interface (BCI) signal processing approaches for evoked potentials. Sample code is provided for two well-established BCI paradigms: (1) P300 Oddball and (2) Steady-State Visual Evoked Potentials (SSVEP). These examples can be straightforwardly extended to more sophisticated designs employing evoked potentials.

1. Transient Evoked Potentials: Four-class P300 Oddball BCI (synchronous operation with discrete selections)
2. Steady-state evoked potentials: $n$-class SSVEP (steady-state visually evoked potential) BCI (asynchronous operation with discrete selections)
20.1 INTRODUCTION
This chapter provides an introductory tutorial on how to implement several fundamental electroencephalogram (EEG)-based brain–computer interface (BCI) processing approaches for evoked potentials using MATLAB®. The examples are intended to provide a simplified, hands-on framework for better understanding the basic structure and parameters used for BCI processing applications. Two examples will be presented, which are representative of common BCI processing scenarios and can be straightforwardly extended to more specific and complex scenarios:

1. Transient Evoked Potentials: Four-class P300 Oddball BCI (synchronous operation with discrete selections)
2. Steady-state evoked potentials: n-class steady-state visually evoked potential (SSVEP) BCI (asynchronous operation with discrete selections)

20.2 FOUR-CLASS P300 ODDBALL
This scenario is based on the classical P300 oddball where stimuli (visual, auditory, or tactile) are presented sequentially and one of the stimuli is to be selected as the target (Farwell & Donchin 1988). In this case, four stimuli are repeatedly presented in block-wise random order. The target stimuli elicit a characteristic P300 response, whereas the nontarget stimuli should not elicit the same neural behavior. A classifier can then be trained in order to distinguish between the target and nontarget stimuli. A segment of the data, denoted as a response, is collected after every stimulus. Multiple responses are collected for a given stimulus and averaged to increase the reliability of response detection. The averaged response for each stimulus is input into a classifier to determine which stimulus was a target, that is, elicited a characteristic P300 response. Classifier calibration is typically required before online operation. When collecting the calibration data, the user is instructed which stimulus to focus on. This is done so that the responses can be properly labeled in order to train the classifier to a given user. The following basic scenario can be generalized to the P300 speller and other variants.

20.2.1 DATA COLLECTION
The characteristic P300 event-related potential (ERP) response is typically found to be strongest over the parietal lobe. Additional relevant neural information can also be found in areas over the occipital lobe, especially when the stimuli are presented visually. The typical setup for a P300 visual oddball BCI includes electrode coverage over Fz, Cz, Pz, P3, P4, P07, P08, and Oz. EEG data from these channels are collected synchronously using stimulus presentation software that time locks the stimulus events to the EEG, for example, BCI2000 and Psychtoolbox. The sampling rate should be kept above roughly 100 Hz in order to accurately reflect the time progression of the P300 waveform. Here, we assume the data to be loaded into MATLAB in a matrix format of $k \times n$ samples × channels. We also assume that there is a vector of length $k$ denoting the start of each stimulus, with zeros everywhere excepting “1” through “4” where a stimulus starts. An additional vector of length $k$ is also necessary to record during the collection of calibration data, where a “1” represents the presentation of a target oddball stimulus, and zeros everywhere else.

20.2.2 DATA PREPROCESSING
The data can be band-pass filtered between 0.1 Hz and approximately 20 to 30 Hz, and then decimated. Care must be taken to accordingly downsample the label vectors to match the rate of the signal. This frequency range was chosen as it still allows for an accurate representation of the dynamics of the EEG ERPs while limiting the number of temporal data points that will ultimately be used as features in our classifier. This smoothing of the data often helps to prevent overfitting of
classifiers. Additionally, there is a computational advantage to starting with a smaller feature space, but with modern computing capabilities, this is becoming less of an issue. Once the data have been preprocessed, the next step is to extract the useful features from the EEG data.

### 20.2.3 Data Segmentation and Feature Extraction

Below is a sample MATLAB function that takes a labeled EEG signal matrix (EEGsignal) and outputs a MATLAB structure containing a matrix of ERPs corresponding to each stimulus label (StimulusCode). Section 1 determines the sample index corresponding to the onset of each stimulus, in this case the transition of StimulusCode from 0 to any label 1–4. The ERPwindow input argument defines the number of samples before and after each stimulus that contains the desired ERP window (this can be converted to milliseconds based on the sampling rate if desired). The ERP windows will be positioned at each stimulus transition. For example, assuming a sampling rate of 100 Hz and a P300 paradigm, a reasonable window would be −100 to 800 ms in order to capture the entire temporal dynamics of the ERP. In this case, an ERPwindow = [−10 80] would collect 10 samples before and 80 samples after each stimulus onset for a total of 91 samples representing each ERP (counting the zeroth sample). Samples collected before the stimulus onset can be used to correct for baseline signal shifts before the stimuli. The final two lines of the function extract the stimulus and type labels for each ERP in the resulting matrix for use in classifier training and validation. It should be noted that the functions in this chapter do not include any error or exception checking. It is left to the user to ensure that the input arguments and variables are properly constructed and configured for specific application scenarios (e.g., window positions should not exceed the length of the signal, etc.). Refer to Figure 20.1 for a graphical depiction of the variables.

```matlab
function [ERPs]=GetERPs(EEGsignal,StimulusCode,StimulusType,ERPwindow)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function extracts time-locked EEG responses (ERPs) from a single trial of a P300 Oddball paradigm.
% % input arguments:
% % EEGsignal: Preprocessed EEG signal matrix [time samples X channels]
% % StimulusCode: Values 0-4. 0 for all samples when no stimulus is presented, 1-4 for all samples associates when respective stimulus 1-4 is presented. Same number of samples as signal.
% % StimulusType: 1 for all samples for which a target stimulus is presented. 0 for all samples for which a non-target stimulus is presented. Same number of samples as signal.
% % ERPwindow: ERP window start and end time samples with respect to stimulus onset of 0 samples [begin end]
% % output arguments:
% % ERPs: Structure containing the following:
% % ERPs: Array of collected responses [stimuli X response window X channels]
% % ERPCode: Vector containing the StimulusCode of each ERP [stimuli X 1]
% % ERPType: Vector containing the StimulusType of each ERP [stimuli X 1]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```
### Section 1: Identify the onset of the stimuli

```matlab
[NumberSamples, NumberChannels] = size(EEGsignal);
StimulusIndex = find(StimulusCode(1:NumberSamples-1) == 0 & ...
    StimulusCode(2:NumberSamples) ~= 0) + 1; % find indices of stim onset
NumberStimuli = length(StimulusIndex); % number of stimuli in trial
```

### Section 2: Extract the ERPs from the EEG signal

```matlab
ERPs.ERPs = zeros(xx, ERPwindow(2) - ERPwindow(1), NumberChannels);
for cnt = 1:NumberStimuli % gather the ERPs
    ERPs.ERPs(cnt, :, :) = EEGsignal(ind(kk) + ERPwindow(1) - 1: ...
        ind(kk) + ERPwindow(2) - 2,:);
end

ERPs.Code = StimulusCode(StimulusIndex); % extract stimulus code for each ERP
ERPs.Type = StimulusType(StimulusIndex); % extract stimulus type for each ERP
```

**FIGURE 20.1** Graphical depiction of the variables for the P300 oddball paradigm. The figure shows the alignment of one sample EEG channel, `EEGsignal(:,1)`, with the data labels. The consecutive `ERPwindows` are aligned with the transition of `StimulusCode` from zero to nonzero values (i.e., when the next stimulus is presented). `StimulusType` represents whether the stimulus is a target (“1”) or nontarget (“0”).
20.2.4 Classifier Training

The output of the function from the Section 20.2.3 contains an array of ERPs with dimensions (stimulus, samples, channels) along with the associated stimulus labels Code and Type, both vectors of dimension (1, stimulus). All or a subset of the ERPs in the array can be used to train a classifier to determine whether a single trial or averaged ERP is the result of a target or nontarget stimulus (as defined by vector Type). Below is a sample MATLAB function for creating a simple binary classifier based on linear discriminant analysis (LDA) (Fisher 1936) to classify the ERPs as targets or nontargets. Note that this is one of the most simplistic classifiers, which is selected primarily for pedagogical purposes. However, LDA classifiers can be very effective in certain contexts. Alternative classifiers such as support vector machines, artificial neural networks, and so on can also be implemented by replacing the *regress*() function call with one for another classifier with comparable input/output arguments (Krusienski et al. 2006). The input argument TrainERPs is the same form as the ERP structure from the Section 20.2.3, but renamed for the case that the ERPs were separated for training and testing. The first line of code reshapes the spatiotemporal ERPs as (samples*channels, stimuli) to create a single feature vector corresponding to each stimulus, as required by the *regress*() classification function. The *regress*() function assumes a linear combination of each of the samples*channels feature in the input vector. The function finds a weight for each feature that minimizes the mean squared error between the equation output and the Type variable representing the targets and nontargets. These weights can be applied to independent ERPs to estimate if they are a target or nontarget. Essentially, the EEG measurement at each spatial location (channel) and time point with respect to the stimulus is being weighted to form a version of a “spatiotemporal detection template.” For example, by taking an independent stimulus observation feature vector (samples*channels) and vector multiplying it by FeatureWeights, the result will be a single scalar value used for classification.

```matlab
function [FeatureWeights]=TrainERPClassifier(TrainERPs,ERPwindow)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function is used to generate feature weights for a linear classifier from a set of labeled responses.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% input arguments:
% TrainERPs: Structure containing collected ERPs and stimulus code/type labels
% windowlen: Begin and end time samples after stimuli of collected responses [begin end]

% output arguments:
% FeatureWeights: Vector containing the weights to multiply to each spatiotemporal feature for classification

FeatureVectors=reshape(TrainERPs.ERPs,NumberERPs,NumberChannels*WindowLen);
FeatureWeights=regress(TrainERPs.Type,[FeatureVectors,...
    ones(1,size(FeatureVectors,1))‘]);
FeatureWeights=FeatureWeights(1:end-1);  % exclude the bias term
```

20.2.5 Classification

At this point, classification to identify which stimulus evoked the oddball response can be performed. For Type labels of 0 and 1, the decision threshold will be 0.5 for deciding if an ERP
represents a target or nontarget response. That is, if this scalar is <0.5, the ERP can be classified as a nontarget, and if this scalar is >0.5, the ERP can be classified as a target. If it is known that only one class contains a target, selecting the highest scalar output is a sensible approach and is implemented in the following TestClassifier() function. Keep in mind that ERPs are generally formed by averaging multiple single-trial responses. However, this same detection approach can be applied to the averaged feature vectors.

function
[PredictedStimulus]=TestClassifier(TestERPs,FeatureWeights,ERPwindow);

% This function tests the performance of a linear classifier generated using % TrainClassifier.m

% input arguments:
% TestERPs: Structure containing collected responses and labels
% FeatureWeights: Vector containing the weights to multiply to each spatiotemporal feature for classification
% ERPwindow: ERP window start and end time samples with respect to stimulus onset of 0 samples [begin end]

% output arguments:
% PredictedStimulus: Predicted target stimulus for a given trial

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Section 1: Put test data through the classifier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
WindowLen= ERPwindow(2)-ERPwindow(1);
[NumberERPs,NumberChannels]=size(TrainERPs.ERPs);
FeatureVectors=reshape(TrainERPs.ERPs,NumberERPs,NumberChannels*WindowLen);
scores=FeatureVectors*FeatureWeights;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Section 2: Select the predicted ‘Target’ stimulus
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for stimuli=1:4
    AvgScore(stimuli)=mean(scores(TestERPs.Code==stimuli));
end
[~,PredictedStimulus]=max(AvgScore);

20.2.6  Considerations
The structure of the classification function presented assumes that a decision will be made after a fixed number of presentations of each of the four stimuli have occurred. However, this may be suboptimal in terms of information transfer rate, and a method for dynamic stopping after a variable number of stimuli could also be implemented (Throckmorton et al. 2013).

20.3  n-CLASS SSVEP
This scenario is based on an SSVEP interface with arbitrary number $n$ possible classes (Sutter 1992). The $n$ targets are spatially distinct symbols, that is, icons, that each flash at distinct constant
The frequency on a computer monitor or LED display. As the user shifts visual attention to one of the targets, the EEG over the visual cortex exhibits stronger oscillatory components at the target frequency and its harmonics. For this case, the canonical correlation analysis (CCA) algorithm generates an EEG spatial filter for each target that maximizes the correlation between a set of sinusoidal templates at each target frequency and its harmonic frequencies (Bin et al. 2009). The spatial filter associated with one of the target frequencies that produces the highest correlation for a given data window designates the current target of the user’s visual attention. The idea is that the sinusoidal templates corresponding to the target frequency should better match the EEG than the templates at the other frequencies. See Figure 20.2 for a graphical depiction of CCA. This approach has an advantage over standard spectral analysis, for example, techniques based on the Fourier transform, in that it simultaneously combines spatial and spectral information in the classification decision and tends to provide more reliable performance. This approach does not require calibration or training before online operation and allows for continuous, asynchronous operation.

20.3.1 DATA COLLECTION

The SSVEP response is strongest over the occipital lobe, with typical electrode coverage over O1, O2, Oz, PO7, and PO8. In contrast to the P300 paradigm, it is not necessary to synchronize the EEG data with the flashing stimuli for this approach, although it may be necessary for more sophisticated approaches or offline analyses. The sampling rate should be set above at least twice the frequency of the highest harmonic of interest in order to satisfy the Nyquist–Shannon sampling theorem. Here, we assume that the data to be analyzed are loaded into MATLAB in a matrix format of $k$ samples $\times m$ channels.
20.3.2 Data Preprocessing

The data can be band-pass filtered between 0.1 Hz and just above the frequency of the maximum stimulus harmonic of interest. This filter is primarily to eliminate noise outside of the frequency range of interest, which can improve the performance of CCA. Other studies have shown that the performance can be improved by further isolating each harmonic using individual sub-band or comb filters (Chen et al. 2015).

20.3.3 Data Segmentation and Classification

Below is a sample MATLAB function that takes a segment of EEG data (EEGsignal) as an input and outputs a classification decision for that data segment. Additionally, the function outputs the correlation coefficients for each of the classes presented. The CCA reference templates at each stimulus frequency are first created. They consist of sine and cosine segments at the fundamental frequency and numHarm harmonics, creating waveforms consisting of \( k \) samples (matching the temporal length of the EEG segment) for each target frequency listed in the vector stimF. Both sine and cosine templates are used because a linear combination of the two can represent a single sinusoid with arbitrary phase, matching the characteristics of the EEG observation. Typically, only two or three harmonics are needed for an accurate classification, but the number of harmonics can be easily reconfigured to meet the performance needs. For each EEG data segment, CCA is performed for the template corresponding to each target frequency. CCA returns a set of optimized spatial weights for the EEG channels and the CCA sinusoidal templates that maximize the resulting correlation, as well as returning the value of this correlation. The provided function then selects the target frequency corresponding to the CCA template that produced the largest correlation for the given data segment. This process is repeated for each subsequent EEG data segment. The data segments can overlap, and the function can be called using varying durations of collected EEG data. For instance, a classification decision could be made every second while using a sliding buffer of the previous 1.5 s worth of EEG data.

```matlab
function [class,rCoeff] = CCA_Classification(EEGsignal,Fs,stimF,numHarm)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function performs CCA classification for a block of data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Section 1: Create reference signals of sines and cosines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
refSigs = zeros(size(EEGsignal,1),numHarm*2,length(stimF));
t = (1/Fs:1/Fs:size(EEGSignal)/Fs)'; % vector of time points
for i = 1:length(stimF)   % for each stimulus frequency
    for j = 1:numHarm   % and for each harmonic
...
The processing time for performing CCA can be limiting. For this reason, it would be beneficial to pass in the reference templates to the \texttt{CCA\_Classification()} function as an additional input argument. Generating the sine and cosine templates each time \texttt{CCA\_Classification()} is called is unnecessary when using a constant data window length for each classification. A classification decision based simply on the largest correlation coefficient is also not necessarily optimal. For instance, a classification decision can be postponed (and more data collected) until the highest correlation coefficient in \texttt{rCoeff} reaches a threshold, or until \texttt{rCoeff} from one class distinguishes itself significantly from the other classes. It is also possible to achieve improved performance by including user-specific EEG reference templates in addition to the sinusoidal reference templates (Chen et al. 2015); however, this results in an additional calibration phase to collect the EEG reference templates. These considerations exemplify the speed versus accuracy and calibration time trade-offs in the design of an online BCI.

Sample four-class P300 and SSVEP data are available upon e-mail request from the authors.

ACKNOWLEDGMENTS

Dean J. Krusienski was supported in part by the National Science Foundation (NSF) under Grants IIS-1608140, IIS-1421948 and IIS-1451028. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES


